KRONECKER REPRESENTATION AND DECOMPOSITIONAL ANALYSIS OF CLOSED QUEUEING NETWORKS WITH PHASE–TYPE SERVICE DISTRIBUTIONS AND ARBITRARY BUFFER SIZES

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

Akın Meriç

June, 2007

ABSTRACT

KRONECKER REPRESENTATION AND DECOMPOSITIONAL ANALYSIS OF CLOSED QUEUEING NETWORKS WITH PHASE–TYPE SERVICE DISTRIBUTIONS AND ARBITRARY BUFFER SIZES

Akın Meriç
M.S. in Computer Engineering
Supervisor: Assoc. Prof. Dr. Tuğrul Dayar
June, 2007

This thesis extends two approximative fixed–point iterative methods based on decomposition for closed queueing networks (QNs) with Coxian service distributions and arbitrary buffer sizes from the literature to include phase–type service distributions. It shows how the irreducible Markov chain associated with each subnetwork in the decomposition can be represented hierarchically using Kronecker products. The proposed methods are implemented in a software tool, which is capable of computing the steady–state probability vector of each subnetwork by a multilevel method at each fixed–point iteration. The two methods are compared with others, one being the multilevel method for the closed QN itself, for accuracy and efficiency on a number of examples using the tool, and their convergence properties are discussed. Numerical results indicate that there is a niche among the problems considered which is filled by the two approximative fixed–point iterative methods.

Keywords: Closed queueing networks · Phase–type service distributions · Kronecker representations · Network decomposition · Fixed–point iteration · Multilevel methods.

ÖZET

FAZ–TİPLİ HİZMET DAĞILIMLARI VE DEĞİŞİK BÜYÜKLÜKTE BEKLEME YERLERİ OLAN KAPALI KUYRUK AĞLARININ KRONECKER GÖSTERİMLERİ VE BÖLMEYE DAYALI ÇÖZÜMLENMESİ

Akın Meriç
Bilgisayar Mühendisliği, Yüksek Lisans
Tez Yöneticisi: Doç. Dr. Tuğrul Dayar
Haziran, 2007

Bu tez, literatürde bulunan Cox hizmet dağılımlı ve değişik büyüklükte bekleme yerleri olan kapalı kuyruk ağları için ayrıştırmaya dayalı iki yaklaşık sabit nokta öteleme yöntemini, faz–tipli servis dağılımlarını kapsayacak şekilde genişletmektedir. Ayrıştırmadan ortaya çıkan altağların her birine karşı gelen indirgenemeyen Markov zincirinin, Kronecker çarpımlar kullanılarak hiyerarşik olarak nasıl ifade edilebileceğini göstermektedir. Önerilen yöntemler her bir altağın uzun vadeli olasılık vektörünü her sabit nokta ötelemesinde çok seviyeli bir yöntemle hesap edebilen bir yazılım paketinde kodlanmıştır. Yöntemler, çeşitli örnekler üzerinde, biri ayrıştırılmamış kapalı kuyruk ağı için çok seviyeli yöntem olmak üzere, yazılım paketi kullanılarak başkalarıyla doğruluk ve etkinlik bakımından karşılaştırılmış ve yakınsama özellikleri tartışılmıştır. Sayısal sonuçlar, iki yaklaşık sabit öteleme yönteminin dikkate alınan problemler arasında doldurduğu bir boşluk olduğunu göstermiştir.

Anahtar sözcükler: Kapalı kuyruk ağları · Faz–tipli hizmet dağılımları · Kronecker gösterimleri · Ağ ayrıştırması · Sabit nokta ötelemesi · Çok seviyeli yöntemler.

Acknowledgement

First, I would like to thank my supervisor Assoc. Prof. Dr. Tuğrul Dayar for his guidance, support, and patience throughout this study. I acknowledge Jean–Michel Fourneau for bringing Raymond Marie's method to our attention. I would also like to thank my thesis committee members Assoc. Prof. Dr. Nail Akar and for Assist. Prof. Dr. Murat Fadıloğlu for their constructive remarks and suggestions. Finally, I would like to thank my family and friends for their encouragement and moral support throughout this study.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Today, obtaining various performance measures for queueing networks (QNs) exactly (up to computer precision) still remains a challenging problem. Indeed, only a small class of QNs can be solved analytically (and exactly) for their performance measures (see, for instance, [5, 22]). This class of networks is called product form and requires specific conditions on the arrival processes, service processes, service disciplines, and buffer sizes of queues. On the other hand, obtaining exact performance measures for networks of queues with general arrival and service time distributions and arbitrary buffer sizes is not straightforward. The class of closed QNs with phase–type service distributions, first–come first–served (FCFS) service disciplines, and finite buffer sizes considered in this thesis are among this latter kind of networks, since they are not product form and their state spaces grow exponentially with numbers of customers, queues, and phases in each queue. Furthermore, customers in these QNs may be subject to blocking because of the finite buffers of some queues. In this thesis, we assume that the customer subject to blocking at the head of a queue is not lost, but forced to wait until its destination queue's buffer is available to accept the customer. Hence, the number of customers in the class of closed QNs considered remains constant.

For steady–state analysis, one needs to solve the system of linear equations

$$\pi Q = 0, \ \sum_{i \in \mathcal{S}} \pi_i = 1, \tag{1.1}$$

where Q denotes the infinitesimal generator matrix of the irreducible Markov chain (MC) describing exponential transition rates among states of the particular closed QN, $\mathcal{S}$ is the set of states of Q, and π is its steady–state probability distribution (row) vector [23, Ch. 3]. When the infinitesimal generator matrix Q is irreducible, then π in (1.1) exists, is unique, and is positive [37, Ch. 1]. By using Kronecker (or tensor) products [16, 40] of smaller matrices to represent Q (see, for instance, [8, 10]) and by performing vector–Kronecker product multiplications [18] within a multilevel (ML) iterative method [11], it is possible to obtain π without generating Q. This state–of–the–art approach results in important storage savings compared to sparse MC solvers and is generally considered to be the fastest solver for Kronecker structured MCs. For a recent review on analyzing MCs based on Kronecker products, see [17].

Several approximative methods for analyzing the steady–state behavior of closed QNs with arbitrary buffer sizes have been proposed. These methods are based on decomposing the network into a set of subnetworks which satisfy certain properties. These subnetworks are analyzed in isolation to obtain marginal (or conditional) performance measures. This approach can be very efficient when the isolated subnetworks are simple to analyze and weakly coupled. Some methods are in the form of iterative aggregation–disaggregation [9, 11, 12], while there are others which force almost exact aggregation for product form QNs with arbitrary buffer sizes [3, 19, 26, 31, 45]. Some methods can be applied to networks with exponentially distributed service rates [1, 19, 31, 38] and some others can be applied to networks with phase–type service distributions [3, 26, 45]. The decomposition procedure introduces the first level of error while computing various performance measures for closed QNs with phase–type service distributions and arbitrary buffer sizes. QNs with phase–type service distributions can also be analyzed by methods that assume exponential service distributions. Yet, this introduces another level of error, because mean service rates of phase–type service distributions are used as if they were exponential service rates. In this respect, approximative methods for QNs with phase–type service distributions and arbitrary buffer sizes introduce less error and are more suitable for obtaining various performance measures.

Before discussing the previous work on decompositional analysis of closed QNs, we mention two methods geared toward open QNs. In 1979, Kühn [24] developed an approximation method for large open FCFS QNs with general service distributions and infinite buffer sizes. In 1983, the method is extended by Whitt [42] and implemented in a software package called Queueing Network Analyzer (QNA). The method decomposes QN into one queue subnetworks and analyzes these subnetworks individually. Subnetworks are related to their environment by arrival and service processes which are assumed to be renewal processes characterized by their first two moments. Numerical experiments show that the method accuracy highly depends on the coefficient of variation of the arrival and service processes.

In 1987, particularly for manufacturing flow line systems, Gershwin [21] presented an algorithm to approximate throughputs of open tandem QNs with finite buffers and blocking in which the service time of queues are determined by failure or repair durations. The algorithm decomposes the QN into subnetworks of individual queues and determines the parameters of subnetworks using relations among the flows through the buffers of the original QN. Numerical experiments indicate that the algorithm approximates throughput values with a maximum relative error of 1% for QNs with three queues and with a maximum relative error of 3% for a larger number of queues. A review of manufacturing flow line system models can be found in [15]. Clearly, these two methods are not the only ones for open QNs in the literature, but now we turn to the subject of interest in this thesis.

It was shown in 1967 by Gordon and Newell [22] that closed QNs with exponential service distributions and infinite buffer sizes have product form solution. Thus, the steady–state distribution of such networks can be computed analytically using normalization constants exactly. In 1973, Buzen [13] devised a method known as the convolution algorithm to efficiently compute the normalization constants. Although the convolution algorithm can be used as an approximative method for computing performance measures of closed QNs with blocking, there are various methods proposed in the literature specifically for closed QNs with blocking and the ones related to the subject of this thesis are briefly reviewed next.

In 1979, Marie [26] proposed an approximation algorithm based on network decomposition to obtain the marginal steady–state distributions of a closed QN with Coxian service distributions and arbitrary buffer sizes. Marie's method decomposes the closed QN into a collection of subnetworks where the transition probabilities between subnetworks are independent of the states of the subnetworks. Thus, each subnetwork is considered as an exponential service station with load–dependent service rate for which the parameters of the equivalent server are obtained by analyzing the subnetwork in isolation under state–dependent Poisson arrivals. Then the approximate results are obtained via a fixed–point iteration scheme. Numerical results for examples in [26] show that the method presents a maximum relative error of 1% for throughput values and presents a maximum relative error of 7% for mean queue length values. Although Marie's method yields highly accurate results, a drawback of the method is that it analyzes the subnetworks numerically which can be a time consuming task for large networks.

In 1986, Suri and Diehl [38] introduced a variable buffer size decomposition method. The method can be applied to closed QNs with blocking before service and which have one queue with infinite capacity. In order to approximate throughput values of the QN, the method uses a network decomposition principle applied to nested subnetworks. The approximate throughput of a queue is computed by aggregating all the downstream queues in the network to a single composite queue and analyzing the queue–composite queue pair. Although numerical experiments present a relative error less that 7%, it is pointed out in [4] that the algorithm cannot produce accurate results for QNs with more than 4 queues.

In 1986, Yao and Buzacott [45] proposed an approximation algorithm for closed QNs with Coxian service distributions and arbitrary buffer sizes. Their method decomposes the network into individual queues and approximates the service distributions of each queue by an exponential distribution with the same mean as the original Coxian server. Experimental results provide a maximum relative error of 2% for throughput values. It is indicated that the method should be mostly adequate when applied to closed QNs with a moderate number of queues and customers.

In 1988, Akyildiz [1] developed an approximation algorithm for the throughput of closed QNs with exponential service distributions. The idea behind his approximation algorithm is that the throughput of a blocking closed QN is approximately the same as an equivalent nonblocking closed QN which has product form queue length distribution. In that respect, the number of customers of the equivalent closed QN without blocking is chosen such that the number of states of the closed QN with blocking is close to the number of states of the closed QN without blocking. The QN under consideration is assumed to be deadlock free, and if blocking occurs, then customers will face blocking after service. Akyildiz's method can produce throughput values with relative error smaller than 2% for closed QNs with blocking and exponential service distributions. Yet, it is unable to produce accurate results for other performance measures or for networks with phase–type service distributions. Akyildiz [2] also proposed mean value analysis for analyzing closed QNs with blocking after service. The proposed method is based on the arrival theorem and extends the classical mean value analysis of Reiser and Lavenberg [33] to include finite queues.

In 1988, Perros et al. [31] proposed a numerical procedure for analyzing closed QNs with exponential service distributions and arbitrary buffer sizes, where the blocking mechanism is defined as blocking after service. The approximation procedure is based on Norton's theorem (see [32]). The closed QN is decomposed into two subnetworks where the queues with infinite buffer sizes are grouped in one subnetwork and the queues that are liable to blocking are grouped in the other. Then the subnetworks with blocking queues is analyzed using throughput values obtained from the nonblocking subnetwork. Numerical experiments show that the approximation procedure yields relative errors less than 1% for throughput and mean queue length values.

In 1989, Onvural and Perros [29] proposed a method to approximate throughput values of closed exponential QNs with blocking. Their method approximates throughput values of the QN using the fact that throughput is a symmetrical function which depends on the finite population in the QN [4, 28]. It can be used with blocking before service and blocking after service blocking mechanisms, and it assumes that deadlocks are resolved by instantaneously exchanging the blocked customers. A drawback of this method is that it is only capable of approximating throughput values for closed exponential QNs with blocking.

Also in 1989, Frein and Dallery [19] presented an approximation method for cyclic closed QNs with arbitrary buffer sizes and exponential service distributions. In their method, the closed QN considered is decomposed into individual queues and the solution process is defined by a fixed–point iteration scheme in which each individual queue is analyzed as an $M/M/1/c/K$ queue, where c is the buffer size of the queue and K is the finite population size. The method yields a maximum relative error of 7% for throughput values and a maximum relative error of 20% for mean queue length values.

In 1989, Altiok [3] proposed an approximation method for closed tandem QNs with phase–type service distributions and arbitrary buffer sizes. Altiok's method decomposes the network into individual queues. Each queue is approximated by an $M/PH/1/c/K$ queue with appropriately chosen phase–type service distribution and arrival rate. Then, the steady–state distribution of queues are computed using an iterative algorithm. Results show that the method yields a maximum relative error of 7% for throughput values and a maximum relative error of 10% for mean queue length values.

In 2000, Vroblefski, Ramesh and Zionts [41] reported an approximation method for closed tandem QNs with arbitrary buffer sizes and state–dependent exponential servers. In their approach, the network is decomposed into subnetworks where each subnetwork consists of a virtual synchronization station preceding a queue. Then, the throughput values are approximated using an iterative scheme in which the subsystems are analyzed independently as an open QN. Numerical experiments conducted under a variety of blocking mechanisms report approximate throughput values to be within at most 6 percent of simulation results.

In this thesis, we extend two approximative fixed–point iterative methods based on decomposition for closed QNs with Coxian service distributions and arbitrary buffer sizes from the literature to include phase–type service distributions. These are Marie's (M) method [26] and Yao and Buzacott's (YB) method [45]. We show how the irreducible MC associated with each subnetwork in the decomposition can be represented hierarchically using Kronecker products. The decompositional nature of the methods imply an additive dimension of scalability. The Kronecker representation of each subnetwork model in the decomposition facilitates yet another form of compactness and a multiplicative dimension of scalability. Since, the methods are already approximative by construction, the closed QN model becomes essentially more compact with the Kronecker representation.

The proposed methods are implemented in a software tool [27], which is capable of computing the steady–state vector of each subnetwork by the ML method at each fixed–point iteration. The methods of M and YB are compared with the ML and successive over–relaxation (SOR) [39] methods for the closed QN itself and with the convolution algorithm (CA) [13] and Akyildiz's mean value analysis (MVABLO) [2], for accuracy and efficiency on a number of examples using the tool. The reason behind using CA and MVABLO is that these methods are approximative analytical methods unlike the methods of M and YB and need almost no computational effort. Hence, this comparison may reveal when it is worthwhile to use approximative iterative methods of M and YB. SOR is included in order to make a comparison with the ML method.

In section 2, we provide the Kronecker representation of the class of closed QNs considered and briefly explain the ML method. In section 3, we discuss the methods of CA, MVABLO, M, and YB. Therein, it is shown how the subnetworks obtained by the decomposition of the closed QN model in the methods of M and YB are represented using Kronecker products. In section 4, we give an upper bound on the number of floating point operations performed in the methods and analyze the methods of M and YB for existence of a fixed–point. In section 5, we briefly discuss on some issues concerning implementation of the software tool. In section 6, we present the results of numerical experiments, and in section 7, we conclude.

Chapter 2

Kronecker Representation and ML Method

In this chapter, we provide a brief overview of Kronecker algebra and give a formal definition of the closed QN model used. An small example is also included in order to clarify the discussion. Then we discuss the ML method used in solving MCs expressed in terms of Kronecker products.

Throughout the text, we denote matrices by upper–case letters, a block of a matrix by specifying the indices of the block in parentheses beside the matrix name, and an element of a matrix by specifying the indices of the element as subscripts of the lower–case matrix name. Rows and columns of matrices representing the evolution of queues are numbered starting from one.

We first define the Kronecker product and Kronecker sum operations [16, 40].

Definition 2.1. *Given two (rectangular) matrices $A \in I\!R^{r_A \times c_A}$ and $B \in I\!R^{r_B \times c_B}$ as in*

$$A = \begin{pmatrix} a_{1,1} & a_{1,2} & \dots & a_{1,c_A} \\ a_{2,1} & a_{2,2} & \dots & a_{2,c_A} \\ \vdots & \vdots & \ddots & \vdots \\ a_{r_A,1} & a_{r_A,2} & \dots & a_{r_A,c_A} \end{pmatrix} \quad \text{and} \quad B = \begin{pmatrix} b_{1,1} & b_{1,2} & \dots & b_{1,c_B} \\ b_{2,1} & b_{2,2} & \dots & b_{2,c_B} \\ \vdots & \vdots & \ddots & \vdots \\ b_{r_B,1} & b_{r_B,2} & \dots & b_{r_B,c_B} \end{pmatrix},$$

*their **tensor product**, written as $C = A \otimes B$ with $C \in I\!R^{r_A r_B \times c_A c_B}$, is defined as*

$$C = \begin{pmatrix} a_{1,1}B & a_{1,2}B & \dots & a_{1,c_A}B \\ a_{2,1}B & a_{2,2}B & \dots & a_{2,c_A}B \\ \vdots & \vdots & \ddots & \vdots \\ a_{r_A,1}B & a_{r_A,2}B & \dots & a_{r_A,c_A}B \end{pmatrix}.$$

Definition 2.2. *Given two square matrices $A \in I\!R^{r_A \times r_A}$ and $B \in I\!R^{r_B \times r_B}$, their **tensor sum**, written as $C = A \oplus B$ with $C \in I\!R^{r_A r_B \times r_A r_B}$, is defined in terms of two Kronecker products as*

$$C = A \otimes I_{r_B} + I_{r_A} \otimes B,$$

where I_{r_A} and I_{r_B} denote identity matrices of order r_A and r_B, respectively.

Some important properties of Kronecker algebra for matrices with appropriate dimensions, which we will be using, are given as follows:

1. Associativity:
   $$A \otimes (B \otimes C) = (A \otimes B) \otimes C \quad \text{and} \quad A \oplus (B \oplus C) = (A \oplus B) \oplus C.$$
2. Distributivity over matrix addition:
   $$(A + B) \otimes (C + D) = A \otimes C + B \otimes C + A \otimes D + B \otimes D.$$
3. Compatibility with matrix multiplication:
   $$(A \times B) \otimes (C \times D) = (A \otimes C) \times (B \otimes D).$$

It is important to note that these properties can be extended to cover Kronecker products and sums including multiple operands [16, 40]. In that sense, Kronecker algebra becomes an important tool to represent the behavior of interacting systems mathematically as we discuss in the next section.

2.1 Kronecker Representation

We consider a class of closed FCFS QNs with arbitrary buffer sizes and phase–type service distributions defined by J queues, K customers, routing probability matrix P, phase–type distribution $(\alpha^{(j)}, T^{(j)})$, where $T^{(j)}$ is the phase–type distribution matrix of order $t^{(j)}$ and $\alpha^{(j)}$ is the initial probability distribution row vector of length $t^{(j)}$ associated with $T^{(j)}$, and buffer size b_j for queue $j \in \{1, 2, \ldots, J\}$. We let $c_j = \min\{K, b_j\}$ and the state of queue j be represented by the ordered pair $i_j = (n_j, \phi_j)$, where $n_j \in \{0, 1, \ldots, c_j\}$ denotes the occupancy of queue j and $\phi_j \in \{0, 1, \ldots, t^{(j)} - 1\}$ denotes the phase of its service process, with the constraint that $\phi_j = 0$ when $n_j = 0$ (that is, phase is irrelevant when the queue is empty). Then $i_j \in \{(0, 0)\} \cup \{1, 2, \ldots, c_j\} \times \{0, 1, \ldots, t^{(j)} - 1\}$. We remark that in our model, an arrival to a destination queue can only take place when the destination queue has space for the arriving customer; otherwise the transition is inhibited. The implication of this assumption is that a customer will remain in the server until space becomes available in the destination queue. Observe that this assumption may be replaced with a more accurate approximation for acyclic phase–type distributions [23, p. 57] by adding one more phase with a relatively large transition rate to the service process.

The irreducible MC representing the evolution of queue $j \in \{1, 2, \ldots, J\}$ is a $(c_j + 1) \times (c_j + 1)$ block tridiagonal matrix and is given by $Q^{(j)} = G^{(j)} + D^{(j)}$ (see, for instance, [8]), where

$$G^{(j)} = \begin{pmatrix} O^{(j)}(0,0) & \lambda_j(0)A^{(j)}(0,1) & & & \\ S^{(j)}(1,0) & O^{(j)}(1,1) & \lambda_j(1)A^{(j)}(1,2) & & \\ \ddots & \ddots & \ddots & & \\ & S^{(j)}(c_j-1,c_j-2) & O^{(j)}(c_j-1,c_j-1) & \lambda_j(c_j-1)A^{(j)}(c_j-1,c_j) & \\ & & S^{(j)}(c_j,c_j-1) & O^{(j)}(c_j,c_j) & \end{pmatrix},$$

$\overline{T}^{(j)} = -T^{(j)}e$, e is the column vector of ones of appropriate length, $O^{(j)}(n_j,n_j) = T^{(j)}$ for $n_j \in \{1,\ldots,c_j\}$, $O^{(j)}(0,0) = 0$, $S^{(j)}(n_j,n_j-1) = \overline{T}^{(j)}\alpha^{(j)}$ for $n_j \in \{2,\ldots,c_j\}$, $S^{(j)}(1,0) = \overline{T}^{(j)}$, $A^{(j)}(n_j,n_j+1) = I_{t^{(j)}}$ for $n_j \in \{1,\ldots,c_{j-1}\}$, $A^{(j)}(0,1) = \alpha^{(j)}$, $\lambda_j(n_j)$ is the rate of arrivals to queue j under buffer occupancy n_j, and $D^{(j)}$ is the diagonal correction matrix summing the rows of $Q^{(j)}$ to zero. The upper–diagonal blocks $G^{(j)}(n_j,n_j+1)$ and lower–diagonal blocks $G^{(j)}(n_j,n_j-1)$ of $G^{(j)}$ represent the service completions and arrivals of customers, respectively. Its diagonal blocks $G^{(j)}(n_j,n_j)$ represent phase changes. The boundary level has a single state, while the other levels each have $t^{(j)}$ states. Hence, $G^{(j)}$ is a $(t^{(j)}c_j+1) \times (t^{(j)}c_j+1)$ matrix.

Assuming that the states of the irreducible MC underlying the closed QN are represented as $i = (i_1, i_2, \ldots, i_J)$, let us us define the mapping $f : \mathcal{S} \to \mathcal{N}$ as

$$\begin{aligned} f(i) = f((i_1,i_2,\ldots,i_J)) &= f(((n_1,\phi_1),(n_2,\phi_2),\ldots,(n_J,\phi_J))) \\ &= (n_1,n_2,\ldots,n_J) = n \qquad (2.1) \end{aligned}$$

for $i \in \mathcal{S}$ (see (1.1)) and $n = (n_1,n_2,\ldots,n_J) \in \mathcal{N}$. This mapping is onto and partitions $\mathcal{S}$ into equivalence classes. The set of equivalence classes defined by f is denoted as $\mathcal{N}$ and has cardinality $|\mathcal{N}|$. We remark that $n \in \mathcal{N}$ is represented using the row vector $(n_1,n_2,\ldots,n_J)$, which satisfies $\sum_{j=1}^{J} n_j = K$.

When queues are interconnected to form a closed QN, the arrival rate of customers to queue $j \in \{1,2,\ldots,J\}$, that is, $\lambda_j(n_j)$, depends on column j of the routing probability matrix P, the states of the queues corresponding to nonzero elements in that column, and the rates by which they complete the last phases of their service processes. Assuming that we associate the lexicographical order with the states in $\mathcal{N}$, then the generator matrix Q of a closed QN can be expressed as an $|\mathcal{N}| \times |\mathcal{N}|$ block matrix with block (n, m) for $n, m \in \mathcal{N}$ given by [8, p. 66]

$$Q(n,m) = \begin{cases} Q^{\{j,k\}}(n,m), & n \neq m \text{ and } m = n - e_j^T + e_k^T \\ D(n,n) + Q(n,n)_D + \sum_{j=1}^{J} Q^{\{j,j\}}(n,n), & n = m \\ 0 & \text{otherwise} \end{cases}, \tag{2.2}$$

where $j, k \in \{1, 2, \ldots, J\}$, e_j is column j of the identity matrix, $m = n - e_j^T + e_k^T$ refers to service completion at queue j and arrival to queue k when in state n so as to make a transition to state m, $D(n,n)$ is the diagonal matrix summing block n of rows in Q to zero,

$$Q^{\{j,k\}}(n,m) = \begin{cases} p_{j,k}(I_{c_j^{\{j,k\}}} \otimes S^{(j)}(n_j,m_j) \otimes I_{r_j^{\{j,k\}}})(I_{c_k^{\{j,k\}}} \otimes A^{(k)}(n_k,m_k) \otimes I_{r_k^{\{j,k\}}}), & j < k \\ p_{j,k}(I_{c_k^{\{j,k\}}} \otimes A^{(k)}(n_k,m_k) \otimes I_{r_k^{\{j,k\}}})(I_{c_j^{\{j,k\}}} \otimes S^{(j)}(n_j,m_j) \otimes I_{r_j^{\{j,k\}}}), & j > k \\ p_{j,j}(I_{c_j^{\{j,j\}}} \otimes S^{(j)}(n_j,n_j-1))(A^{(j)}(n_j,n_j+1) \otimes I_{r_j^{\{j,j\}}}), & j = k \end{cases},$$

$$Q(n,n)_D = \bigoplus_{j=1}^{J} O^{(j)}(n_j,n_j) = \sum_{j=1}^{J} I_{c_j^{\{j,j\}}} \otimes O^{(j)}(n_j,n_j) \otimes I_{r_j^{\{j,j\}}},$$

where $c_x^{\{j,k\}} = \prod_{z=1}^{x-1} size^{\{j,k\}}(z)$ and $r_x^{\{j,k\}} = \prod_{z=x+1}^{J} size^{\{j,k\}}(z)$ represent product of column and row sizes of matrices respectively, and

$$size^{\{j,k\}}(z) = \begin{cases} \#\_of\_rows(O^{(z)}(n_z,n_z)), & z \neq j \text{ and } z \neq k \\ \#\_of\_rows(A^{(k)}(n_k,m_k)), & z = k \text{ and } k > j \\ \#\_of\_cols(A^{(k)}(n_k,m_k)), & z = k \text{ and } k < j \\ \#\_of\_rows(S^{(j)}(n_j,m_j)), & z = j \text{ and } j > k \\ \#\_of\_cols(S^{(j)}(n_j,m_j)), & z = j \text{ and } j < k \end{cases}. \tag{2.3}$$

Example 1.

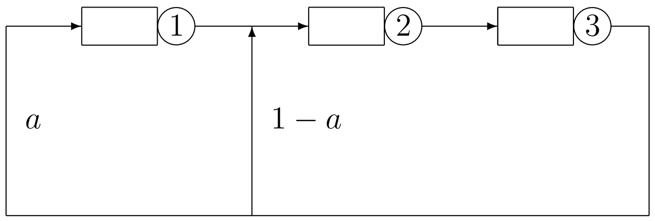

Figure 2.1: A closed QN.

Consider the closed QN in Figure 2.1, which consists of three queues, two customers, and the routing probability matrix

$$P = \begin{array}{c|ccc} & 1 & 2 & 3 \\ \hline 1 & 0 & 1 & 0 \\ 2 & 0 & 0 & 1 \\ 3 & a & 1-a & 0 \end{array}$$

with $0 < a < 1$. Hence, $N = 3$ and $K = 2$, the service distributions and buffer sizes of queues are given by

$$\begin{aligned} T^{(1)} &= \begin{pmatrix} -\mu_1^{(1)} \end{pmatrix}, & \alpha^{(1)} &= (1), & b_1 &= 2, \\ T^{(2)} &= \begin{pmatrix} -\mu_1^{(2)} & \mu_1^{(2)} \\ 0 & -\mu_2^{(2)} \end{pmatrix}, & \alpha^{(2)} &= (1,0), & b_2 &= 2, \\ T^{(3)} &= \begin{pmatrix} -\mu_1^{(3)} & \mu_1^{(3)} \\ 0 & -\mu_2^{(3)} \end{pmatrix}, & \alpha^{(3)} &= (1,0), & b_3 &= 1. \end{aligned}$$

Queue 1 has an exponential service distribution, queues 2 and 3 have hypoexponential service distributions with capacities $c_1 = c_2 = 2$ and $c_3 = 1$.

Table 2.1: State space $\mathcal{S}$ versus set of equivalence classes $\mathcal{N}$ for Example 1.

$\mathcal{S}$	$\mathcal{N}$
$((\underline{0},0),(\underline{1},1),(\underline{1},1))$, $((\underline{0},0),(\underline{1},1),(\underline{1},2))$, $((\underline{0},0),(\underline{1},2),(\underline{1},1))$, $((\underline{0},0),(\underline{1},2),(\underline{1},2))$	(0,1,1)
$((\underline{0},0),(\underline{2},1),(\underline{0},0))$, $((\underline{0},0),(\underline{2},2),(\underline{0},0))$	(0,2,0)
$((\underline{1},1),(\underline{0},0),(\underline{1},1))$, $((\underline{1},1),(\underline{0},0),(\underline{1},2))$	(1,0,1)
$((\underline{1},1),(\underline{1},1),(\underline{0},0))$, $((\underline{1},1),(\underline{1},2),(\underline{0},0))$	(1,1,0)
$((\underline{2},1),(\underline{0},0),(\underline{0},0))$	(2,0,0)

For this example, $\mathcal{N}$ is obtained from $\mathcal{S}$ using (2.1) as in Table 2.1. The state space S consists of 11 states, which are divided into 5 equivalence classes with cardinalities 4, 2, 2, 2, 1. Observe that $p_{1,1} = p_{2,2} = p_{3,3} = 0$. This implies that the third (summation) term in (2.2) for $n = m$ evaluates to zero, since $Q^{\{j,j\}}(n,n) = 0$ for $j \in \{1,2,3\}$ and $n \in \mathcal{N}$. Blocks of Q are computed from (2.2) as follows and diagonal elements of Q are represented using $*$'s.

$$\begin{aligned}
Q((0,1,1),(0,1,1)) &= D((0,1,1),(0,1,1)) + Q((0,1,1),(0,1,1))_D \\
&\quad + \sum_{j=1}^{3} Q^{\{j,j\}}((0,1,1),(0,1,1)) \\
&= D((0,1,1),(0,1,1)) + I_{c_1^{\{1,1\}}} \otimes O^{(1)}(0,0) \otimes I_{r_1^{\{1,1\}}} \\
&\quad + I_{c_2^{\{2,2\}}} \otimes O^{(2)}(1,1) \otimes I_{r_2^{\{2,2\}}} + I_{c_3^{\{3,3\}}} \otimes O^{(3)}(1,1) \otimes I_{r_3^{\{3,3\}}} \\
&= D((0,1,1),(0,1,1)) + (0) \otimes I_4 \\
&\quad + I_1 \otimes \begin{pmatrix} -\mu_1^{(2)} & \mu_1^{(2)} \\ 0 & -\mu_2^{(2)} \end{pmatrix} \otimes I_2 + I_2 \otimes \begin{pmatrix} -\mu_1^{(3)} & \mu_1^{(3)} \\ 0 & -\mu_2^{(3)} \end{pmatrix} \\
&= D((0,1,1),(0,1,1)) + \begin{pmatrix} -\mu_1^{(2)} & 0 & \mu_1^{(2)} & 0 \\ 0 & -\mu_1^{(2)} & 0 & \mu_1^{(2)} \\ 0 & 0 & -\mu_2^{(2)} & 0 \\ 0 & 0 & 0 & -\mu_2^{(2)} \end{pmatrix} \\
&\quad + \begin{pmatrix} -\mu_1^{(3)} & \mu_1^{(3)} & 0 & 0 \\ 0 & -\mu_2^{(3)} & 0 & 0 \\ 0 & 0 & -\mu_1^{(3)} & \mu_1^{(3)} \\ 0 & 0 & 0 & -\mu_2^{(3)} \end{pmatrix} \\
&= \begin{pmatrix} * & \mu_1^{(3)} & \mu_1^{(2)} & 0 \\ 0 & * & 0 & \mu_1^{(2)} \\ 0 & 0 & * & \mu_1^{(3)} \\ 0 & 0 & 0 & * \end{pmatrix}.
\end{aligned}$$

$$\begin{aligned}
Q((0,1,1),(0,2,0)) &= Q^{\{3,2\}}((0,1,1),(0,2,0)) \\
&= p_{3,2}(I_{c_2^{\{3,2\}}} \otimes A^{(2)}(1,2) \otimes I_{r_2^{\{3,2\}}})(I_{c_3^{\{3,2\}}} \otimes S^{(3)}(1,0) \otimes I_{r_3^{\{3,2\}}}) \\
&= p_{3,2}\left(I_1 \otimes \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \otimes I_2\right)\left(I_2 \otimes \begin{pmatrix} 0 \\ \mu_2^{(3)} \end{pmatrix}\right) \\
&= (1-a)(I_4)\begin{pmatrix} 0 & 0 \\ \mu_2^{(3)} & 0 \\ 0 & 0 \\ 0 & \mu_2^{(3)} \end{pmatrix} \\
&= \begin{pmatrix} 0 & 0 \\ (1-a)\mu_2^{(3)} & 0 \\ 0 & 0 \\ 0 & (1-a)\mu_2^{(3)} \end{pmatrix}.
\end{aligned}$$

$$\begin{aligned}
Q((0,1,1),(1,1,0)) &= Q^{\{3,1\}}((0,1,1),(1,1,0)) \\
&= p_{3,1}(I_{c_1^{\{3,1\}}} \otimes A^{(1)}(0,1) \otimes I_{r_1^{\{3,1\}}})(I_{c_3^{\{3,1\}}} \otimes S^{(3)}(1,0) \otimes I_{r_3^{\{3,1\}}}) \\
&= p_{3,1}(A^{(1)}(0,1) \otimes I_{r_1^{\{3,1\}}})(I_{c_3^{\{3,1\}}} \otimes S^{(3)}(1,0)) \\
&= a((1) \otimes I_4)\left(I_2 \otimes \begin{pmatrix} 0 \\ \mu_2^{(3)} \end{pmatrix}\right) \\
&= \begin{pmatrix} 0 & 0 \\ a\mu_2^{(3)} & 0 \\ 0 & 0 \\ 0 & a\mu_2^{(3)} \end{pmatrix}.
\end{aligned}$$

$$\begin{aligned}
Q((0,2,0),(0,1,1)) &= Q^{\{2,3\}}((0,2,0),(0,1,1)) \\
&= p_{2,3}(I_{c_2^{\{2,3\}}} \otimes S^{(2)}(2,1) \otimes I_{r_2^{\{2,3\}}})(I_{c_3^{\{2,3\}}} \otimes A^{(3)}(0,1) \otimes I_{r_3^{\{2,3\}}}) \\
&= p_{2,3}\left(I_1 \otimes \begin{pmatrix} 0 & 0 \\ \mu_2^{(2)} & 0 \end{pmatrix} \otimes I_1\right)(I_2 \otimes (1 \;\; 0)) \\
&= \begin{pmatrix} 0 & 0 \\ \mu_2^{(2)} & 0 \end{pmatrix}\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix} \\
&= \begin{pmatrix} 0 & 0 & 0 & 0 \\ \mu_2^{(2)} & 0 & 0 & 0 \end{pmatrix}.
\end{aligned}$$

$$\begin{aligned}
Q((0,2,0),(0,2,0)) &= D((0,2,0),(0,2,0)) + Q((0,2,0),(0,2,0))_D \\
&\quad + \sum_{j=1}^{3} Q^{\{j,j\}}((0,2,0),(0,2,0)) \\
&= D((0,2,0),(0,2,0)) + I_{c_1^{\{1,1\}}} \otimes O^{(1)}(0,0) \otimes I_{r_1^{\{1,1\}}} \\
&\quad + I_{c_2^{\{2,2\}}} \otimes O^{(2)}(2,2) \otimes I_{r_2^{\{2,2\}}} + I_{c_3^{\{3,3\}}} \otimes O^{(3)}(0,0) \otimes I_{r_3^{\{3,3\}}} \\
&= D((0,2,0),(0,2,0)) + (0) \otimes I_2 \\
&\quad + I_1 \otimes \begin{pmatrix} -\mu_1^{(2)} & \mu_1^{(2)} \\ 0 & -\mu_2^{(2)} \end{pmatrix} \otimes I_1 + I_2 \otimes (0) \\
&= \begin{pmatrix} * & \mu_1^{(2)} \\ 0 & * \end{pmatrix}.
\end{aligned}$$

$$\begin{aligned}
Q((1,0,1),(0,1,1)) &= Q^{\{1,2\}}((1,0,1)(0,1,1)) \\
&= p_{1,2}(I_{c_1^{\{1,2\}}} \otimes S^{(1)}(1,0) \otimes I_{r_1^{\{1,2\}}})(I_{c_2^{\{1,2\}}} \otimes A^{(2)}(0,1) \otimes I_{r_2^{\{1,2\}}}) \\
&= p_{1,2}\left((\mu_1^{(1)}) \otimes (1 \ \ 0) \otimes I_2\right) \\
&= \begin{pmatrix} \mu_1^{(1)} & 0 & 0 & 0 \\ 0 & \mu_1^{(1)} & 0 & 0 \end{pmatrix}.
\end{aligned}$$

$$\begin{aligned}
Q((1,0,1),(1,0,1)) &= D((1,0,1),(1,0,1)) + Q((1,0,1),(1,0,1))_D \\
&\quad + \sum_{j=1}^{3} Q^{\{j,j\}}((1,0,1),(1,0,1)) \\
&= D((1,0,1),(1,0,1)) + I_{c_1^{\{1,1\}}} \otimes O^{(1)}(1,1) \otimes I_{r_1^{\{1,1\}}} \\
&\quad + I_{c_2^{\{2,2\}}} \otimes O^{(2)}(0,0) \otimes I_{r_2^{\{2,2\}}} + I_{c_3^{\{3,3\}}} \otimes O^{(3)}(1,1) \otimes I_{r_3^{\{3,3\}}} \\
&= D((1,0,1),(1,0,1)) + (0) \otimes I_2 \\
&\quad + I_1 \otimes (0) \otimes I_2 + I_1 \otimes \begin{pmatrix} -\mu_1^{(3)} & \mu_1^{(3)} \\ 0 & -\mu_2^{(3)} \end{pmatrix} \\
&= \begin{pmatrix} * & \mu_1^{(3)} \\ 0 & * \end{pmatrix}.
\end{aligned}$$

$$\begin{aligned}
Q((1,0,1),(1,1,0)) &= Q^{\{3,2\}}((1,0,1),(1,1,0)) \\
&= p_{3,2}(I_{c_2^{\{3,2\}}} \otimes A^{(2)}(0,1) \otimes I_{r_2^{\{3,2\}}})(I_{c_3^{\{3,2\}}} \otimes S^{(3)}(1,0) \otimes I_{r_3^{\{3,2\}}}) \\
&= (1-a)(I_1 \otimes (1 \;\; 0) \otimes I_2)\left(I_2 \otimes \begin{pmatrix} 0 \\ \mu_2^{(3)} \end{pmatrix}\right) \\
&= (1-a)\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{pmatrix}\begin{pmatrix} 0 & 0 \\ \mu_2^{(3)} & 0 \\ 0 & 0 \\ 0 & \mu_2^{(3)} \end{pmatrix} \\
&= \begin{pmatrix} 0 & 0 \\ (1-a)\mu_2^{(3)} & 0 \end{pmatrix}.
\end{aligned}$$

$$\begin{aligned}
Q((1,0,1),(2,0,0)) &= Q^{\{3,1\}}((1,0,1),(2,0,0)) \\
&= p_{3,1}(I_{c_1^{\{3,1\}}} \otimes A^{(1)}(1,2) \otimes I_{r_1^{\{3,1\}}})(I_{c_3^{\{3,1\}}} \otimes S^{(3)}(1,0) \otimes I_{r_3^{\{3,1\}}}) \\
&= a\ ((1) \otimes I_2)\left(I_1 \otimes \begin{pmatrix} 0 \\ \mu_2^{(3)} \end{pmatrix}\right) \\
&= \begin{pmatrix} 0 \\ a\mu_2^{(3)} \end{pmatrix}.
\end{aligned}$$

$$\begin{aligned}
Q((1,1,0),(0,2,0)) &= Q^{\{1,2\}}((1,1,0)(0,2,0)) \\
&= p_{1,2}(I_{c_1^{\{1,2\}}} \otimes S^{(1)}(1,0) \otimes I_{r_1^{\{1,2\}}})(I_{c_2^{\{1,2\}}} \otimes A^{(2)}(1,2) \otimes I_{r_2^{\{1,2\}}}) \\
&= ((\mu_1^{(1)}) \otimes I_2)\left(I_1 \otimes \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}\right) \\
&= \begin{pmatrix} \mu_1^{(1)} & 0 \\ 0 & \mu_1^{(1)} \end{pmatrix}.
\end{aligned}$$

$$\begin{aligned}
Q((1,1,0),(1,0,1)) &= Q^{\{2,3\}}((1,1,0)(1,0,1)) \\
&= p_{2,3}(I_{c_2^{\{2,3\}}} \otimes S^{(2)}(1,0) \otimes I_{r_2^{\{2,3\}}})(I_{c_3^{\{2,3\}}} \otimes A^{(3)}(0,1) \otimes I_{r_3^{\{2,3\}}}) \\
&= \left(\begin{pmatrix} 0 \\ \mu_2^{(2)} \end{pmatrix} \otimes I_1\right)(I_1 \otimes (1 \;\; 0)) \\
&= \begin{pmatrix} 0 & 0 \\ \mu_2^{(2)} & 0 \end{pmatrix}.
\end{aligned}$$

$$\begin{aligned}
Q((1,1,0),(1,1,0)) &= D((1,1,0),(1,1,0)) + Q((1,1,0),(1,1,0))_D \\
&\quad + \sum_{j=1}^{3} Q^{\{j,j\}}((1,1,0),(1,1,0)) \\
&= D((1,1,0),(1,1,0)) + I_{c_1^{\{1,1\}}} \otimes O^{(1)}(1,1) \otimes I_{r_1^{\{1,1\}}} \\
&\quad + I_{c_2^{\{2,2\}}} \otimes O^{(2)}(1,1) \otimes I_{r_2^{\{2,2\}}} \\
&\quad + I_{c_3^{\{3,3\}}} \otimes O^{(3)}(0,0) \otimes I_{r_3^{\{3,3\}}} \\
&= D((1,1,0),(1,1,0)) + (0) \otimes I_2 + I_1 \otimes \begin{pmatrix} -\mu_1^{(2)} & \mu_1^{(2)} \\ 0 & -\mu_1^{(2)} \end{pmatrix} \otimes I_1 \\
&\quad + I_2 \otimes (0) \\
&= \begin{pmatrix} * & \mu_1^{(2)} \\ 0 & * \end{pmatrix}.
\end{aligned}$$

$$\begin{aligned}
Q((2,0,0),(1,1,0)) &= Q^{\{1,2\}}((2,0,0),(1,1,0)) \\
&= p_{1,2}(I_{c_1^{\{1,2\}}} \otimes S^{(1)}(2,1) \otimes I_{r_1^{\{1,2\}}})(I_{c_2^{\{1,2\}}} \otimes A^{(2)}(0,1) \otimes I_{r_2^{\{1,2\}}}) \\
&= (\mu_1^{(1)} \otimes I_1)(I_1 \otimes (1 \;\; 0)) \\
&= (\mu_1^{(1)} \;\; 0).
\end{aligned}$$

$$
\begin{aligned}
Q((2,0,0),(2,0,0)) &= D((2,0,0),(2,0,0)) + Q((2,0,0),(2,0,0))_D \\
&\quad + \sum_{j=1}^{3} Q^{\{j,j\}}((1,1,0),(1,1,0)) \\
&= D((2,0,0),(2,0,0)) + I_{c_1^{\{1,1\}}} \otimes O^{(1)}(2,2) \otimes I_{r_1^{\{1,1\}}} \\
&\quad + I_{c_2^{\{2,2\}}} \otimes O^{(2)}(0,0) \otimes I_{r_2^{\{2,2\}}} + I_{c_3^{\{3,3\}}} \otimes O^{(3)}(0,0) \otimes I_{r_3^{\{3,3\}}} \\
&= (0) \otimes I_1 + I_1 \otimes (0) \otimes I_1 + I_1 \otimes (0) = 0.
\end{aligned}
$$

Thus, we have the generator matrix

$$
Q = \left(\begin{array}{cccc|cc|cc|cc|c}
* & \mu_1^{(3)} & \mu_1^{(2)} & & & & & & & & \\
 & * & & \mu_1^{(2)} & (1-a)\mu_2^{(3)} & & & & a\mu_2^{(3)} & & \\
 & & * & \mu_1^{(3)} & & & & & & & \\
 & & & * & & (1-a)\mu_2^{(3)} & & & & a\mu_2^{(3)} & \\
\hline
 & & & & * & \mu_1^{(2)} & & & & & \\
\mu_2^{(2)} & & & & & * & & & & & \\
\hline
\mu_1^{(1)} & & & & & & * & \mu_1^{(3)} & & & \\
 & \mu_1^{(1)} & & & & & & * & (1-a)\mu_2^{(3)} & & a\mu_2^{(3)} \\
\hline
 & & & & \mu_1^{(1)} & & & & * & \mu_1^{(2)} & \\
 & & & & & \mu_1^{(1)} & \mu_2^{(2)} & & & * & \\
\hline
 & & & & & & & & \mu_1^{(1)} & & *
\end{array}\right).
$$

Hierarchical representation of QNs consists of a two level structure and assumes information abstraction between levels [8]. The first level structure, called high level model (HLM), determines the routing among second level structures, called low level models (LLMs). LLMs, either consist of queues or are themselves structures. The HLM defines the routing between sublevel structures. In that sense, the hierarchy can be extended to an arbitrary number of levels. Any piece of information belonging to a particular level is hidden from subsequent levels. In our representation, we restrict closed QN models to include only two levels of hierarchy. Thus, we have queues as LLMs and define the interaction among LLMs by an HLM (see Figure 2.2 which is taken from [9]).

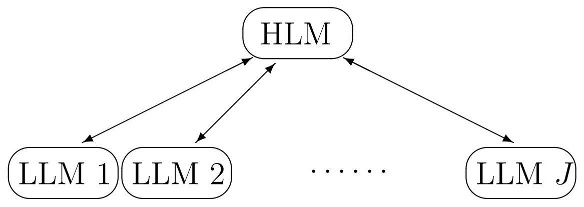

Figure 2.2: HLM–LLM relationship.

In order to define the HLM, we need to specify the transitions among LLMs. There are two ingredients that help us to reveal these transitions: one is the set $\mathcal{N}$ and the other is the routing probability matrix P. Since each component of $n \in \mathcal{N}$ corresponds to the number of customers in an LLM, possible transitions from n to m, where $n, m \in \mathcal{N}$, can be determined by considering P and $m = n - e_j^T + e_k^T$ as discussed after (2.2). These transitions are represented by an $(\mathcal{N} \times \mathcal{N})$ matrix called the HLM matrix. In that respect, elements of $\mathcal{N}$ are named as *states* of the HLM.

Example 1 consists of 3 LLMs and 5 HLM states, resulting with a generator matrix Q of order 11. There are 9 transitions among HLM states, corresponding to blocks ((0,1,1),(0,2,0)), ((0,1,1),(1,1,0)), ((0,2,0),(0,1,1)), ((1,0,1),(0,1,1)), ((1,0,1),(1,1,0)), ((1,0,1),(2,0,0)), ((1,1,0),(0,2,0)), ((1,1,0),(1,0,1)), and ((2,0,0), (1,1,0)) of Q obtained by using Kronecker products and 5 local transitions corresponding to diagonal blocks of Q obtained by using Kronecker sums. Thus, the HLM matrix of order 5 has 14 nonzeros with local transitions along the diagonal and transitions that result from movement of customers between queues in the off–diagonal. Each nonzero element in the HLM matrix corresponds to a Kronecker product of 3 LLM matrices, which are defined by the arrival and service processes of queues.

In practice, Q is never stored nor generated explicitly; instead an efficient vector–Kronecker product algorithm is used to carry out the steady–state analysis of Q underlying the closed QN. A state–of–the–art method that can be used toward this end is introduced in the next section.

2.2 ML Method

An ML method [11, 12] originating from aggregation–disaggregation and using multigrid iteration can be employed to obtain the steady–state vector of the generator matrix of a closed QN which is hierarchically modeled as described. The ML method, which is capable of aggregating according to a fixed or circular order using V, F, or W cycles, is given in Algorithms 1 and 2, where the vector ψ defines aggregation order and S refers to the smoother type. Let $l \in \{0, 1, \ldots, J\}$ define the current level in the hierarchy. Then one V cycle of the ML method proceeds as follows. At the finest level, $l = 0$, a number of iterations (using one of the iterative methods Power, Jacobi over–relaxation – JOR or SOR) are applied to the vector $x^{(0)}$ with uniformly distributed elements using a splitting [39] of the generator matrix $Q_0 = Q$ and a smoothed vector $\tilde{x}^{(0)}$ is obtained (line 8 in Algorithm 1). Then, for the next level, $l = 1$, Q_0 is aggregated with respect to an LLM by using the smoothed vector $\tilde{x}^{(0)}$ (line 10 in Algorithm 1). Thus the elements of $\tilde{x}^{(0)}$ and the blocks of Q_0, which correspond to elements of the HLM matrix, are aggregated to obtain $x^{(1)}$ and Q_1, respectively. In the $(l+1)$th aggregation step, $l < J$ the smoothed vector $\tilde{x}^{(l)}$ and the matrix Q_l are used in the aggregation procedure to obtain $x^{(l+1)}$ and Q_{l+1} (lines 8 and 9 in Algorithm 2). At the coarsest level, where all LLMs become aggregated, Q collapses to the aggregated matrix Q_J of order $|\mathcal{N}|$, and the linear system

$$x^{(J)} Q_J = 0, \quad \textstyle\sum_{i=1}^{|\mathcal{N}|} x_i^{(J)} = 1 \tag{2.4}$$

is solved exactly (line 2 in Algorithm 2). At this point, the ML method starts to move in the reverse direction, from the coarsest level to the finest performing a number of iterations at each level after disaggregation (lines 16 and 17 in Algorithm 2, and lines 17 and 18 in Algorithm 1). In this way, the solution vector $x^{(J)}$ at the coarsest level is mapped back to the finest level. At the finest level, when a cycle finishes, the method computes the residual vector and either stops if a predefined tolerance is met or proceeds for another cycle.

Algorithm 1 Driver for ML method where input parameters *pre* and *post* define number of pre and post smoothings, O and C define order of aggregation and cycle type, and MAX_IT and $STOP\_TOL$ define maximum number of iterations and stopping tolerance, respectively.

```
mlDriver($pre, post, O, C, MAX\_IT, STOP\_TOL$)
 1: $\psi \leftarrow (1, 2, \ldots, J)$; $l \leftarrow 0$; $Q_l \leftarrow Q$; $x^{(l)} \leftarrow$ initial approximation; $it \leftarrow 0$; $stop \leftarrow FALSE$;
 2: if $C = W$ or $C = F$ then
 3:     $\gamma \leftarrow 2$;
 4: else
 5:     $\gamma \leftarrow 1$;
 6: end if
 7: repeat
 8:     $\tilde{x}^{(l)} \leftarrow S(Q_l, x^{(l)}, w, pre)$;
 9:     obtain $x^{(l+1)}$ by aggregating $\tilde{x}^{(l)}$ with respect to LLM $\psi_{l+1}$;
10:     obtain $Q_{l+1}$ using $Q_l$ and $\tilde{x}^{(l)}$;
11:     if $\gamma = 1$ then
12:         $y^{(l+1)} \leftarrow \mathrm{ML}(Q_{l+1}, x^{(l+1)}, \psi, \gamma, l+1, pre, post)$;
13:     else
14:         $y^{(l+1)} \leftarrow \mathrm{ML}(Q_{l+1}, x^{(l+1)}, \psi, \gamma, l+1, pre, post)$;
15:         $y^{(l+1)} \leftarrow \mathrm{ML}(Q_{l+1}, y^{(l+1)}, \psi, \gamma, l+1, pre, post)$;
16:     end if
17:     obtain $y^{(l)}$ by disaggregating $y^{(l+1)}$ with respect to LLM $\psi_{l+1}$;
18:     $\tilde{y}^{(l)} \leftarrow S(Q_l, y^{(l)}, w, post)$;
19:     if $C = F$ then
20:         $\gamma \leftarrow 2$;
21:     end if
22:     $x^{(l)} \leftarrow \tilde{y}^{(l)}$; $it = it + 1$;
23:     $x^{(l)} \leftarrow x^{(l)}/(x^{(l)}e)$; $r \leftarrow -x^{(l)}Q_l$;
24:     if $it \geq MAX\_IT$ or $\|r\| \leq STOP\_TOL$ then
25:         $stop \leftarrow TRUE$;
26:     else if $O = CIRCULAR$ then
27:         $\psi_k \leftarrow \psi_{(k \bmod J)+1}$ for $k \in \{1, 2, \ldots, J\}$;
28:     end if
29: until ($stop$)
30: return $x^{(l)}$;
```

Algorithm 2 Recursive ML function.

```
ML($Q_l, x^{(l)}, \psi, \gamma, l, pre, post$)
1: if $l = \#$ of components in $\psi$ then
2:     $\tilde{y}^{(l)} \leftarrow$ solve($Q_l, x^{(l)}$) subject to $\tilde{y}^{(l)} e = 1$;
3:     if C = F then
4:         $\gamma \leftarrow 1$;
5:     end if
6: else
7:     $\tilde{x}^{(l)} \leftarrow S(Q_l, x^{(l)}, w, pre)$;
8:     obtain $x^{(l+1)}$ by aggregating $\tilde{x}^{(l)}$ with respect to LLM $\psi_{l+1}$;
9:     obtain $Q_{l+1}$ using $Q_l$ and $\tilde{x}^{(l)}$;
10:    if $\gamma = 1$ then
11:        $y^{(l+1)} \leftarrow$ ML($Q_{l+1}, x^{(l+1)}, \psi, \gamma, l+1, pre, post$);
12:    else
13:        $y^{(l+1)} \leftarrow$ ML($Q_{l+1}, x^{(l+1)}, \psi, \gamma, l+1, pre, post$);
14:        $y^{(l+1)} \leftarrow$ ML($Q_{l+1}, y^{(l+1)}, \psi, \gamma, l+1, pre, post$);
15:    end if
16:    obtain $y^{(l)}$ by disaggregating $y^{(l+1)}$;
17:    $\tilde{y}^{(l)} \leftarrow S(Q_l, y^{(l)}, w, post)$;
18:    return $\tilde{y}^{(l)}$;
19: end if
```

Now, we discuss aggregation and disaggregation operations in more detail for the class of closed QNs with phase–type service distributions and arbitrary buffer sizes. Let us represent by $\mathcal{S}^{(l)}$ the state space of Q_l. Furthermore, let $\mathcal{S}_n^{(l)}$ denote the states of $\mathcal{S}^{(l)}$ corresponding to $n \in \mathcal{N}$ and $s_n^{(l)}$ denote an element of $\mathcal{S}_n^{(l)} \subseteq \mathcal{S}^{(l)}$, where $\mathcal{S}^{(l)} = \cup_{n \in \mathcal{N}} \mathcal{S}_n^{(l)}$ and $\cap_{n \in \mathcal{N}} \mathcal{S}_n^{(l)} = \emptyset$. Then for $l < J$, states of the aggregated generator matrix Q_l are mapped to states of the coarser aggregated generator matrix Q_{l+1} by the next definition.

Definition 2.3. *Let us define* $s_n^{(l)} = (s_n^{(l)}(1), \ldots, s_n^{(l)}(\psi_j), \ldots, s_n^{(l)}(J)) \in \mathcal{S}_n^{(l)}$, *where*

$$s_n^{(l)}(\psi_j) = \begin{cases} n_{\psi_j} & if \;\; \psi_j \leq l \\ (n_{\psi_j}, \phi_{\psi_j}) & if \;\; \psi_j > l \end{cases}$$

where $\psi_j \leq l$ *means queue* ψ_j *is aggregated and* $\psi_j > l$ *means queue* ψ_j *is not aggregated for* $\psi_j \in \{1, 2, \ldots, J\}$, *and* $n = (n_1, n_2, \ldots, n_J) \in \mathcal{N}$. *Then the mapping* $g_{n,\psi_l}^{(l+1)} : \mathcal{S}_n^{(l)} \rightarrow \mathcal{S}_n^{(l+1)}$, *which aggregates the* ψ_{l+1}*th component of* $s_n^{(l)}$ *at level* l, *is* *defined by*

$$\begin{aligned} g^{(l+1)}_{n,\psi_{l+1}}(s^{(l)}_n) &= g^{(l+1)}_{n,\psi_{l+1}}((s^{(l)}_n(1),\ldots,s^{(l)}_n(\psi_{l+1}),\ldots,s^{(l)}_n(J))) \\ &= (s^{(l)}_n(1),\ldots,n_{\psi_{l+1}},\ldots,s^{(l)}_n(J)) = s^{(l+1)}_n. \end{aligned}$$

With this definition, we see that the blocks of the generator matrix Q are aggregated with respect to phase states of queues. In Figure 2.3, we show the aggregation and disaggregation of the states defined by $\mathcal{N}$ through levels of an ML cycle for Example 1 for fixed ordering of LLMs.

$$\begin{tikzcd}
l=0 & l=1 & l=2 & l=3 \\
((0,0),(1,1),(1,1)) \arrow[r, leftrightarrow, "g^{(1)}_{(0,1,1),1}"] & (0,(1,1),(1,1)) \arrow[ddr, leftrightarrow, "g^{(2)}_{(0,1,1),2}"] & & \\
((0,0),(1,1),(1,2)) \arrow[r, leftrightarrow] & (0,(1,1),(1,2)) \arrow[ddr, leftrightarrow] & & \\
((0,0),(1,2),(1,1)) \arrow[r, leftrightarrow] & (0,(1,2),(1,1)) \arrow[r, leftrightarrow] & (0,1,(1,1)) \arrow[dr, leftrightarrow, "g^{(3)}_{(0,1,1),3}"] & \\
((0,0),(1,2),(1,2)) \arrow[r, leftrightarrow] & (0,(1,2),(1,2)) \arrow[r, leftrightarrow] & (0,1,(1,2)) \arrow[r, leftrightarrow] & (0,1,1) \\
((0,0),(2,1),(0,0)) \arrow[r, leftrightarrow, "g^{(1)}_{(0,2,0),1}"] & (0,(2,1),(0,0)) \arrow[dr, leftrightarrow, "g^{(2)}_{(0,2,0),2}"] & & \\
((0,0),(2,2),(0,0)) \arrow[r, leftrightarrow] & (0,(2,2),(0,0)) \arrow[r, leftrightarrow] & (0,2,(0,0)) \arrow[r, leftrightarrow, "g^{(3)}_{(0,2,0),3}"] & (0,2,0) \\
((1,1),(0,0),(1,1)) \arrow[r, leftrightarrow, "g^{(1)}_{(1,0,1),1}"] & (1,(0,0),(1,1)) \arrow[r, leftrightarrow, "g^{(2)}_{(1,0,1),2}"] & (1,0,(1,1)) \arrow[dr, leftrightarrow, "g^{(3)}_{(1,0,1),3}"] & \\
((1,1),(0,0),(1,2)) \arrow[r, leftrightarrow] & (1,(0,0),(1,2)) \arrow[r, leftrightarrow] & (1,0,(1,2)) \arrow[r, leftrightarrow] & (1,0,1) \\
((1,1),(1,1),(0,0)) \arrow[r, leftrightarrow, "g^{(1)}_{(1,1,0),1}"] & (1,(1,1),(0,0)) \arrow[dr, leftrightarrow, "g^{(2)}_{(1,1,0),2}"] & & \\
((1,1),(1,2),(0,0)) \arrow[r, leftrightarrow] & (1,(1,2),(0,0)) \arrow[r, leftrightarrow] & (1,1,(0,0)) \arrow[r, leftrightarrow, "g^{(3)}_{(1,1,0),3}"] & (1,1,0) \\
((2,1),(0,0),(0,0)) \arrow[r, leftrightarrow, "g^{(1)}_{(2,0,0),1}"] & (2,(0,0),(0,0)) \arrow[r, leftrightarrow, "g^{(2)}_{(2,0,0),2}"] & (2,0,(0,0)) \arrow[r, leftrightarrow, "g^{(3)}_{(2,0,0),3}"] & (2,0,0)
\end{tikzcd}$$

Figure 2.3: Evolution of states through levels of an ML cycle for which the queues are aggregated in the order $\psi = (1,2,3)$ for Example 1.

Consequently, nonzero blocks of the aggregated generator matrix Q_l at level l of an ML cycle can be represented in terms of Kronecker products using the set of vectors with elements defined by

$$d^{(l)}_{(n,m),\psi}(s^{(l)}_n) = \frac{\sum_{s^{(l-1)}_n \in \mathcal{S}^{(l-1)}_n, g^{(l)}_{n,\psi_l}(s^{(l-1)}_n)=s^{(l)}_n} \tilde{x}^{(l-1)}(s^{(l-1)}_n)\ d^{(l-1)}_{(n,m)}(s^{(l-1)}_n)\ (e^T_{s^{(l-1)}_n} \tilde{G}^{(\psi_l)}_{(n,m)} e)}{x^{(l)}(s^{(l)}_n)}, \tag{2.5}$$

where $n, m \in \mathcal{N}$, ψ is the vector of indices of queues whose elements define the aggregation order, ψ_l is the aggregated component of $s^{(l-1)}_n$ at level l, $x^{(l-1)}$ is the smoothed vector at level $(l-1)$, $x^{(l)}(s^{(l)}_n) = \sum_{s^{(l-1)}_n \in \mathcal{S}^{(l-1)}_n, g^{(l)}_{n,\psi_l}(s^{(l-1)}_n)=s^{(l)}_n} \tilde{x}^{(l-1)}(s^{(l-1)}_n)$ is $s^{(l)}_n$th element of the new vector to be smoothed at level l, $e_{s^{(l-1)}_n}$ is the $s^{(l-1)}_n$th column of the identity matrix of order $\#\_of\_rows(\tilde{G}^{(\psi_l)}_{(n,n)})$, and

$$\tilde{G}^{(\psi_l)}_{(n,m)} = \begin{cases} G^{(\psi_l)}(n_{\psi_l}, m_{\psi_l}), & n_{\psi_l} \neq m_{\psi_l} \\ I_{t^{(\psi_l)}}, & n_{\psi_l} = m_{\psi_l} \text{ and } n_{\psi_l} \neq 0 \\ 1, & n_{\psi_l} = m_{\psi_l} \text{ and } n_{\psi_l} = 0 \end{cases}.$$

Hence, using (2.2), blocks of Q_l are defined by

$$Q_l(n,m) = \begin{cases} Q^{\{\psi_j,\psi_k\}}_l(n,m), & n \neq m \text{ and } m = n - e^T_{\psi_j} + e^T_{\psi_k} \\ D_l(n,n) + Q_l(n,n)_D + \sum_{l<\psi_j, \psi_j \in \{1,2,\ldots,J\}} Q^{\{\psi_j,\psi_j\}}_l(n,n), & n = m \\ 0 & \text{otherwise} \end{cases}, \tag{2.6}$$

where e_{ψ_j} is column ψ_j of the identity matrix, $m = n - e^T_{\psi_j} + e^T_{\psi_k}$ refers to service completion at queue ψ_j and arrival to queue ψ_k when in state n so as to make a transition to state m, $D_l(n,n)$ is the diagonal matrix summing block n of rows in Q_l to zero,

$$Q_l^{\{\psi_j,\psi_k\}}(n,m) = \begin{cases} p_{\psi_j,\psi_k}(I_{c_{\psi_j}^{\{\psi_j,\psi_k\}}} \otimes S^{(\psi_j)}(n_{\psi_j},m_{\psi_j}) \otimes I_{r_{\psi_j}^{\{\psi_j,\psi_k\}}}) \\ (I_{c_{\psi_k}^{\{\psi_j,\psi_k\}}} \otimes A^{(\psi_k)}(n_{\psi_k},m_{\psi_k}) \otimes I_{r_{\psi_k}^{\{\psi_j,\psi_k\}}}), & l < \psi_j < \psi_k \\ p_{\psi_j,\psi_k}\mathrm{diag}(d_{(n,m),\psi}^{(l)})(I_{c_{\psi_k}^{\{\psi_j,\psi_k\}}} \otimes A^{(\psi_k)}(n_{\psi_k},m_{\psi_k}) \otimes I_{r_{\psi_k}^{\{\psi_j,\psi_k\}}}), & \psi_j \le l < \psi_k \\ p_{\psi_j,\psi_k}(I_{c_{\psi_k}^{\{\psi_j,\psi_k\}}} \otimes A^{(\psi_k)}(n_{\psi_k},m_{\psi_k}) \otimes I_{r_{\psi_k}^{\{\psi_j,\psi_k\}}}) \\ (I_{c_{\psi_j}^{\{\psi_j,\psi_k\}}} \otimes S^{(\psi_j)}(n_{\psi_j},m_{\psi_j}) \otimes I_{r_{\psi_j}^{\{\psi_j,\psi_k\}}}), & l < \psi_k < \psi_j \\ p_{\psi_j,\psi_k}\mathrm{diag}(d_{(n,m),\psi}^{(l)})(I_{c_{\psi_j}^{\{\psi_j,\psi_k\}}} \otimes S^{(\psi_j)}(n_{\psi_j},m_{\psi_j}) \otimes I_{r_{\psi_j}^{\{\psi_j,\psi_k\}}}), & \psi_k \le l < \psi_j \\ p_{\psi_j,\psi_j}(I_{c_{\psi_j}^{\{\psi_j,\psi_j\}}} \otimes S^{(\psi_j)}(n_{\psi_j},n_{\psi_j}-1)) \\ (A^{(\psi_j)}(n_{\psi_j},n_{\psi_j}+1) \otimes I_{r_{\psi_j}^{\{\psi_j,\psi_j\}}}), & l < \psi_j = \psi_k \\ p_{\psi_j,\psi_k}\mathrm{diag}(d_{(n,m),\psi}^{(l)}), & \psi_j,\psi_k \le l \end{cases},$$

$$Q_l(n,n)_D = \bigoplus_{\substack{l<\psi_j \\ \psi_j\in\{1,2,\dots,J\}}} O^{(\psi_j)}(n_{\psi_j},n_{\psi_j}) = \sum_{\substack{l<\psi_j \\ \psi_j\in\{1,2,\dots,J\}}} I_{c_{\psi_j}^{\{\psi_j,\psi_j\}}} \otimes O^{(\psi_j)}(n_{\psi_j},n_{\psi_j}) \otimes I_{r_{\psi_j}^{\{\psi_j,\psi_j\}}},$$

where for $j \in \{1,2,\dots,J\}$, $\psi_j \le l$ means queue ψ_j is aggregated and $l < \psi_j$ means queue ψ_j is not aggregated at lth level when the letter l is used, $c_x^{\{\psi_j,\psi_k\}} = \prod_{z<x,\ z\in\{\psi_{l+1},\psi_{l+2},\dots,\psi_J\}} size^{\{\psi_j,\psi_k\}}(z)$, $r_x^{\{\psi_j,\psi_k\}} = \prod_{z>x,\ z\in\{\psi_{l+1},\psi_{l+2},\dots,\psi_J\}} size^{\{\psi_j,\psi_k\}}(z)$, $size$ is defined as in (2.3) and $\mathrm{diag}(d_{(n,m),\psi}^{(l)})$ represents a square matrix with diagonal elements from $d_{(n,m),\psi}^{(l)}$.

Example 1 (continued).

The aggregated generator matrix Q_1 is the same as Q (see (2.2) and Figure 2.3), and we have

$$\begin{aligned} d_{((1,0,1),(0,1,1)),\psi}^{(1)}(1,(0,0),(1,1)) &= d_{((1,0,1),(0,1,1)),\psi}^{(1)}(1,(0,0),(1,2)) \\ &= d_{((1,1,0),(0,2,0)),\psi}^{(1)}(1,(1,1),(0,0)) \\ &= d_{((1,1,0),(0,2,0)),\psi}^{(1)}(1,(1,2),(0,0)) \\ &= d_{((2,0,0),(1,1,0)),\psi}^{(1)}(2,(0,0),(0,0)) = \mu_1^{(1)} \end{aligned}$$

with other $d^{(1)}_{(n,m),\psi} = 1$, and $\tilde{x}^{(1)} \in I\!R^{1\times 11}$ as the smoothed vector for the first level. Observe that $p_{1,1} = p_{2,2} = p_{3,3} = 0$, implying the third (summation) term in (2.2) for $n = m$ evaluates to zero, since $Q^{\{j,j\}}(n,n) = 0$ for $j = 3$ and $n \in \mathcal{N}$. Having defined the aggregation operation, we now compute the blocks of Q_2 of order 7, which is obtained by aggregating LLM 2 in level 2 using (2.5) and (2.6) for Example 1, where $\psi = (1,2,3)$. Hence, we define the blocks of Q_2 as follows.

Block $((0,1,1),(0,1,1))$:

$$\begin{aligned}
Q_2((0,1,1),(0,1,1)) &= D_2((0,1,1),(0,1,1)) + Q_2((0,1,1),(0,1,1))_D \\
&\quad + \sum_{j=3}^{3} Q_2^{\{j,j\}}((0,1,1),(0,1,1)) \\
&= D_2((0,1,1),(0,1,1)) + I_{c_3^{\{3,3\}}} \otimes O^{(3)}(1,1) \otimes I_{r_3^{\{3,3\}}} \\
&= D_2((0,1,1),(0,1,1)) + \begin{pmatrix} -\mu_1^{(3)} & \mu_1^{(3)} \\ 0 & -\mu_2^{(3)} \end{pmatrix} \\
&= \begin{pmatrix} * & \mu_1^{(3)} \\ 0 & * \end{pmatrix}.
\end{aligned}$$

Block $((0,1,1),(0,2,0))$:

$$d^{(2)}_{((0,1,1),(0,2,0)),\psi}((0,1,(1,1)))$$

$$\begin{aligned}
&= \frac{\tilde{x}^{(1)}((0,(1,1),(1,1)))\ d^{(1)}_{((0,1,1),(0,2,0)),\psi}((0,(1,1),(1,1)))\ \left(\begin{pmatrix} 1 & 0 \end{pmatrix}\begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}\begin{pmatrix} 1 \\ 1 \end{pmatrix}\right)}{\tilde{x}^{(1)}((0,(1,1),(1,1))) + \tilde{x}^{(1)}((0,(1,2),(1,1)))} \\
&+ \frac{\tilde{x}^{(1)}((0,(1,2),(1,1)))\ d^{(1)}_{((0,1,1),(0,2,0)),\psi}((0,(1,2),(1,1)))\ \left(\begin{pmatrix} 0 & 1 \end{pmatrix}\begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}\begin{pmatrix} 1 \\ 1 \end{pmatrix}\right)}{\tilde{x}^{(1)}((0,(1,1),(1,1))) + \tilde{x}^{(1)}((0,(1,2),(1,1)))} \\
&= \frac{\tilde{x}^{(1)}((0,(1,1),(1,1))) + \tilde{x}^{(1)}((0,(1,2),(1,1)))}{\tilde{x}^{(1)}((0,(1,1),(1,1))) + \tilde{x}^{(1)}((0,(1,2),(1,1)))} = 1
\end{aligned}$$

$$d^{(2)}_{((0,1,1),(0,2,0)),\psi}((0,1,(1,2)))$$

$$
\begin{aligned}
&= \frac{\tilde{x}^{(1)}((0,(1,1),(1,2)))\; d^{(1)}_{((0,1,1),(0,2,0)),\psi}((0,(1,1),(1,2)))\; \left(\begin{pmatrix} 1 & 0 \end{pmatrix} \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \begin{pmatrix} 1 \\ 1 \end{pmatrix} \right)}{\tilde{x}^{(1)}((0,(1,1),(1,2))) + \tilde{x}^{(1)}((0,(1,2),(1,2)))} \\
&+ \frac{\tilde{x}^{(1)}((0,(1,2),(1,2)))\; d^{(1)}_{((0,1,1),(0,2,0)),\psi}((0,(1,2),(1,2)))\; \left(\begin{pmatrix} 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \begin{pmatrix} 1 \\ 1 \end{pmatrix} \right)}{\tilde{x}^{(1)}((0,(1,1),(1,2))) + \tilde{x}^{(1)}((0,(1,2),(1,2)))} \\
&= \frac{\tilde{x}^{(1)}((0,(1,1),(1,2))) + \tilde{x}^{(1)}((0,(1,2),(1,2)))}{\tilde{x}^{(1)}((0,(1,1),(1,2))) + \tilde{x}^{(1)}((0,(1,2),(1,2)))} = 1
\end{aligned}
$$

$$
\begin{aligned}
Q_2((0,1,1),(0,2,0)) &= Q_2^{\{3,2\}}((0,1,1),(0,2,0)) \\
&= p_{3,2}\ \mathrm{diag}(d^{(2)}_{((0,1,1),(0,2,0)),\psi})\ (I_{c_3^{\{3,2\}}} \otimes S^{(3)}(1,0) \otimes I_{r_3^{\{3,2\}}}) \\
&= (1-a) I_2 \begin{pmatrix} 0 \\ \mu_2^{(3)} \end{pmatrix} \\
&= \begin{pmatrix} 0 \\ (1-a)\mu_2^{(3)} \end{pmatrix}.
\end{aligned}
$$

Block $((0,1,1),(1,1,0))$:

$$d^{(2)}_{((0,1,1),(1,1,0)),\psi}((0,1,(1,1)))$$

$$
\begin{aligned}
&= \frac{\tilde{x}^{(1)}((0,(1,1),(1,1)))\; d^{(1)}_{((0,1,1),(1,1,0)),\psi}((0,(1,1),(1,1)))\; \left(\begin{pmatrix} 1 & 0 \end{pmatrix} \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \begin{pmatrix} 1 \\ 1 \end{pmatrix} \right)}{\tilde{x}^{(1)}((0,(1,1),(1,1))) + \tilde{x}^{(1)}((0,(1,2),(1,1)))} \\
&+ \frac{\tilde{x}^{(1)}((0,(1,2),(1,1)))\; d^{(1)}_{((0,1,1),(1,1,0)),\psi}((0,(1,2),(1,1)))\; \left(\begin{pmatrix} 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \begin{pmatrix} 1 \\ 1 \end{pmatrix} \right)}{\tilde{x}^{(1)}((0,(1,1),(1,1))) + \tilde{x}^{(1)}((0,(1,2),(1,1)))} \\
&= \frac{\tilde{x}^{(1)}((0,(1,1),(1,1))) + \tilde{x}^{(1)}((0,(1,2),(1,1)))}{\tilde{x}^{(1)}((0,(1,1),(1,1))) + \tilde{x}^{(1)}((0,(1,2),(1,1)))} = 1
\end{aligned}
$$

$$d^{(2)}_{((0,1,1),(1,1,0)),\psi}((0,1,(1,2)))$$

$$= \frac{\tilde{x}^{(1)}((0,(1,1),(1,2)))\ d^{(1)}_{((0,1,1),(1,1,0)),\psi}((0,(1,1),(1,2)))\ \left(\begin{pmatrix} 1 & 0 \end{pmatrix}\begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}\begin{pmatrix} 1 \\ 1 \end{pmatrix}\right)}{\tilde{x}^{(1)}((0,(1,1),(1,2))) + \tilde{x}^{(1)}((0,(1,2),(1,2)))}$$
$$+ \frac{\tilde{x}^{(1)}((0,(1,2),(1,2)))\ d^{(1)}_{((0,1,1),(1,1,0)),\psi}((0,(1,2),(1,2)))\ \left(\begin{pmatrix} 0 & 1 \end{pmatrix}\begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}\begin{pmatrix} 1 \\ 1 \end{pmatrix}\right)}{\tilde{x}^{(1)}((0,(1,1),(1,2))) + \tilde{x}^{(1)}((0,(1,2),(1,2)))}$$
$$= \frac{\tilde{x}^{(1)}((0,(1,1),(1,2))) + \tilde{x}^{(1)}((0,(1,2),(1,2)))}{\tilde{x}^{(1)}((0,(1,1),(1,2))) + \tilde{x}^{(1)}((0,(1,2),(1,2)))} = 1$$

$$\begin{aligned}
Q_2((0,1,1),(1,1,0)) &= Q_2^{\{3,1\}}((0,1,1),(1,1,0)) \\
&= p_{3,1}\ \mathrm{diag}(d^{(2)}_{((0,1,1),(1,1,0)),\psi})\ (I_{c_3^{\{3,1\}}} \otimes S^{(3)}(1,0) \otimes I_{r_3^{\{3,1\}}}) \\
&= aI_2 \begin{pmatrix} 0 \\ \mu_2^{(3)} \end{pmatrix} \\
&= \begin{pmatrix} 0 \\ a\mu_2^{(3)} \end{pmatrix}.
\end{aligned}$$

Block $((0,2,0),(0,1,1))$:

$$d^{(2)}_{((0,2,0),(0,1,1)),\psi}((0,2,(0,0)))$$

$$= \frac{\tilde{x}^{(1)}((0,(2,1),(0,0)))\ d^{(1)}_{((0,2,0),(0,1,1)),\psi}((0,(2,1),(0,0)))\ \left(\begin{pmatrix} 1 & 0 \end{pmatrix}\begin{pmatrix} 0 & 0 \\ \mu_2^{(2)} & 0 \end{pmatrix}\begin{pmatrix} 1 \\ 1 \end{pmatrix}\right)}{\tilde{x}^{(1)}((0,(2,1),(0,0))) + \tilde{x}^{(1)}((0,(2,2),(0,0)))}$$
$$+ \frac{\tilde{x}^{(1)}((0,(2,2),(0,0)))\ d^{(1)}_{((0,2,0),(0,1,1)),\psi}((0,(2,2),(0,0)))\ \left(\begin{pmatrix} 0 & 1 \end{pmatrix}\begin{pmatrix} 0 & 0 \\ \mu_2^{(2)} & 0 \end{pmatrix}\begin{pmatrix} 1 \\ 1 \end{pmatrix}\right)}{\tilde{x}^{(1)}((0,(2,1),(0,0))) + \tilde{x}^{(1)}((0,(2,2),(0,0)))}$$
$$= \frac{\tilde{x}^{(1)}((0,(2,2),(0,0)))}{\tilde{x}^{(1)}((0,(2,1),(0,0))) + \tilde{x}^{(1)}((0,(2,2),(0,0)))}\mu_2^{(2)}$$

$$\begin{aligned}
Q_2((0,2,0),(0,1,1)) &= Q_2^{\{2,3\}}((0,2,0),(0,1,1)) \\
&= p_{2,3}\ \mathrm{diag}(d^{(2)}_{((0,2,1),(0,1,1)),\psi})\ (I_{c_3^{\{2,3\}}} \otimes A^{(3)}(0,1) \otimes I_{r_3^{\{2,3\}}}) \\
&= \frac{\tilde{x}^{(1)}((0,(2,2),(0,0)))}{\tilde{x}^{(1)}((0,(2,1),(0,0))) + \tilde{x}^{(1)}((0,(2,2),(0,0)))} \mu_2^{(2)} \begin{pmatrix} 1 & 0 \end{pmatrix} \\
&= \begin{pmatrix} \frac{\tilde{x}^{(1)}((0,(2,2),(0,0)))}{\tilde{x}^{(1)}((0,(2,1),(0,0)))+\tilde{x}^{(1)}((0,(2,2),(0,0)))} \mu_2^{(2)} & 0 \end{pmatrix}.
\end{aligned}$$

Block $((0,2,0),(0,2,0))$:

$$\begin{aligned}
Q_2((0,2,0),(0,2,0)) &= D_2((0,2,0),(0,2,0)) + Q_2((0,2,0),(0,2,0))_D \\
&\quad + \sum_{j=3}^{3} Q_2^{\{j,j\}}((0,2,0),(0,2,0)) \\
&= D_2((0,2,0),(0,2,0)) + I_{c_3^{\{3,3\}}} \otimes O^{(3)}(0,0) \otimes I_{r_3^{\{3,3\}}} \\
&= *.
\end{aligned}$$

Block $((1,0,1),(0,1,1))$:

$$d^{(2)}_{((1,0,1),(0,1,1)),\psi}((1,0,(1,1)))$$

$$= \frac{\tilde{x}^{(1)}((1,(0,0),(1,1)))\ d^{(1)}_{((1,0,1),(0,1,1)),\psi}((1,(0,0),(1,1)))\ \left(\begin{pmatrix} 1 \end{pmatrix} \begin{pmatrix} 1 & 0 \end{pmatrix} \begin{pmatrix} 1 \\ 1 \end{pmatrix} \right)}{\tilde{x}^{(1)}((1,(0,0),(1,1)))}$$

$$= \frac{\tilde{x}^{(1)}((1,(0,0),(1,1)))\ \mu_1^{(1)}\ \left(\begin{pmatrix} 1 \end{pmatrix} \begin{pmatrix} 1 & 0 \end{pmatrix} \begin{pmatrix} 1 \\ 1 \end{pmatrix} \right)}{\tilde{x}^{(1)}((1,(0,0),(1,1)))} = \mu_1^{(1)}$$

$$d^{(2)}_{((1,0,1),(0,1,1)),\psi}((1,0,(1,2)))$$

$$= \frac{\tilde{x}^{(1)}((1,(0,0),(1,2)))\ d^{(1)}_{((1,0,1),(0,1,1)),\psi}((1,(0,0),(1,2)))\ \left(\begin{pmatrix} 1 \end{pmatrix} \begin{pmatrix} 1 & 0 \end{pmatrix} \begin{pmatrix} 1 \\ 1 \end{pmatrix} \right)}{\tilde{x}^{(1)}((1,(0,0),(1,2)))}$$

$$= \frac{\tilde{x}^{(1)}((1,(0,0),(1,2)))\ \mu_1^{(1)}\ \left(\begin{pmatrix} 1 \end{pmatrix} \begin{pmatrix} 1 & 0 \end{pmatrix} \begin{pmatrix} 1 \\ 1 \end{pmatrix} \right)}{\tilde{x}^{(1)}((1,(0,0),(1,2)))} = \mu_1^{(1)}$$

$$\begin{aligned}
Q_2((1,0,1),(0,1,1)) &= Q_2^{\{1,2\}}((1,0,1),(0,1,1)) \\
&= p_{1,2}\ \mathrm{diag}(d^{(2)}_{((1,0,1),(0,1,1)),\psi}) \\
&= \begin{pmatrix} \mu_1^{(1)} & 0 \\ 0 & \mu_1^{(1)} \end{pmatrix}.
\end{aligned}$$

Block $((1,0,1),(1,0,1))$:

$$\begin{aligned}
Q_2((1,0,1),(1,0,1)) &= D_2((1,0,1),(1,0,1)) + Q_2((1,0,1),(1,0,1))_D \\
&\quad + \sum_{j=3}^{3} Q_2^{\{j,j\}}((1,0,1),(1,0,1)) \\
&= D_2((1,0,1),(1,0,1)) + I_{c_3^{\{3,3\}}} \otimes O^{(3)}(1,1) \otimes I_{r_3^{\{3,3\}}} \\
&= D_2((1,0,1),(1,0,1)) + \begin{pmatrix} -\mu_1^{(3)} & \mu_1^{(3)} \\ 0 & -\mu_2^{(3)} \end{pmatrix} \\
&= \begin{pmatrix} * & \mu_1^{(3)} \\ 0 & * \end{pmatrix}.
\end{aligned}$$

Block $((1,0,1),(1,1,0))$:

$$d^{(2)}_{((1,0,1),(1,1,0)),\psi}((1,0,(1,1)))$$

$$\begin{aligned}
&= \frac{\tilde{x}^{(1)}((1,(0,0),(1,1)))\ d^{(1)}_{((1,0,1),(1,1,0)),\psi}((1,(0,0),(1,1)))\ \Big(\begin{pmatrix} 1 \end{pmatrix}\begin{pmatrix} 1 \end{pmatrix}\begin{pmatrix} 1 \end{pmatrix}\Big)}{\tilde{x}^{(1)}((1,(0,0),(1,1)))} \\
&= \frac{\tilde{x}^{(1)}((1,(0,0),(1,1)))}{\tilde{x}^{(1)}((1,(0,0),(1,1)))} = 1
\end{aligned}$$

$$d^{(2)}_{((1,0,1),(1,1,0)),\psi}((1,0,(1,2)))$$

$$\begin{aligned}
&= \frac{\tilde{x}^{(1)}((1,(0,0),(1,2)))\ d^{(1)}_{((1,0,1),(1,1,0)),\psi}((1,(0,0),(1,2)))\ \Big(\begin{pmatrix} 1 \end{pmatrix}\begin{pmatrix} 1 \end{pmatrix}\begin{pmatrix} 1 \end{pmatrix}\Big)}{\tilde{x}^{(1)}((1,(0,0),(1,2)))} \\
&= \frac{\tilde{x}^{(1)}((1,(0,0),(1,2)))}{\tilde{x}^{(1)}((1,(0,0),(1,2)))} = 1
\end{aligned}$$

$$\begin{aligned}
Q_2((1,0,1),(1,1,0)) &= Q_2^{\{3,2\}}((1,0,1),(1,1,0)) \\
&= p_{3,2}\ \mathrm{diag}(d^{(2)}_{((1,0,1),(1,1,0)),\psi})(I_{c_3^{\{3,2\}}} \otimes S^{(3)}(1,0) \otimes I_{r_3^{\{3,2\}}}) \\
&= (1-a)I_2 \begin{pmatrix} 0 \\ \mu_2^{(3)} \end{pmatrix} \\
&= \begin{pmatrix} 0 \\ (1-a)\mu_2^{(3)} \end{pmatrix}.
\end{aligned}$$

Block $((1,0,1),(2,0,0))$:

$$d^{(2)}_{((1,0,1),(2,0,0)),\psi}((1,0,(1,1)))$$

$$\begin{aligned}
&= \frac{\tilde{x}^{(1)}(1,(0,0),(1,1)))\ d^{(1)}_{((1,0,1),(2,0,0)),\psi}((1,(0,0),(1,1)))\ \left(\begin{pmatrix} 1 \end{pmatrix}\begin{pmatrix} 1 \end{pmatrix}\begin{pmatrix} 1 \end{pmatrix}\right)}{\tilde{x}^{(1)}((1,(0,0),(1,1)))} \\
&= \frac{\tilde{x}^{(1)}((1,(0,0),(1,1)))}{\tilde{x}^{(1)}((1,(0,0),(1,1)))} = 1
\end{aligned}$$

$$d^{(2)}_{((1,0,1),(2,0,0)),\psi}((1,0,(1,2)))$$

$$\begin{aligned}
&= \frac{\tilde{x}^{(1)}((1,(0,0),(1,2)))\ d^{(1)}_{((1,0,1),(2,0,0)),\psi}((1,(0,0),(1,2)))\ \left(\begin{pmatrix} 1 \end{pmatrix}\begin{pmatrix} 1 \end{pmatrix}\begin{pmatrix} 1 \end{pmatrix}\right)}{\tilde{x}^{(1)}((1,(0,0),(1,2)))} \\
&= \frac{\tilde{x}^{(1)}((1,(0,0),(1,2)))}{\tilde{x}^{(1)}((1,(0,0),(1,2)))} = 1
\end{aligned}$$

$$\begin{aligned}
Q_2((1,0,1),(2,0,0)) &= Q_2^{\{3,1\}}((1,0,1),(2,0,0)) \\
&= p_{3,1}\ \mathrm{diag}(d^{(2)}_{((1,0,1),(2,0,0)),\psi})(I_{c_3^{\{3,1\}}} \otimes S^{(3)}(1,0) \otimes I_{r_3^{\{3,1\}}}) \\
&= aI_2 \begin{pmatrix} 0 \\ \mu_2^{(3)} \end{pmatrix} \\
&= \begin{pmatrix} 0 \\ a\mu_2^{(3)} \end{pmatrix}.
\end{aligned}$$

Block $((1,1,0),(0,2,0))$:

$$d^{(2)}_{((1,1,0),(0,2,0)),\psi}((1,1,(0,0)))$$

$$\begin{aligned}
&= \frac{\tilde{x}^{(1)}((1,(1,1),(0,0)))\ d^{(1)}_{((1,1,0),(0,2,0)),\psi}((1,(1,1),(0,0)))\ \left(\begin{pmatrix} 1 & 0 \end{pmatrix}\begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}\begin{pmatrix} 1 \\ 1 \end{pmatrix}\right)}{\tilde{x}^{(1)}((1,(1,1),(0,0))) + \tilde{x}^{(1)}((1,(1,2),(0,0)))} \\
&+ \frac{\tilde{x}^{(1)}((1,(1,2),(0,0)))\ d^{(1)}_{((1,1,0),(0,2,0)),\psi}((1,(1,2),(0,0)))\ \left(\begin{pmatrix} 0 & 1 \end{pmatrix}\begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}\begin{pmatrix} 1 \\ 1 \end{pmatrix}\right)}{\tilde{x}^{(1)}((1,(1,1),(0,0))) + \tilde{x}^{(1)}((1,(1,2),(0,0)))} \\
&= \frac{\tilde{x}^{(1)}((1,(1,1),(0,0)))\ \mu_1^{(1)}\ \left(\begin{pmatrix} 1 & 0 \end{pmatrix}\begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}\begin{pmatrix} 1 \\ 1 \end{pmatrix}\right)}{\tilde{x}^{(1)}((1,(1,1),(0,0))) + \tilde{x}^{(1)}((1,(1,2),(0,0)))} \\
&+ \frac{\tilde{x}^{(1)}((1,(1,2),(0,0)))\ \mu_1^{(1)}\ \left(\begin{pmatrix} 0 & 1 \end{pmatrix}\begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}\begin{pmatrix} 1 \\ 1 \end{pmatrix}\right)}{\tilde{x}^{(1)}((1,(1,1),(0,0))) + \tilde{x}^{(1)}((1,(1,2),(0,0)))} \\
&= \frac{\tilde{x}^{(1)}((1,(1,1),(0,0))) + \tilde{x}^{(1)}((1,(1,2),(0,0)))}{\tilde{x}^{(1)}((1,(1,1),(0,0))) + \tilde{x}^{(1)}((1,(1,2),(0,0)))}\mu_1^{(1)} = \mu_1^{(1)}
\end{aligned}$$

$$Q_2((1,1,0),(0,2,0)) = Q_2^{\{1,2\}}((1,1,0),(0,2,0)) = p_{1,2}\ \mathrm{diag}(d^{(2)}_{((1,1,0),(0,2,0)),\psi}) = \mu_1^{(1)}.$$

Block $((1,1,0),(1,0,1))$:

$$d^{(2)}_{((1,1,0),(1,0,1)),\psi}((1,1,(0,0)))$$

$$\begin{aligned}
&= \frac{\tilde{x}^{(1)}((1,(1,1),(0,0)))\ d^{(1)}_{((1,1,0),(1,0,1)),\psi}((1,(1,1),(0,0)))\ \left(\begin{pmatrix} 1 & 0 \end{pmatrix}\begin{pmatrix} 0 \\ \mu_2^{(2)} \end{pmatrix}\begin{pmatrix} 1 \end{pmatrix}\right)}{\tilde{x}^{(1)}((1,(1,1),(0,0))) + \tilde{x}^{(1)}((1,(1,2),(0,0)))} \\
&+ \frac{\tilde{x}^{(1)}((1,(1,2),(0,0)))\ d^{(1)}_{((1,1,0),(1,0,1)),\psi}((1,(1,2),(0,0)))\ \left(\begin{pmatrix} 0 & 1 \end{pmatrix}\begin{pmatrix} 0 \\ \mu_2^{(2)} \end{pmatrix}\begin{pmatrix} 1 \\ 1 \end{pmatrix}\right)}{\tilde{x}^{(1)}((1,(1,1),(0,0))) + \tilde{x}^{(1)}((1,(1,2),(0,0)))} \\
&= \frac{\tilde{x}^{(1)}((1,(1,2),(0,0)))}{\tilde{x}^{(1)}((1,(1,1),(0,0))) + \tilde{x}^{(1)}((1,(1,2),(0,0)))}\mu_2^{(2)}
\end{aligned}$$

$$\begin{aligned}
Q_2((1,1,0),(1,0,1)) &= Q_2^{\{2,3\}}((1,1,0),(1,0,1)) \\
&= p_{2,3}\ \mathrm{diag}(d^{(2)}_{((1,1,0),(0,2,0)),\psi})(I_{c_3^{\{2,3\}}} \otimes A^{(3)}(0,1) \otimes I_{r_3^{\{2,3\}}}) \\
&= \frac{\tilde{x}^{(1)}((1,(1,2),(0,0)))}{\tilde{x}^{(1)}((1,(1,1),(0,0))) + \tilde{x}^{(1)}((1,(1,2),(0,0)))}\mu_2^{(2)} \begin{pmatrix} 1 & 0 \end{pmatrix} \\
&= \begin{pmatrix} \frac{\tilde{x}^{(1)}((1,(1,2),(0,0)))}{\tilde{x}^{(1)}((1,(1,1),(0,0)))+\tilde{x}^{(1)}((1,(1,2),(0,0)))}\mu_2^{(2)} & 0 \end{pmatrix}.
\end{aligned}$$

Block $((1,1,0),(1,1,0))$:

$$\begin{aligned}
Q_2((1,1,0),(1,1,0)) &= D_2((1,1,0),(1,1,0)) + Q_2((1,1,0),(1,1,0))_D \\
&\quad + \sum_{j=3}^{3} Q_2^{\{j,j\}}((1,1,0),(1,1,0)) \\
&= D_2((1,1,0),(1,1,0)) + I_{c\{3,3\}_3} \otimes O^{(3)}(0,0) \otimes I_{r\{3,3\}_3} = *.
\end{aligned}$$

Block $((2,0,0),(1,1,0))$:

$$d^{(2)}_{((2,0,0),(1,1,0)),\psi}((2,0,(0,0)))$$

$$= \frac{\tilde{x}^{(1)}((2,(0,0),(0,0)))\ d^{(1)}_{((2,0,0),(1,1,0)),\psi}((2,(0,0),(0,0)))\ \left(\begin{pmatrix} 1 \end{pmatrix}\begin{pmatrix} 1 \end{pmatrix}\begin{pmatrix} 1 \end{pmatrix}\right)}{\tilde{x}^{(1)}((2,(0,0),(0,0)))} = \mu_1^{(1)}$$

$$Q_2((2,0,0),(1,1,0)) = Q_2^{\{1,2\}}((2,0,0),(1,1,0)) = p_{1,2}\ \mathrm{diag}(d^{(2)}_{((2,0,0),(1,1,0)),\psi}) = \mu_1^{(1)}$$

Block $((2,0,0),(2,0,0))$:

$$\begin{aligned}
Q_2((2,0,0),(2,0,0)) &= D_2((2,0,0),(2,0,0)) + Q_2((2,0,0),(2,0,0))_D \\
&\quad + \sum_{j=3}^{3} Q_2^{\{j,j\}}((2,0,0),(2,0,0)) \\
&= D_2((2,0,0),(2,0,0)) + I_{c_3^{\{3,3\}}} \otimes O^{(3)}(0,0) \otimes I_{r_3^{\{3,3\}}} = *.
\end{aligned}$$

Thus, we have the generator matrix

$$Q_2 = \left(\begin{array}{cc|c|cc|c|c} * & \mu_1^{(3)} & & & & & \\ & * & (1-a)\mu_2^{(3)} & & & a\mu_2^{(3)} & \\ \hline \frac{\tilde{x}^{(1)}(6)}{\tilde{x}^{(1)}(5)+\tilde{x}^{(1)}(6)}\mu_2^{(2)} & & * & & & & \\ \hline \mu_1^{(1)} & & & * & \mu_1^{(3)} & & \\ & \mu_1^{(1)} & & & * & (1-a)\mu_2^{(3)} & a\mu_2^{(3)} \\ \hline & & \mu_1^{(1)} & \frac{\tilde{x}^{(1)}(10)}{\tilde{x}^{(1)}(9)+\tilde{x}^{(1)}(10)}\mu_2^{(2)} & & * & \\ \hline & & & & & \mu_1^{(1)} & * \end{array} \right),$$

where the elements of $\tilde{x}^{(1)}$ are enumerated in lexicographical order of the states of Q_1.

In the ML method, aggregated generator matrices are never generated explicitly. The smoothed vectors are obtained via the efficient vector–Kronecker product multiplication algorithm used by iterative methods based on splittings. In order to handle rectangular blocks of matrices, the vector–Kronecker product multiplication algorithm is slightly modified and given by Algorithm 3. The diagonal elements of a generator matrix are computed by using a slightly modified version of Algorithm 3 which can perform Kronecker product–vector multiplication.

In the next chapter, we extend two approximative iterative methods based on decomposition from the literature so that they can utilize the Kronecker representation and employ the ML method.

Algorithm 3 Modified vector–Kronecker product multiplication algorithm computing $y = x(I_{nLeft} \otimes A \otimes I_{nRight})$.

```
**leftMult**$(x, nLeft, nRight, A)$
1: $base1 \leftarrow 0;$
2: $base2 \leftarrow 0;$
3: $jump1 \leftarrow \#\_of\_rows(A) \times nRight;$
4: $jump2 \leftarrow \#\_of\_cols(A) \times nRight;$
5: for $i \leftarrow 1$ to $nLeft$ do
6:     for $j \leftarrow 1$ to $nRight$ do
7:         $index \leftarrow base1 + j;$
8:         for $k \leftarrow 1$ to $\#\_of\_rows(A)$ do
9:             $z(k) \leftarrow x(index);$
10:            $index \leftarrow index + nRight;$
11:        end for
12:        $z \leftarrow z \times A;$
13:        $index \leftarrow base2 + j;$
14:        for $k \leftarrow 1$ to $\#\_of\_cols(A)$ do
15:            $y(index) \leftarrow z(k);$
16:            $index \leftarrow index + nRight;$
17:        end for
18:    end for
19:    $base1 \leftarrow base1 + jump1;$
20:    $base2 \leftarrow base2 + jump2;$
21: end for
22: return $y;$
```

Chapter 3

Approximative Decompositional Methods

In this chapter, we describe four approximative methods based on decomposition for closed QNs. These are the convolution algorithm, Akyildiz's mean value analysis (MVABLO), Marie's method, and Yao and Buzacott's method. The convolution algorithm is used in the analysis of closed QNs with infinite buffer sizes and exponential service distributions. MVABLO is used in the analysis of closed QNs with exponential service distributions and arbitrary buffer sizes. Marie's and Yao and Buzacott's methods are used in the analysis of closed QNs with Coxian service distributions and arbitrary buffer sizes. Marie's and Yao and Buzacott's methods are extended to include phase–type service distributions.

3.1 Convolution Algorithm

In 1967, Gordon and Newell showed that closed QNs with queues having exponential service distributions and infinite buffer sizes have product form solution [22]. Such networks are named Gordon–Newell QNs (GNQNs). Steady–state probabilities of GNQNs are computed using the service demands of customers per passage from queues. In that sense, the analysis of Gordon and Newell involves queue–wise decomposition of the QN. For a GNQN with J queues, service demands per passage of queue i, S_i, in the GNQN is defined by the product $v_i\mu_i$ for $i \in \{1, 2, \ldots, J\}$. The value μ_i denotes the mean service rate of queue i and if we take a queue, say queue k, in the GNQN as the reference queue, then v_i denotes the visit ratio of queue i relative to queue k. In other words, v_i expresses the number of visits to queue i between two consecutive visits to queue j. Visit ratios of queues in the closed QN are computed by solving the linear system $vP = v$, where P is the routing probability matrix of the closed QN, under the condition that $v_k = 1$. Consequently, if the closed QN has K customers and state space $\mathcal{N}$, then the steady–state solution is given by the product form equation

$$\pi(n) = \frac{1}{NC^{(J)}(K)} \prod_{i=1}^{J} S_i^{n_i}, \tag{3.1}$$

where $S_i = v_i\mu_i$, $n \in \mathcal{N}$, and the normalization constant $NC^{(J)}(K)$ is defined as

$$NC^{(J)}(K) = \sum_{n\in\mathcal{N}} \prod_{i=1}^{J} S_i^{n_i}.$$

Computation of the normalization constant may require large number of operations if $|\mathcal{N}|$ is large. Therefore, a recursive scheme for the computation of normalization constants of GNQNs was first introduced in [13] and called convolution algorithm. The algorithm is defined by the equation

$$NC^{(j)}(k+1) = NC^{(j-1)}(k+1) + S_j NC^{(j)}(k), \tag{3.2}$$

where $j \in \{2, 3, \ldots, J\}$, $k \in \{0, 1, \ldots, K-1\}$, and boundary values are defined by $NC^{(1)}(k) = S_1^k$. Computation of the normalization constants are handled by Algorithm 4 using (3.2), where the vectors $NC^{(j)}$ are replaced with the vector NC of length $(K+1)$. After the computation of the normalization constants, the steady–state distribution vector can be calculated using (3.1).

Algorithm 4 Convolution algorithm.

```
**CA**($\mu, P, J, K$)
1: $NC(0) \leftarrow 1$;
2: solve $vP = v$ subject to $v_1 = 1$;
3: **for** $i \leftarrow 1$ to $J$ **do**
4:     $S_i \leftarrow v_i \mu_i$;
5: **end for**
6: **for** $k \leftarrow 1$ to $K$ **do**
7:     $NC(k) \leftarrow S_1^k$;
8: **end for**
9: **for** $j \leftarrow 2$ to $J$ **do**
10:     **for** $k \leftarrow 1$ to $K$ **do**
11:         $NC(k) \leftarrow NC(k) + S_j NC(k-1)$;
12:     **end for**
13: **end for**
14: **return** $NC$;
```

3.2 Akyildiz's Mean Value Analysis

Akyildiz's mean value analysis (MVABLO) [2] is based on the classical mean value analysis (MVA) [33], but it handles blocking QNs. MVA works on GNQNs and average performance measures are obtained by a scheme which does not require the computation of the underlying CTMC of the closed QN or the normalization constants. MVABLO also uses such a scheme to derive average performance measures in closed QNs with exponential service rates, but in order to introduce blocking it assumes that two properties hold: a queue whose successor queue has a full buffer becomes blocked after service completion and a queue whose buffer is full cannot accept any customer.

The first property implies that in order to compute the mean residence time of a blocked queue, the mean remaining service time of a customer in the downstream queue which causes blocking should be added to the mean residence time of a customer in the blocked queue. Therefore, for a closed QN having exponential service distributions with K customers and J queues, if the total capacity of queues $\sum_{j=1}^{J} c_j > K$, then the mean residence time of the customers in the blocked queue when there are $k \in \{1, 2, \ldots, K\}$ customers is defined as

$$E[R_j(k)] = \mu_j(1 + E[N_j(k-1)]) + BT_i\Big(\frac{v_j p_{j,i}}{v_i}\Big), \tag{3.3}$$

where $E[N_j(k-1)]$ denotes the average number of customers in queue j when there are k customers in the closed QN, BT_i denotes the mean service time of the successor queue i which causes blocking, v_i denotes the visit ratio for queue i, and $p_{j,i}$ is the routing probability from queue j to queue i. The first term of the sum in (3.3) is used in MVA when computing the mean residence time of a customer in queue j. The second term of the sum in (3.3) adds the mean remaining service time of a customer when there are multiple downstream queues.

The second property tells that a full station cannot accept a new customer. Hence, the mean residence time of a customer is computed by

$$E[R_j(k)] = \mu_j E[N_j(k-1)]. \tag{3.4}$$

Thus, using (3.3) and (3.4), MVABLO is given in Algorithm 5. Algorithm 5 computes average residence times of customers and average numbers of customers in queues, as well as the throughput of the network, ν, when there are $k \in \{1, 2, \ldots, K\}$ customers.

Algorithm 5 Mean value analysis for blocking queueing networks (MVABLO).

```
MVABLO($\mu, c, P, J, K$)
 1: solve $vP = v$ subject to $v_1 = 1$;
 2: for $i \leftarrow 1$ to $J$ do
 3:    $E[N_i(0)] \leftarrow 0$;
 4:    $BT_i(0) \leftarrow 0$;
 5:    $z_i(0) \leftarrow 1$;
 6: end for
 7: for $k \leftarrow 1$ to $K$ do
 8:    repeat
 9:       for $i \leftarrow 1$ to $J$ do
10:          $E[R_i(k)] \leftarrow \mu_i(z_i(k-1) + E[N_i(k-1)]) + BT_i(k-1)$;
11:       end for
12:       $\nu(k) \leftarrow k/(\sum_{i \leftarrow 1}^{J} v_i E[R_i(k)])$;
13:       for $i \leftarrow 1$ to $J$ do
14:          $E[N_i(k)] \leftarrow \nu(k) v_i E[R_i(k)]$;
15:          if $E[N_i(k)] > c_i$ then
16:             $z_i(k-1) \leftarrow 0$;
17:             for $j \leftarrow 1$ to $J$ do
18:                $BT_j(k-1) \leftarrow BT_j(k-1) + \mu_i(v_j p_{j,i}/v_i)$;
19:             end for
20:          else
21:             $z_i(k) \leftarrow z_i(k-1)$;
22:             $BT_i(k) \leftarrow BT_i(k-1)$;
23:          end if
24:       end for
25:    until $E[N_i(k)] < c_i$
26: end for
27: return $\nu, E[N_i(k)], E[R_i(k)]$ for $i \in \{1, 2, \ldots, J\}$ and $k \in \{0, 1, \ldots, K\}$;
```

3.3 Marie's Method

Marie's method consists of two stages. In the first stage, it decomposes the closed QN into subnetworks, and in the second stage, it analyzes the throughputs of subnetworks using a fixed–point iteration scheme. The decomposition satisfies some specific conditions, which imply that the steady–state probabilities of a subnetwork are independent of the states of other subnetworks. Thus, in the fixed–point iteration scheme, these independent subnetworks can be analyzed in isolation as open QNs assuming they have state dependent Poisson arrival rates.

In the first stage, the decomposition of queues into subnetworks depend on buffer sizes of queues and number of customers in the closed QN, and is described as follows. Given a closed QN, customers arriving to finite buffer queues of a subnetwork must come from queues that belong to the same subnetwork. In other words, any upstream queue, whose leaving customers are directed to a finite buffer queue, must be in the same subnetwork with the finite buffer queue. We accomplish the decomposition by using the recursive algorithm given in Algorithms 6 and 7. Algorithm 6 is the driver for the decomposition. It takes the set of queue indices, the number of queues, the number of customers, the buffer sizes of queues, and the routing probability matrix of a closed QN as parameters. Algorithm 6, together with Algorithm 7, proceeds as follows. If the set of queue indices is not empty, then an empty partition set and the minimum element from the set of queue indices is passed to Algorithm 7 (line 4 in Algorithm 6). Algorithm 7 checks whether this index belongs to the partition set or not. If the index belongs to the partition set, then Algorithm 7 returns the partition set. If the index does not belong to the partition set, then it is added to the partition set and Algorithm 7 checks if the added index represents a queue with infinite or finite buffer (line 6 in Algorithm 7). If the index represents an infinite buffer queue, then Algorithm 7 searches for queues, which have finite buffer sizes and customer arrivals from the infinite buffer queue, and adds these queues to the partition set if possible (lines 7 to 11 in Algorithm 7). On the other hand, if the index corresponds to a finite buffer queue, then Algorithm 7 have two cases to consider sequentially. First, it searches for indices of queues which have arrivals to the finite buffer queue and adds these queues to the partition set if possible (lines 14 to 16 in Algorithm 7). Second, it searches for indices of queues which have arrivals from the finite buffer queue and have finite buffers, and adds these queues to the partition set if possible (lines 17 to 19 in Algorithm 7). In this way, Algorithm 7 finds all the queue indices that belong to the same partition and returns the partition set to Algorithm 6. Indices belonging to the returned partition are removed from the set of indices of queues (line 5 in Algorithm 6), and if the set of indices is not empty, then Algorithm 6 proceeds with the minimum element from the remaining set of indices to construct another partition set. Using this algorithm, the set of queue indices $I = \{1, 2, \ldots, J\}$ of the closed QN is partitioned into subsets $\mathcal{J}^{(k)}$ of I, where $k \in \{1, 2, \ldots, S\}$ and S is the number of subnetworks. The partition of the closed QN in Example 1 introduced in Chapter 2 can be seen in Figure 3.1. The closed QN is partitioned into two subnetworks, where $\mathcal{J}^{(1)} = \{1\}$ and $\mathcal{J}^{(2)} = \{2, 3\}$.

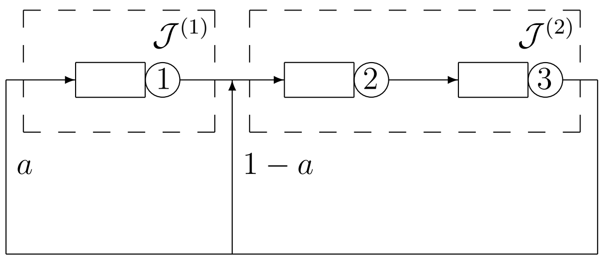

Figure 3.1: Decomposition of Example 1.

Algorithm 6 Driver for decomposition algorithm.

```
netDecomposeDriver($I, J, K, c, P$)
 1: $i \leftarrow 1$;
 2: while $I \neq \emptyset$ do
 3:     $\mathcal{J}^{(i)} \leftarrow \emptyset$;
 4:     $I_1 \leftarrow \min(I)$;
 5:     $\mathcal{J}^{(i)} \leftarrow$ netDecompose($I_1, J, K, P, c, \mathcal{J}^{(i)}$);
 6:     $I \leftarrow I - \mathcal{J}^{(i)}$;
 7:     $i \leftarrow i + 1$;
 8: end while
 9: $S \leftarrow i - 1$;
10: return $\{\mathcal{J}^{(1)}, \mathcal{J}^{(2)}, \ldots, \mathcal{J}^{(i-1)}\}$ and $S$;
```

Algorithm 7 Decomposition algorithm.

```
netDecompose($I_1, J, K, P, c, \mathcal{J}$)
 1: if $I_1 \notin \mathcal{J}$ then
 2:    $\mathcal{J} \leftarrow \mathcal{J} \cup \{I_1\}$;
 3: else
 4:    return $\mathcal{J}$;
 5: end if
 6: if $c_{I_1} \geq K$ then
 7:    for $j \leftarrow 1$ to $J$ do
 8:       if $P(I_1, j) > 0$ and $c_j < K$ then
 9:          $\mathcal{J} \leftarrow$ netDecompose($j, J, K, P, c, \mathcal{J}$);
10:       end if
11:    end for
12: else
13:    for $j \leftarrow 1$ to $J$ do
14:       if $P(j, I_1) > 0$ then
15:          $\mathcal{J} \leftarrow$ netDecompose($j, J, K, P, c, \mathcal{J}$);
16:       end if
17:       if $P(I_1, j) > 0$ and $c_j < K$ then
18:          $\mathcal{J} \leftarrow$ netDecompose($j, J, K, P, c, \mathcal{J}$);
19:       end if
20:    end for
21: end if
```

In the second stage, a fixed–point iteration scheme is proposed to obtain approximations to throughputs of subnetworks. The fixed–point iteration scheme is defined in [26] by the following system of equations:

$$\nu_j^{(r)}(i) = \frac{\alpha_j NC_j^{(r-1)}(K-i)}{NC_j^{(r-1)}(K-i+1)} \frac{\pi_j^{(r-1)}(i-1)}{\pi_j^{(r-1)}(i)}, \quad i \in \{1,2,\ldots,K\}, j \in \{1,2,\ldots,S\}. \tag{3.5}$$

$$A_m^{(r-1)}(K_m) = \begin{cases} 1 & \text{if} \quad K_m = 0 \\ \prod_{i=1}^{K_m} \nu_j^{(r-1)}(i) & \text{if} \quad K_m > 0 \end{cases},$$

$$\alpha = (\alpha_1, \alpha_2, \ldots, \alpha_S),$$

$$NC_v^{(r-1)}(u) = \sum_{K_1,K_2,\ldots,K_S} \left(\prod_{m=1}^{S} \frac{\alpha_m^{K_m}}{A_m^{(r-1)}(K_m)} \right) \text{ and } \sum_{m=1}^{S} K_m = u, K_v = 0,$$

$$\lambda_j^{(r-1)}(i) = \frac{\alpha_j NC_j^{(r-1)}(K-i-1)}{NC_j^{(r-1)}(K-i)},$$

where $\nu_j^{(r)}(i)$ and $\pi_j^{(r)}(i)$ denote approximated throughput and steady–state solution of the open QN defined by $\mathcal{J}^{(j)}$ when there are i customers in step r of the fixed–point iteration, respectively. The throughput and steady–state distribution vectors for subnetwork j are then given by $\nu_j^{(r)} = (\nu_j^{(r)}(1), \nu_j^{(r)}(2), \ldots, \nu_j^{(r)}(K))$ and $\pi_j^{(r)} = (\pi_j^{(r)}(1), \pi_j^{(r)}(2), \ldots, \pi_j^{(r)}(K))$, respectively. K_j denotes the number of customers in subnetwork j defined by $\mathcal{J}^{(j)}$ and

$$\alpha_k = \sum_{i \in \mathcal{J}^{(k)}} x_i \left(\sum_{j \in I/\mathcal{J}^{(k)}} p_{i,j} \right) \quad \text{for} \quad k \in \{1,2,\ldots,S\}$$

defines visit ratio of subnetwork for which x is a solution of $xP = x$ subject to $x_1 = 1$.

To execute step r of the fixed–point iteration, we compute the steady–state probability distribution $\pi_j^{(r-1)}$ of subnetwork j defined by $\mathcal{J}^{(j)}$ in step $(r-1)$. In step $(r-1)$, subnetwork $\mathcal{J}^{(j)}$ is perceived as an open QN with state dependent Poisson arrival rates $\lambda_j^{(r-1)}(i)$ for $i \in \{1, 2, \ldots, K\}$ and analyzed for its steady–state solution. In order to carry out the steady–state analysis of the open QN, we model the open QN as a closed QN, which consists of the subnetwork's queues and a slack queue. The slack queue is an infinite buffer queue, simulates the state dependent Poisson arrivals of customers to the subnetwork, and has exponentially distributed service times with load dependent service rates (see Figure 3.2). In this manner, subnetworks of the example, defined by partitions $\mathcal{J}^{(1)}$ and $\mathcal{J}^{(2)}$, are modeled as in Figure 3.3.

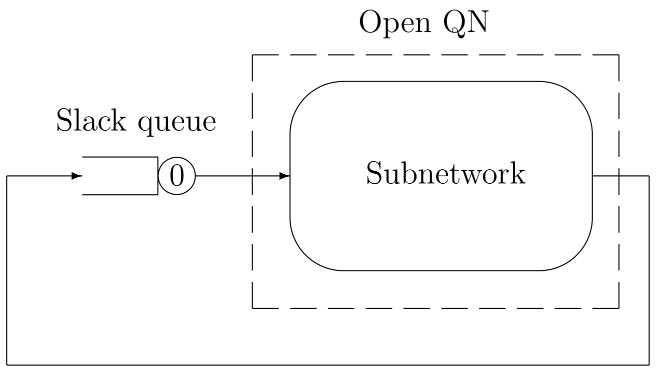

Figure 3.2: Open QN modeled as closed QN with a slack queue.

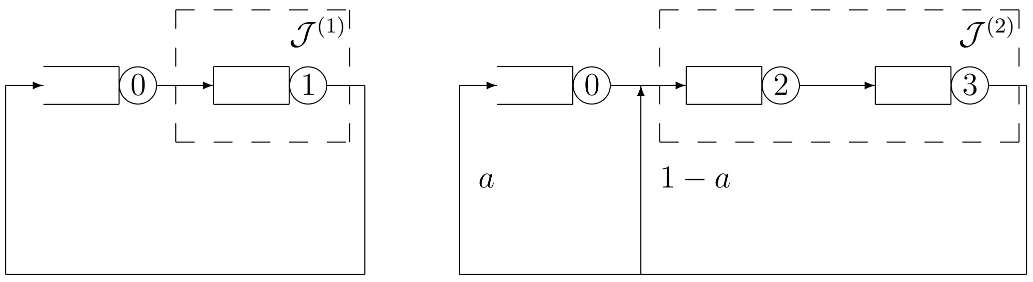

Figure 3.3: Decomposition of Example 1 for Marie's method.

Hence, the closed QN is modeled hierarchically by defining subnetworks of queues and the slack queue as LLMs, and constructing the HLM matrices, which exploit the interactions between queues in subnetworks. Consequently, in step r of Marie's method, the ML method is utilized to find the steady–state probability distribution $\pi_j^{(r)}$ of the closed subnetwork corresponding to $\mathcal{J}^{(j)}$ (see Step 4 of Marie's method in Table 3.1).

Table 3.1: Marie's method vs. Yao and Buzacott's method

Marie's method	**Yao & Buzacott's method**
Step 1. Decompose the closed QN into subnetworks.	**Step 1.** Set state dependent exponential service rates of queues μ to some initial value.
Step 2. Set throughput values of subnetworks ν to some initial value.	**Step 2.** Analyze the exponential network, which consists of queues with state dependent exponential servers and obtain steady state probabilities π^{exp}.
Step 3. Compute state dependent arrival rates λ for each subnetwork.	**Step 3.** Compute state dependent arrival rates λ for each queue using π^{exp}.
Step 4. Analyze subnetworks as open QNs under state dependent Poisson arrivals λ to derive steady–state probabilities π.	**Step 4.** Analyze each queue in isolation with its original service distribution and state dependent Poisson arrivals and derive steady–state probabilities π.
Step 5. Compute new ν values using λ and π, and goto Step 2 until convergence.	**Step 5.** Compute new throughput values ν using λ and π, initialize μ with ν and goto Step 1 until convergence.

3.4 Yao and Buzacott's Method

The idea behind Yao and Buzacott's method is to transform the closed QN into an exponential network, where each queue has an exponentially distributed service time with state dependent rate. After this transformation, Yao and Buzacott's method uses a fixed–point iteration scheme on the decomposed network to compute throughput rates of queues. The decomposition in this approach is maximal in the sense that individual queues become subnetworks. Each subnetwork is represented by using two queues: a slack queue, which has state dependent exponential service rate to simulate state dependent Poisson arrivals to the individual queue, and the individual queue with Coxian service distribution of closed QN, which is extended by a $\lambda(i)/PH/1/c_j$ queue, for $i \in \{0, 1, \ldots, K-1\}$ and $j \in \{1, 2, \ldots, J\}$ (see Figure 3.4). Yao and Buzacott's method also uses an iterative scheme to find the solution of a fixed–point equation, which computes throughputs of queues in the closed QN. The iterative scheme of Yao and Buzacott's method, whose steps are defined in Table 3.1, is based on the following system of equations:

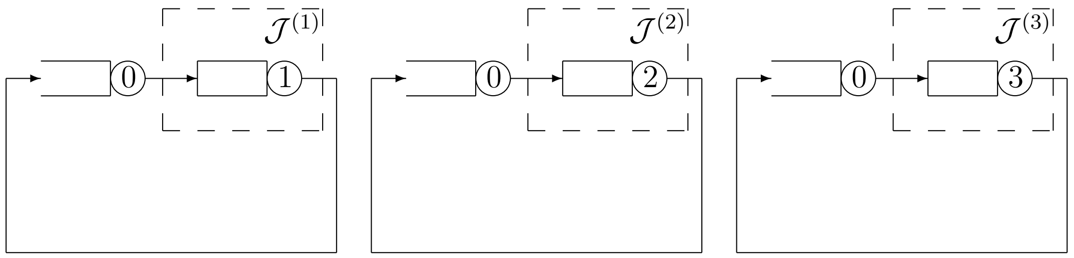

Figure 3.4: Decomposition of Example 1 for Yao and Buzacott's method.

$$\nu_j^{(r)}(i) = \lambda_j^{(r-1)}(i-1)\frac{\pi_j^{(r-1)}(i-1)}{\pi_j^{(r-1)}(i)}, \quad i \in \{1, 2, \ldots, c_j\}, \, j \in \{1, 2, \ldots, J\}, \quad (3.6)$$

$$\pi^{exp^{(r-1)}} Q^{exp^{(r-1)}} = 0; \quad \sum_{i=1}^{|\mathcal{N}|} \pi_i^{exp^{(r-1)}} = 1,$$

$$\lambda_j^{(r-1)}(i-1) = \nu_j^{(r-1)}(i)\frac{\pi_j^{exp^{(r-1)}}(i)}{\pi_j^{exp^{(r-1)}}(i-1)},$$

where $\pi^{exp^{(r)}}$ is the steady–state vector of the state dependent exponential closed QN's generator matrix, $Q^{exp^{(r)}}$, constructed by replacing the original service distributions of queues with state dependent exponential distributions that possess rates of $\mu_j^{(r)}(i) = \nu_j^{(r)}(i)$, $\pi_j^{exp^{(r-1)}}(i)$ is the marginal probability of having i customers in queue j in step r of the iteration, and $\pi_j^{(r)}$ is the steady–state probability vector of subnetwork j defined by $\mathcal{J}^{(j)}$, which includes queue j and a slack queue with state dependent exponential service rate $\mu_j^{(r)}(c_j - i) = \lambda_j^{(r)}(i)$.

Subnetworks that consist of individual queues of the closed QN are hierarchically modeled as closed QNs (Figure 3.4). As in Marie's case, the slack queue simulates state dependent Poisson arrivals from the outer environment to the subnetworks. In iteration step r of (3.6), the steady–state probability distribution $\pi_j^{(r)}$ of the subnetwork defined by $\mathcal{J}^{(j)}$ is computed by the ML method (see Step 4 of Yao and Buzacott's method in Table 3.1). The decomposition procedure of Yao and Buzacott's method represents subnetworks of the closed QN with one queue from the closed QN and a slack queue. Therefore each subnetwork consists of two LLMs and the ML method proceeds only for two levels. On the other hand, the generated exponential closed QN is solved for its steady–state distribution $\pi^{exp^{(r)}}$ by BiCGStab with ILU preconditioning [36].

Although, Marie's and Yao and Buzacott's methods are both approximation schemes based on decomposing the closed QN into subnetworks and analyzing these subnetworks as open QNs under state dependent Poisson arrivals, their methodologies differ in two ways: the decomposition procedure and the computation procedure of the state dependent Poisson arrival rates. In the former case, the decomposition satisfies predefined conditions which imply that the steady–state probabilities of a subnetwork is independent of the states of other subnetworks. In the latter case, a maximal decomposition of the closed QN is assumed, where each queue is treated as a subnetwork. Yet, in both methods, the sets $\mathcal{J}^{(k)}$ partition the closed QN into mutually exclusive sets of queues. Marie's method derives the state dependent arrival rates using normalizing constants. On the other hand, Yao and Buzacott's method computes the rates using marginal probabilities of subnetworks.

In the next chapter, we analyze Marie's and Yao and Buzacott's methods for their time and space complexities, and existence of a fixed–point.

Chapter 4

Analysis

This chapter is divided into two sections. In the first section, we provide analysis for the time and space complexity of the methods discussed in chapter 3. In the last section, the methods which use fixed–point iteration, namely Marie's method and Yao and Buzacott's method, are analyzed for the existence of a fixed–point.

4.1 Complexity Analysis

In this section, we start by giving upper bounds on the number of floating–point operations and space requirements in the convolution algorithm and Akyildiz's mean value analysis. Then we continue by giving an upper bound on the number of floating–point operations for one V, F, or W cycle of the ML method and proceed by giving upper bounds on the number of floating–point operations for one iteration of Marie's method and Yao and Buzacott's method together with upper bounds on their space requirements.

4.1.1 Convolution Algorithm

We give an upper bound on the number of floating–point operations performed by the convolution algorithm of Algorithm 4. Obtaining the solution of the linear system at line 2 of Algorithm 4 requires at most $O(J^3)$ floating–point operations. The for loop between lines 3 and 5 performs $O(J)$ floating–point operations. The for loop between lines 6 and 8 performs $O(J)$ floating–point operations. The part of the algorithm that computes the normalization constants between lines 9 and 13 performs $O(JK)$ floating–point operations. Altogether, an upper bound on the number of floating–point operations performed by the convolution algorithm is

$$O(J^3) + O(JK).$$

Again by considering Algorithm 4, we need one vector of length K to hold the normalization constants and need one vector of length J to hold the service demands of customers. Thus the space requirement of the convolution algorithm is

$$O(J + K).$$

4.1.2 Akyildiz's Mean Value Analysis

In order to give an upper bound on the number of floating–point operations performed in MVABLO, let us inspect Algorithm 5. The solution process at the first line requires $O(J^3)$ floating–point operations. The for loop between lines 17 and 19 performs $O(J)$ floating–point operations. The for loop between lines 13 and 24 turns J times and performs $O(J^2)$ floating–point operations. The for loop between lines 9 and 11 performs $O(J)$ floating point operations. By adding the number of floating–point operations performed by the for loop between lines 9 and 11, one step of the repeat loop between lines 8 and 25 performs at most $O(J^2)$ floating–point operations. The for loop between lines 7 and 26 turns K times, and if we define the number of times the repeat loop at step k is executed by η_k for $k \in \{1, 2, \ldots, K\}$, then an upper bound on the number of floating–point operations performed by MVABLO is

$$O(J^3) + \sum_{k=1}^{K} \eta_k O(J^2).$$

To find an upper bound on the space requirement of MVABLO, let us first look at line 10 in Algorithm 5. The variable $E[R_i(k)]$ is defined for $i \in \{1, 2, \ldots, J\}$ and $k \in \{1, 2, \ldots, K\}$, and needs $O(JK)$ storage. The variables $z_i(k)$, $E[N_i(k)]$, and $BT_i(k)$ is defined for $i \in \{1, 2, \ldots, J\}$ and $k \in \{0, 1, \ldots, K\}$, and they need $O(J(K+1))$ storage. Also, the variable $\nu(k)$ is defined for $k \in \{1, 2, \ldots, K\}$ and needs $O(K)$ storage. Therefore, the space requirement of MVABLO is given by

$$O(JK).$$

4.1.3 Iterative Methods Based on Splittings

In order to give an upper bound on the number of floating–point operations performed in one cycle of the ML method, we need to devise an upper bound on the number of floating–point operations performed in levels of an ML cycle when we use the Power, JOR, and SOR methods as smoothers. Detailed information on these iterative methods based on splittings for Kronecker representations can be found in [39].

To start with, the vector–Kronecker product multiplication algorithm performs at most $\prod_{i=1}^{J} n_i \sum_{i=1}^{J} n_i$ number of floating–point operations for a Kronecker product of J matrices of orders n_i for $i = \{1, 2, \ldots, J\}$ (see [18]). Let Q be the generator matrix of a closed QN with J queues and let us denote Q's HLM state space by $\mathcal{N}$. At any level l of an ML cycle, the iterative methods Power, JOR, and SOR perform the same number of vector Kronecker product multiplications for nondiagonal blocks of Q_l. Using this fact and (2.6) for $n, m \in \mathcal{N}$, we have the following equation which gives the number of floating–point operations performed for the (n, m)th nondiagonal block of Q_l when we use Power, JOR, or SOR methods as smoothers at level l of an ML cycle.

$$
{}_{ND}\mathcal{F}_l^{\{\psi_j,\psi_k\}}(n,m) = \begin{cases}
((c_{\psi_j}^{\{\psi_j,\psi_k\}} \times \#\_of\_rows(S^{(\psi_j)}(n_{\psi_j},m_{\psi_j})) \times r_{\psi_j}^{\{\psi_j,\psi_k\}}) & \\
\times(c_{\psi_j}^{\{\psi_j,\psi_k\}} + \#\_of\_rows(S^{(\psi_j)}(n_{\psi_j},m_{\psi_j})) + r_{\psi_j}^{\{\psi_j,\psi_k\}})) & \\
+((c_{\psi_k}^{\{\psi_j,\psi_k\}} \times \#\_of\_cols(A^{(\psi_k)}(n_{\psi_k},m_{\psi_k})) \times r_{\psi_k}^{\{\psi_j,\psi_k\}}) & \\
\times(c_{\psi_k}^{\{\psi_j,\psi_k\}} + \#\_of\_cols(A^{(\psi_k)}(n_{\psi_k},m_{\psi_k})) + r_{\psi_k}^{\{\psi_j,\psi_k\}})) & \\
+2length(x_l(n)), & l < \psi_j,\psi_k \\
\\
((c_{\psi_k}^{\{\psi_j,\psi_k\}} \times \#\_of\_cols(A^{(\psi_k)}(n_{\psi_k},m_{\psi_k})) \times r_{\psi_k}^{\{\psi_j,\psi_k\}}) & \\
\times(c_{\psi_k}^{\{\psi_j,\psi_k\}} + \#\_of\_cols(A^{(\psi_k)}(n_{\psi_k},m_{\psi_k})) + r_{\psi_k}^{\{\psi_j,\psi_k\}})) & \\
+3length(x_l(n)), & \psi_j \le l < \psi_k \\
\\
((c_{\psi_j}^{\{\psi_j,\psi_k\}} \times \#\_of\_rows(S^{(\psi_j)}(n_{\psi_j},m_{\psi_j})) \times r_{\psi_j}^{\{\psi_j,\psi_k\}}) & \\
\times(c_{\psi_j}^{\{\psi_j,\psi_k\}} + \#\_of\_rows(S^{(\psi_j)}(n_{\psi_j},m_{\psi_j})) + r_{\psi_j}^{\{\psi_j,\psi_k\}})) & \\
+3length(x_l(n)), & \psi_k \le l < \psi_j \\
\\
3length(x_l(n)), & \psi_j,\psi_k \le l
\end{cases}, \quad (4.1)
$$

where $l \in \{0,1,\ldots,J-1\}$, x_l is the iteration vector at level l of an ML cycle, $x_l(n)$ represents the multiplied part of the iteration vector corresponding to HLM state n and $j,k \in \{1,2,\ldots,J\}$.

Now let us give an upper bound on the number of floating–point operations performed by the vector–Kronecker product multiplication algorithm in Power, JOR, and SOR methods for diagonal blocks of Q_l. Again using (2.6), we obtain the following result which represents the number of floating–point operations performed by Power method for diagonal block (n,n) of Q_l.

$$
\begin{aligned}
{}_{POWER}\mathcal{F}_l(n,n)_D \quad = \sum_{\substack{l<\psi_j \\ \psi_j\in\{1,2,\ldots,J\}}} & \Big(((c_{\psi_j}^{\{\psi_j,\psi_j\}} \times \#\_of\_rows(O^{(\psi_j)}(n_{\psi_j},n_{\psi_j})) \times r_j^{\{\psi_j,\psi_j\}}) \\
& \times(c_{\psi_j}^{\{\psi_j,\psi_j\}} + \#\_of\_rows(O^{(\psi_j)}(n_{\psi_j},n_{\psi_j})) + r_{\psi_j}^{\{\psi_j,\psi_j\}})) \\
& +(c_{\psi_j}^{\{\psi_j,\psi_j\}} \times \#\_of\_rows(S^{(\psi_j)}(n_{\psi_j},n_{\psi_j})A^{(\psi_j)}(n_{\psi_j},n_{\psi_j})) \times r_{\psi_j}^{\{\psi_j,\psi_j\}}) \\
& +(c_{\psi_j}^{\{\psi_j,\psi_j\}} + \#\_of\_rows(S^{(\psi_j)}(n_{\psi_j},n_{\psi_j})A^{(\psi_j)}(n_{\psi_j},n_{\psi_j})) + r_{\psi_j}^{\{\psi_j,\psi_j\}}) \\
& +length(x_l(n))\Big) \qquad (4.2)
\end{aligned}
$$

Hence, using (4.1) and (4.2), we have at most the following number of floating–point operations for one iteration of Power method for all blocks of Q_l.

$$\mathcal{F}_{POWER}^{(l)} = 3length(x_l) + \Big(\sum_{\substack{\psi_j,\psi_k \in \{1,2,\ldots,J\} \\ n,m \in \mathcal{N}}} {}_{ND}\mathcal{F}_l^{\{\psi_j,\psi_k\}}(n,m) + {}_{POWER}\mathcal{F}_l(n,n)_D \Big). \quad (4.3)$$

When we consider the JOR method, we see that it also executes the same number of vector–Kronecker product multiplications performed by Power method. In JOR, dividing the iteration vector x_l's elements by diagonal entries of Q_l introduces $length(x_l)$ divisions instead of $length(x_l)$ multiplications. Also in JOR, we may need to do extra $2length(x_l)$ floating–point operations if the relaxation parameter is other than 1. Hence, we give an upper bound on the number of floating–point operations needed to perform one iteration of JOR at step l of an ML cycle also by (4.3).

Computation of new values of the iteration vector x_l in the SOR method becomes more complicated than in Power and JOR methods because of the block multiplication of the iteration vector x_l with Kronecker products. This complexity arises from the fact that SOR method uses old estimates of the iteration vector x_l to find new estimates as soon as they have been computed (see [23]). Therefore, in order to find the new estimates of the iteration vector x_l corresponding to n, we have to solve an upper–triangular system $x_l^{(new)}(n)(U_l(n,n) - D_l(n,n)) = x_l^{(old)}(n)L_l(n,n)$ for each diagonal block of Q_l, where $U_l(n,n)$ corresponds to strictly upper– and $L_l(n,n)$ corresponds to strictly lower–triangular parts of $Q_l(n,n)$ [39]. Thus,

$$Q_l(n,n) = U_l(n,n) + L_l(n,n) - D_l(n,n),$$

where $U_l(n,n)$ and $L_l(n,n)$ can be computed using Kronecker products by introducing strictly upper and lower triangular parts of $O^{(\psi_j)}(n_{\psi_j}, n_{\psi_j})$ and $S^{(\psi_j)}(n_{\psi_j}, n_{\psi_j} - 1)A^{(\psi_j)}(n_{\psi_j}, n_{\psi_j} + 1)$ by $O_U^{(\psi_j)}(n_{\psi_j}, n_{\psi_j})$, $O_L^{(\psi_j)}(n_{\psi_j}, n_{\psi_j})$ and $(S^{(\psi_j)}(n_{\psi_j}, n_{\psi_j} - 1)A^{(\psi_j)}(n_{\psi_j}, n_{\psi_j} + 1))_U$, $(S^{(\psi_j)}(n_{\psi_j}, n_{\psi_j} - 1)A^{(\psi_j)}(n_{\psi_j}, n_{\psi_j} + 1))_L$, respectively. Then from the definition of $Q^{\{\psi_j,\psi_k\}}(n,m)$ following (2.2), we have

$$\begin{aligned} U_l(n,n) &= \sum_{\substack{l<\psi_j \\ \psi_j\in\{1,2,\ldots,J\}}} I_{c_{\psi_j}^{\{\psi_j,\psi_j\}}} \otimes O_U^{(\psi_j)}(n_{\psi_j},n_{\psi_j}) \otimes I_{r_{\psi_j}^{\{\psi_j,\psi_j\}}} \\ &+ p_{\psi_j,\psi_j}(I_{c_{\psi_j}^{\{\psi_j,\psi_j\}}} \otimes (S^{(\psi_j)}(n_{\psi_j},n_{\psi_j}-1)A^{(\psi_j)}(n_{\psi_j},n_{\psi_j}+1))_U \otimes I_{r_{\psi_j}^{\{\psi_j,\psi_j\}}}) \end{aligned}$$

and

$$\begin{aligned} L_l(n,n) &= \sum_{\substack{l<\psi_j \\ \psi_j\in\{1,2,\ldots,J\}}} I_{c_{\psi_j}^{\{\psi_j,\psi_j\}}} \otimes O_L^{(\psi_j)}(n_{\psi_j},n_{\psi_j}) \otimes I_{r_{\psi_j}^{\{\psi_j,\psi_j\}}} \\ &+ p_{\psi_j,\psi_j}(I_{c_{\psi_j}^{\{\psi_j,\psi_j\}}} \otimes (S^{(\psi_j)}(n_{\psi_j},n_{\psi_j}-1)A^{(\psi_j)}(n_{\psi_j},n_{\psi_j}+1))_L \otimes I_{r_{\psi_j}^{\{\psi_j,\psi_j\}}}). \end{aligned}$$

Here, we see that the product $x_l^{(old)}(n)L_l(n,n)$ takes at most

$$\begin{aligned} &\sum_{\substack{l<\psi_j \\ \psi_j\in\{1,2,\ldots,J\}}} (((c_{\psi_j}^{\{\psi_j,\psi_j\}} \times \#\_of\_rows(O_L^{(\psi_j)}(n_{\psi_j},n_{\psi_j})) \times r_{\psi_j}^{\{\psi_j,\psi_j\}}) \\ &\times(c_{\psi_j}^{\{\psi_j,\psi_j\}} + \#\_of\_rows(O_L^{(\psi_j)}(n_{\psi_j},n_{\psi_j})) + r_{\psi_j}^{\{\psi_j,\psi_j\}})) \\ &+((c_{\psi_j}^{\{\psi_j,\psi_j\}} \times \#\_of\_rows((S^{(\psi_j)}(n_{\psi_j},n_{\psi_j}-1)A^{(\psi_j)}(n_{\psi_j},n_{\psi_j}+1))_L) \times r_{\psi_j}^{\{\psi_j,\psi_j\}}) \\ &\times(c_{\psi_j}^{\{\psi_j,\psi_j\}} + \#\_of\_rows((S^{(\psi_j)}(n_{\psi_j},n_{\psi_j}-1)A^{(\psi_j)}(n_{\psi_j},n_{\psi_j}+1))_L) + r_{\psi_j}^{\{\psi_j,\psi_j\}})) \\ &+length(x_l(n))) \end{aligned} \tag{4.4}$$

floating–point operations, where $length(x_l(n))$ is added to account for the addition of the resulting vector with part of the iteration vector corresponding to HLM state n. The upper–triangular system is solved by Algorithm 8 with the order of aggregation defined by ψ for which

$$\begin{aligned} U^{(\psi_j)}(n) &= O_U^{(\psi_j)}(n_{\psi_j},n_{\psi_j}) \otimes I_{r_{\psi_j}^{\{\psi_j,\psi_j\}}} \\ &+ p_{\psi_j,\psi_j}((S^{(\psi_j)}(n_{\psi_j},n_{\psi_j}-1)A^{(\psi_j)}(n_{\psi_j},n_{\psi_j}+1))_U \otimes I_{r_{\psi_j}^{\{\psi_j,\psi_j\}}}), \end{aligned}$$

where $j \in \{1,2,\ldots,J\}$, $\psi_j \in \{1,2,\ldots,J\}$ and $l < \psi_j$. Hence, using the information above, obtaining a solution through Algorithm 8 needs at most

Algorithm 8 Finds solution of $yU = x$ for y and aggregation order $\psi = (1, 2, \ldots, J)$, where U is an upper–triangular matrix described by (4.1).

newIterVec$(y, D(intrvl, intrvl), [U^{(k)}(n), \ldots, U^{(J)}(n)], k, n, t, oMatSz)$

```
if $n_k > 0$ and $t^{(k)} > 1$ then
    $nMatSz \leftarrow t^{(k)}$;
    $z \leftarrow$ zero row vector of length $oMatSz$;
else
    $nMatSz \leftarrow 1$;
end if
$yVecL \leftarrow oMatSz/nMatSz$;
for $i \leftarrow 1$ to $nMatSz$ do
    $intrvl \leftarrow ((i-1)(yVecL+1), \ldots, (i)(yVecL))$;
    if $k = J - 1$ then
        if $n_J = 0$ or $t^{(J)} = 1$ or $U^{(J)}(n) = 0$ then
            $y(intrvl) \leftarrow solve(y(intrvl), D(intrvl, intrvl))$;
        else
            $y(intrvl) \leftarrow solve(y(intrvl), (D(intrvl, intrvl) - U^{(J)}(n)))$;
        end if
    else if $k = J$ then
        if $n_J = 0$ or $t^{(J)} = 1$ or $U^{(J)}(n) = 0$ then
            $y \leftarrow solve(y, D(intrvl, intrvl))$;
        else
            $y \leftarrow solve(y, (D(intrvl, intrvl) - U^{(J)}(n)))$;
        end if
        return $y$
    else
        $y(intrvl) \leftarrow$ newIterVec$(y(intrvl), D(intrvl, intrvl),$
                        $[U^{(k+1)}(n), \ldots, U^{(J)}(n)], k+1, n, t, yVecL)$;
    end if
    if $n_k > 0$ and $i < nMatSz$ then
        $z(intrvl) \leftarrow y(intrvl)$;
        $y \leftarrow y - zU^{(k)}(n)$;
        $z(intrvl) \leftarrow 0$;
    end if
    return $y$;
end for
```

$$\sum_{\substack{l<\psi_k \\ \psi_k\neq\psi_J \\ \psi_k\in\{1,2,\ldots,J\}}} (t^{(\psi_k)}-1)\times(((\#\_of\_rows(O_U^{(\psi_k)}(n_{\psi_k},n_{\psi_k}))\times r_{\psi_k}^{\{\psi_k,\psi_k\}})$$
$$\times(\#\_of\_rows(O_U^{(\psi_k)}(n_{\psi_k},n_{\psi_k}))+r_{\psi_k}^{\{\psi_k,\psi_k\}}))$$
$$+((\#\_of\_rows((S^{(\psi_k)}(n_{\psi_k},n_{\psi_k}-1)A^{(k)}(n_{\psi_k},n_{\psi_k}+1))_U)\times r_{\psi_k}^{\{\psi_k,\psi_k\}})$$
$$\times(\#\_of\_rows((S^{(\psi_k)}(n_{\psi_k},n_{\psi_k}-1)A^{(\psi_k)}(n_{\psi_k},n_{\psi_k}+1))_U)+r_{\psi_k}^{\{\psi_k,\psi_k\}}))$$
$$+length(x_l(n))) \tag{4.5}$$

floating–point operations for line 29 of Algorithm 8 and at most

$$\Big(\sum_{\substack{l<\psi_k \\ \psi_k\neq\psi_J \\ \psi_k\in\{1,2,\ldots,J\}}} \#\_of\_rows(O_U^{(\psi_k)}(n_{\psi_k},n_{\psi_k}))\Big)\times\Big(\frac{\#\_of\_rows(U^{(\psi_J)}(n))^3}{3}+\frac{\#\_of\_rows(U^{(\psi_J)}(n))^2}{2}$$
$$+\frac{\#\_of\_rows(U^{(\psi_J)}(n))(\#\_of\_rows(U^{(\psi_J)}(n))+1)}{2}\Big) \tag{4.6}$$

floating–point operations for the solution procedures in lines 12, 14, 18, and 20 of Algorithm 8. Hence, using (4.4), (4.5), and (4.6), the number of floating–point operations performed for diagonal block (n,n) in SOR is at most

$$
\begin{aligned}
{}_{SOR}\mathcal{F}_l(n,n)_D \;=\; & \Big(\sum_{\substack{l<\psi_j \\ \psi_j\in\{1,2,\ldots,J\}}} (((c_{\psi_j}^{\{\psi_j,\psi_j\}} \times \#\_of\_rows(O_L^{(\psi_j)}(n_{\psi_j},n_{\psi_j})) \times r_{\psi_j}^{\{\psi_j,\psi_j\}}) \\
& \times(c_{\psi_j}^{\{\psi_j,\psi_j\}} + \#\_of\_rows(O_L^{(\psi_j)}(n_{\psi_j},n_{\psi_j})) + r_{\psi_j}^{\{\psi_j,\psi_j\}})) \\
& +((c_{\psi_j}^{\{\psi_j,\psi_j\}} \times \#\_of\_rows((S^{(\psi_j)}(n_{\psi_j},n_{\psi_j}-1)A^{(\psi_j)}(n_{\psi_j},n_{\psi_j}+1))_L) \times r_{\psi_j}^{\{\psi_j,\psi_j\}}) \\
& \times(c_{\psi_j}^{\{\psi_j,\psi_j\}} + \#\_of\_rows((S^{(\psi_j)}(n_{\psi_j},n_{\psi_j}-1)A^{(\psi_j)}(n_{\psi_j},n_{\psi_j}+1))_L) + r_{\psi_j}^{\{\psi_j,\psi_j\}})) \\
& +length(x_l(n)))\Big) \\
& +\Big(\sum_{\substack{l<\psi_k \\ \psi_k\neq\psi_J \\ \psi_k\in\{1,2,\ldots,J\}}} (t^{(\psi_k)}-1) \times (((\#\_of\_rows(O_U^{(\psi_k)}(n_{\psi_k},n_{\psi_k})) \times r_{\psi_k}^{\{\psi_k,\psi_k\}}) \\
& \times(\#\_of\_rows(O_U^{(\psi_k)}(n_{\psi_k},n_{\psi_k})) + r_{\psi_k}^{\{\psi_k,\psi_k\}})) \\
& +((\#\_of\_rows((S^{(\psi_k)}(n_{\psi_k},n_{\psi_k}-1)A^{(k)}(n_{\psi_k},n_{\psi_k}+1))_U) \times r_{\psi_k}^{\{\psi_k,\psi_k\}}) \\
& \times(\#\_of\_rows((S^{(\psi_k)}(n_{\psi_k},n_{\psi_k}-1)A^{(\psi_k)}(n_{\psi_k},n_{\psi_k}+1))_U) + r_{\psi_k}^{\{\psi_k,\psi_k\}})) \\
& +length(x_l(n)))\Big) \\
& +\Big(\Big(\sum_{\substack{l<\psi_k \\ \psi_k\neq\psi_J \\ \psi_k\in\{1,2,\ldots,J\}}} \#\_of\_rows(O_U^{(\psi_k)}(n_{\psi_k},n_{\psi_k}))\Big) \\
& \times\Big(\frac{\#\_of\_rows(U^{(\psi_J)}(n))^3}{3} + \frac{\#\_of\_rows(U^{(\psi_J)}(n))^2}{2} \qquad (4.7) \\
& +\frac{\#\_of\_rows(U^{(\psi_J)}(n))(\#\_of\_rows(U^{(\psi_J)}(n))+1)}{2}\Big)\Big). \qquad (4.8)
\end{aligned}
$$

Finally, using (4.1) and (4.7) we have at most

$$
\mathcal{F}_{SOR}^{(l)} = 2length(x_l) + \sum_{\substack{\psi_j,\psi_k\in\{1,2,\ldots,J\} \\ n,m\in\mathcal{N}}} {}_{ND}\mathcal{F}_l^{\{\psi_j,\psi_k\}}(n,m) + {}_{SOR}\mathcal{F}_l(n,n)_D \qquad (4.9)
$$

floating–point operations performed in one iteration of SOR method, where if the relaxation parameter is other than 1, SOR method may need to execute extra $2length(x_l)$ floating–point operations at level l of an ML cycle. Hence, we have given the total number of floating–point operations needed for one iteration of Power, JOR, and SOR methods at step l of an ML cycle. Clearly, Power, JOR, and SOR methods require $O(length(x_l))$ storage at step l of an ML cycle.

4.1.4 ML Method

To find an upper bound on the number of floating–point operations performed by one V cycle of the ML method, we consider the steps of the ML method. Algorithms 1 and 2 imply that at any level $l \in \{0, 1, \ldots, J-1\}$ of an ML cycle, the ML method performs $(pre+post)$ number of smoothing operations, one operation to compute diagonal elements of Q_l, one aggregation operation, one disaggregation operation, and one operation to find the vectors $d^{(l)}_{(n,m)}$, which are given in (2.5). Among these operations, computing the diagonal elements of Q_l costs at most $\mathcal{F}^{(l)}_{POWER}$ floating–point operations. The aggregation operation performs at most $length(x_l)$ floating–point operations in level l. The disaggregation operation works on the iteration vector x_l at level l of an ML cycle and performs at most $2length(x_l)$ floating–point operations. When we consider (2.5) that defines the vectors $d^{(l)}_{(n,m),\psi}$, we see that at most

$$\mathcal{F}^{(l)}_{(n,m)}(s^{(l)}_n) = \sum_{\substack{s^{(l-1)}_n \in \mathcal{S}^{(l-1)}_n, \\ g^{(l)}_{n,\psi_l}(s^{(l-1)}_n) = s^{(l)}_n}} \left(2 + \#\_of\_cols(\tilde{G}^{(\psi_l)}_{(n,m)})\right)$$

floating–point operations are performed for each $s^{(l)}_n \in \mathcal{S}^{(l)}_n$. Thus, we have a total of at most

$$\mathcal{F}^{(l)}_d = \sum_{\substack{n,m \in \mathcal{N} \\ n \neq m}} \mathcal{F}^{(l)}_{(n,m)}(s^{(l)}_n) \tag{4.10}$$

floating–point operations performed when computing the vectors $d^{(l)}_{(n,m)}$ at level l of a V cycle in the ML method. Consequently, for level l of a V cycle in ML for $l \in \{0, 1, \ldots, J-1\}$, the upper bound on the number floating–point operations is given by

$$_V\mathcal{F}^{(l)}_{ML}(S, pre, post, w) = \left\{ \begin{array}{ll} (pre+post)\mathcal{F}^{(l)}_{SOR} + \mathcal{F}^{(l)}_d + \mathcal{F}^{(l)}_{POWER}, & S = SOR \\ (pre+post+1)\mathcal{F}^{(l)}_{POWER} + \mathcal{F}^{(l)}_d, & S \in \{POWER, JOR\} \end{array} \right. . \tag{4.11}$$

Using (4.10), one V cycle of ML method performs at most

$$\begin{aligned} {}_V\mathcal{F}_{ML} &= \sum_{l=0}^{J-1} {}_V\mathcal{F}_{ML}^{(l)}(S, pre, post, w) + \mathcal{F}_{POWER}^{(0)} + length(x_0) \\ &\quad + \mathcal{F}_{POWER}^{(J)} + O(|\mathcal{N}|^3) \end{aligned} \tag{4.12}$$

floating–point operations, where $\mathcal{F}_{POWER}^{(0)}$ is the number of floating–point operations performed when computing the residual vector r, $length(x_0)$ is the number of floating–point operations needed to normalize the iteration vector, $O(|\mathcal{N}|^3)$ is the number of floating–point operations performed when solving the coarsest matrix Q_J directly, respectively, and $\mathcal{F}_{POWER}^{(J)}$ is the number of floating–point operations needed to compute the diagonal elements of the coarsest matrix Q_J.

Having given an upper bound on the number of floating–point operations for one V cycle of ML, using (4.10) we provide an upper bound on the number of floating–point operations that can be performed in level l of F and W cycles for $l \in \{0, 1, \ldots, J-1\}$, respectively, as

$${}_F\mathcal{F}_{ML}^{(l)}(S, pre, post, w) = \begin{cases} (2l+1)\Big((pre+post)\mathcal{F}_{SOR}^{(l)} + \mathcal{F}_d^{(l)} + \mathcal{F}_{POWER}^{(l)}\Big), & S = SOR \\ (2l+1)\Big((pre+post+1)\mathcal{F}_{POWER}^{(l)} + \mathcal{F}_d^{(l)}\Big), & S \in \{POWER, JOR\} \end{cases} \tag{4.13}$$

$${}_W\mathcal{F}_{ML}^{(l)}(S, pre, post, w) = \begin{cases} \kappa(l)\Big((pre+post)\mathcal{F}_{SOR}^{(l)} + \mathcal{F}_d^{(l)} + \mathcal{F}_{POWER}^{(l)}\Big), & S = SOR \\ \kappa(l)\Big((pre+post+1)\mathcal{F}_{POWER}^{(l)} + \mathcal{F}_d^{(l)}\Big), & S \in \{POWER, JOR\} \end{cases} \tag{4.14}$$

In (4.13), κ is a recursive function defined by

$$\kappa(l) = 2\kappa(l-1) \quad \text{for } l \in \{2, 3, \ldots, J-1\} \text{ and } \kappa(1) = 3,$$

and thus, using (4.12) and (4.13), the upper bounds on the number of floating–point operations for one F cycle and one W cycle of ML method are given, respectively as

$$\begin{aligned} {}_F\mathcal{F}_{ML} &= \sum_{l=1}^{J-1} {}_F\mathcal{F}_{ML}^{(l)}(S, pre, post, w) + \mathcal{F}_{POWER}^{(0)} + length(x_0) \\ &+ {}_V\mathcal{F}_{ML}^{(0)}(S, pre, post, w) \\ &+ J(\mathcal{F}_{POWER}^{(J)} + O(|\mathcal{N}|^3), \end{aligned} \quad (4.15)$$

$$\begin{aligned} {}_W\mathcal{F}_{ML} &= \sum_{l=1}^{J-1} {}_W\mathcal{F}_{ML}^{(l)}(S, pre, post, w) + \mathcal{F}_{POWER}^{(0)} + length(x_0) \\ &+ {}_V\mathcal{F}_{ML}^{(0)}(S, pre, post, w) \\ &+ 2^{(J-1)}(\mathcal{F}_{POWER}^{(J)} + O(|\mathcal{N}|^3). \end{aligned} \quad (4.16)$$

Assuming that we store the diagonals of Q_l seperately, diagonals of Q_l need $O(\sum_{l=0}^{J-1} length(x_l))$ storage for $l \in \{0, 1, \ldots, J-1\}$ of an ML cycle. Solution vectors x_l need $O(\sum_{l=0}^{J} length(x_l))$ storage for levels $l \in \{0, 1, \ldots, J\}$ of an ML cycle. The vectors $d_{(n,m),\psi}^{(l)}$ need $O(\sum_{l=1}^{J} \sum_{\substack{n,m \in \mathcal{N} \\ n \neq m}} d_{(n,m),\psi}^{(l)})$ storage in ML method. Generating Q_J in sparse format needs at most $O(|\mathcal{N}|^2)$ storage at the last level of ML method. Smoothers at any level of an ML cycle need $O(length(x_0))$ storage. Hence the storage space needed by the ML method is given by

$$O(\sum_{l=0}^{J} length(x_l) + \sum_{l=1}^{J} \sum_{\substack{n,m \in \mathcal{N} \\ n \neq m}} d_{(n,m),\psi}^{(l)} + |\mathcal{N}|^2).$$

4.1.5 Marie's Method

Having provided an upper bound on the number of floating–point operations performed in one cycle of the ML method, we now give an upper bound on the number of floating–point operations performed in one iteration of Marie's method summarized in Table 3.1.

In the first step, the decomposition process is carried out by Algorithms 6 and 7, and in the second step, throughput values are set to some initial value. In these steps, Algorithms 6 and 7 do not incur any floating–point operations. Yet, in the third step, in order to compute arrival rates $\lambda_j(i)$ of subnetworks at any iteration step of Marie's method from (3.5), one has to compute the values represented by $A_m(K_m)$ and then the values represented by $NC_v(u)$. For any $m \in \{1, 2, \ldots, S\}$, $A_m(K_m)$ requires at most K_m floating–point operations. Thus, computing an $NC_v(u)$ value requires at most

$$\max_{u,v}\Big(\sum_{K_1,\ldots,K_S} S(2\prod_{m=1}^{S} K_m + 1)\Big) \tag{4.17}$$

floating–point operations. Then, using (4.16), at most

$$2 + 2\max_{u,v}\Big(\sum_{K_1,\ldots,K_S} S(2\prod_{m=1}^{S} K_m + 1)\Big)$$

floating–point operations are performed to compute $\lambda_j(i)$. Since $i \in \{1, 2, \ldots, K\}$ and $j \in \{1, 2, \ldots, S\}$, at most

$$KS\Big(2 + 2\max_{u,v}\Big(\sum_{K_1,\ldots,K_S} S(2\prod_{m=1}^{S} K_m + 1)\Big)\Big) \tag{4.18}$$

floating–point operations are performed to compute all the $\lambda_j(i)$s in the third step of Table 3.1.

In the fourth step, each subnetwork is analyzed for its steady–state probabilities using the ML method. Let us denote one V, F, or W cycle of ML in subnetwork k by ${}_{\{V,F,W\}}\mathcal{F}_{ML}^{(k)}$, and number of cycles performed to compute the steady–state vector of subnetwork k by σ_k. Then using (4.11), (4.14), and (4.15) an upper bound on the number of floating–point operations performed while obtaining the steady–state vectors of subnetworks can be written as

$$\sum_{k=1}^{S} \sigma_k\big({}_{\{V,F,W\}}\mathcal{F}_{ML}^{(k)}\big). \tag{4.19}$$

The last step computes the new throughput values $\nu_j(i)$ and performs at most 4 floating–point operations for $i \in \{1, 2, \ldots, K\}$ and $j \in \{1, 2, \ldots, S\}$ Hence, using (4.17) and (4.18) an upper bound on the number of floating–point operations performed in one iteration of Marie's method turns out to be

$$4KS + KS\Big(2 + 2\max_{u,v}\Big(\sum_{K_1,\ldots,K_S} S(2\prod_{m=1}^{S} K_m + 1)\Big)\Big) + \sum_{k=1}^{S} \sigma_k(_{\{V,F,W\}}\mathcal{F}_{ML}^{(k)}).$$

In order to give an upper bound on the storage requirements of Marie's method, let us investigate (3.5). The variables $\pi_j(i)$ and $NC_j(i)$ are defined for $i \in \{0, 1, \ldots, K\}$ and $j \in \{1, 2, \ldots, S\}$, and occupies $O(S(K+1))$ storage. The variable $\nu_j(i)$ is defined for $i \in \{1, 2, \ldots, K\}$ and $j \in \{1, 2, \ldots, S\}$, and occupies $O(SK)$ storage. The variable $\lambda_j(i)$ is defined for $i \in \{0, 1, \ldots, K-1\}$ and $j \in \{1, 2, \ldots, S\}$, and occupies $O(SK)$ storage. If the upper bound on the storage requirement of the ML method for subnetwork k, which is obtained from the decomposition procedure of Marie's method, is defined by $\mathcal{B}_k$ for $k \in \{1, 2, \ldots, S\}$, then the storage requirements of Marie's method is given by

$$O\Big(S(K+1) + \sum_{k=1}^{S} \mathcal{B}_k\Big).$$

4.1.6 Yao and Buzacott's Method

To find an upper bound on the number of floating–point operations performed in one iteration of Yao and Buzacott's method, we follow the steps listed in Table 3.1 as in the previous section.

In the first step, Yao and Buzacott's method sets the state dependent exponential rates of queues to some initial values. Thus, in that step, the method performs no floating–point operations.

In the second step, the state dependent exponential network is analyzed for its steady–state vector using Gaussian elimination and this step needs at most

$O(|\mathcal{N}|^3)$ floating–point operations.

In the third step, from (3.2) we compute the state dependent arrival rates $\lambda_j(i)$ for $j \in \{1, 2, \ldots, J\}$ and $i \in \{1, 2, \ldots, c_j\}$. This computation performs 2 floating–point operations for $j \in \{1, 2, \ldots, J\}$ and $i \in \{1, 2, \ldots, c_j\}$.

In the fourth step, each queue is analyzed in isolation for its steady–state vector. If there are J queues in the closed QN and we represent one cycle of the ML method for queue k by ${}_{\{V,F,W\}}\mathcal{F}_{ML}^{(k)}$ and number of cycles performed to compute the steady–state vector of queue k by γ_k, then the maximum number of floating–point operations performed while obtaining the steady–state vectors of queues is given by

$$\sum_{k=1}^{N} \gamma_k({}_{\{V,F,W\}}\mathcal{F}_{ML}^{(k)}).$$

In the last step, Yao and Buzacott's algorithm computes new throughput values using (3.2), and thus, performs 2 floating–point operations for $j \in \{1, 2, \ldots, N\}$ and $i \in \{1, 2, \ldots, c_j\}$. Hence, for one iteration of Yao and Buzacott's method the upper bound on the number of floating–point operations is given by

$$O(|\mathcal{N}|^3) + \sum_{k=1}^{J} \gamma_k({}_{\{V,F,W\}}\mathcal{F}_{ML}^{(k)}) + 4\sum_{j=1}^{J} c_j. \tag{4.20}$$

To find an upper bound on the storage requirements of Yao and Buzacott's method, we inspect (3.6). The variable $\lambda_j(i)$ is defined for $j \in \{1, 2, \ldots, J\}$ and $i \in \{0, 1, \ldots, c_j - 1\}$, and needs $O(\sum_{j=1}^{J} c_j)$ storage. The variable $\pi_j(i)$ is defined for $j \in \{1, 2, \ldots, J\}$ and $i \in \{0, 1, \ldots, c_j\}$, and needs $O(\sum_{j=1}^{J}(c_j + 1))$ storage. The variable $\nu_j(i)$ is defined for $j \in \{1, 2, \ldots, J\}$ and $i \in \{1, 2, \ldots, c_j - 1\}$, and needs $O(\sum_{j=1}^{J} c_j)$ storage. Hence, if the storage requirement of the ML method for subnetwork k of Yao and Buzacott's method is denoted by $\mathcal{C}_k$, then the storage requirement of Yao and Buzacott's method is given by

$$O(\sum_{j=1}^{J} c_j + \sum_{k=1}^{J} \mathcal{C}_k).$$

4.2 Existence of a Fixed–Point

In this section, we prove that the fixed–point equations defined for Marie's method and Yao and Buzacott's method have unique fixed–points. In order to arrive at this result, we show that the methods' fixed–point equations satisfy the conditions of Brouwer's fixed–point theorem [30], which is stated next.

Theorem 4.1. ***(Brouwer's fixed–point theorem)*** *Let $F : A \subset \mathbb{R}^N \rightarrow \mathbb{R}^N$ be continuous on the compact, convex set A, and suppose that $F(A) \subseteq A$, where $F(A)$ stands for $\cup_{a \in A}\{F(a)\}$. Then, F has a fixed–point in A.*

We recall the definition of *communicating class* [25, p. 644].

Definition 4.1. *A subset of states in a CTMC is called a* ***communicating class*** *if all the states in this subset are reachable from each other. A communicating class $\mathcal{C}$ is* ***closed*** *if the states outside class $\mathcal{C}$ are not reachable from states in class $\mathcal{C}$.*

The next results [25, p. 645] follows from the definition of closed communicating class and is used in the next subsections.

Lemma 4.1. *The steady state vector of a CTMC as a function of some nonzero entries $\lambda_1, \lambda_2, \ldots, \lambda_k$ of the generator matrix Q is continuous at all values of $\lambda_i \geq 0$ for $i = \{1, 2, \ldots, k\}$ if for all values of $\lambda_i \geq 0$, the CTMC has exactly one closed communicating class.*

4.2.1 Marie's Method

For Marie's method, let $\nu = (\nu_1(1), \nu_1(2), \ldots, \nu_1(K), \nu_2(1), \nu_2(2), \ldots, \nu_2(K), \ldots, \nu_S(1), \nu_S(2), \ldots, \nu_S(K)) \in \mathbb{R}_+^{KS}$. Then we can write the fixed–point equation of Marie's method in (3.5) as

$$\nu = M(\nu),$$

where $M : I\!R_+^{KS} \to I\!R_+^{KS}$ is given by

$$M(\nu) = (\xi_1^{(1)}(\nu), \ldots, \xi_K^{(1)}(\nu), \xi_1^{(2)}(\nu), \ldots, \xi_K^{(2)}(\nu), \ldots, \xi_1^{(S)}(\nu), \ldots, \xi_K^{(S)}(\nu)),$$

M equals the elementwise product of two vector valued functions $M_1 : I\!R_+^{KS} \to I\!R_+^{KS}$ and $M_2 : I\!R_+^{KS} \to I\!R_+^{KS}$ such that

$$M(\nu) = M_1(\nu) \odot M_2(\nu),$$

$\odot$ denotes the elementwise product operator for two vectors of the same length,

$$\begin{aligned} M_1(\nu) &= (\xi_1^{(1,1)}(\nu), \ldots, \xi_K^{(1,1)}(\nu), \xi_1^{(1,2)}(\nu), \ldots, \xi_K^{(1,2)}(\nu), \ldots, \xi_1^{(1,S)}(\nu), \ldots, \xi_K^{(1,S)}(\nu)) \\ &= \Big(\frac{\alpha_1 NC_1(K-1)}{NC_1(K)}, \ldots, \frac{\alpha_1 NC_1(0)}{NC_1(1)}, \frac{\alpha_2 NC_2(K-1)}{NC_2(K)}, \ldots, \frac{\alpha_2 NC_2(0)}{NC_2(1)}, \ldots, \\ &\quad \frac{\alpha_S NC_S(K-1)}{NC_S(K)}, \ldots, \frac{\alpha_S NC_S(0)}{NC_S(1)}\Big), \end{aligned}$$

and

$$\begin{aligned} M_2(\nu) &= (\xi_1^{(2,1)}(\nu), \ldots, \xi_K^{(2,1)}(\nu), \xi_1^{(2,2)}(\nu), \ldots, \xi_K^{(2,2)}(\nu), \ldots, \xi_1^{(2,S)}(\nu), \ldots, \xi_K^{(2,S)}(\nu)) \\ &= (\frac{\pi_1(0)}{\pi_1(1)}, \ldots, \frac{\pi_1(K-1)}{\pi_1(K)}, \frac{\pi_2(0)}{\pi_2(1)}, \ldots, \frac{\pi_2(K-1)}{\pi_2(K)}, \ldots, \frac{\pi_S(0)}{\pi_S(1)}, \ldots, \frac{\pi_S(K-1)}{\pi_S(K)}). \end{aligned}$$

Lemma 4.2. *For $\nu \in I\!R_+^{KS}$, the function $NC_v(u)$, which is defined in (3.5), is continuous on $I\!R_+$ and $NC_v(u) \neq 0$ for $u, v \in I\!N$ and $0 \le u \le K$, $1 \le u \le S$.*

Proof. In order to show that $NC_v(u)$ is continuous on $I\!R_+$, we need to show that the function $A_j(K_j)$ defined in (3.5) is continuous on $I\!R_+$ since

$$NC_v(u) = \sum_{K_1,\ldots,K_S} \prod_{j=1}^{S} \frac{\alpha_j^{K_j}}{A_j(K_j)}.$$

Fix $u, v \in I\!N$ for some $0 \le u \le K$, $1 \le v \le S$, then $A_j(K_j) : I\!R_+ \to I\!R_+$ is defined by

$$A_j(K_j) = \begin{cases} 1, & K_j = 0 \\ \prod_{i=1}^{K_j} \nu_j(i), & K_j > 0 \end{cases}.$$

Since $\nu \in I\!R_+^{KS}$, being product of components of ν, $A_j(K_j)$ is continuous on $I\!R_+$ and $A_j(K_j) \neq 0$.

Also, α_k for $k \in \{1, 2, \dots S\}$ is defined by

$$\alpha_k = \sum_{i \in \mathcal{J}^{(k)}} x_i \left(\sum_{j \in I/\mathcal{J}^{(k)}} p_{i,j} \right),$$

where x is the unique positive left eigenvector of P, the routing probability matrix of the closed QN consisting of one communicating class. $x_j > 0$ for $j \in \mathcal{J}^{(k)}$, and thus, $\alpha_k > 0$ for $k \in \{1, 2, \dots, S\}$. Hence, $NC_v(u)$ is continuous and $NC_v(u) \neq 0$ on $I\!R_+$.

Proposition 4.1. *The fixed–point equation defined by $\nu = M(\nu)$ for Marie's method has a fixed–point.*

Proof. We show that M satisfies the conditions required in Brouwer's fixed–point theorem. Let us first show that M is continuous on $I\!R_+^{KS}$. This can be done by showing for any $\nu \in I\!R_+^{KS}$ that M is continuous in each component. Thus, for any $\xi_i^{(j)}$, $i \in \{1, 2, \dots, K\}$ and $j \in \{1, 2, \dots, S\}$, $\xi_i^{(j)} = \xi_i^{(1,j)} \xi_i^{(2,j)}$. Here, $\xi_i^{(1,j)}$ is continuous by Lemma 4.2 and $\xi_i^{(2,j)}$ is continuous by Lemma 4.1. Hence, being the product of two continuous functions, $\xi_i^{(j)}$ is continuous on $I\!R_+$. This implies M is continuous on $I\!R_+^{KS}$.

Now, let us define the set E for which $M(E) \subseteq E$. To define the set E, we use structural induction on the definition of $\xi_i^{(j)}$. Let us choose $\nu^{(0)} \in I\!R_+^{KS}$ as the initial approximation, where $\nu_j^{(0)}(i) \in [a_i^{(j)}, b_i^{(j)}]$ for some $a_i^{(j)}, b_i^{(j)} \in I\!R_+$ and $\nu^{(0)} \in E = [a_1^{(1)}, b_1^{(1)}] \times \dots \times [a_K^{(1)}, b_K^{(1)}] \times [a_1^{(2)}, b_1^{(2)}] \times \dots \times [a_K^{(2)}, b_K^{(2)}] \times \dots \times [a_1^{(S)}, b_1^{(S)}] \times \dots \times [a_K^{(S)}, b_K^{(S)}]$, where $\times$ denotes the Cartesian product operator. Then for the base case we have $\xi_i^{(j)}(\nu^{(0)}) = \xi_i^{(1,j)}(\nu^{(0)}) \xi_i^{(2,j)}(\nu^{(0)})$. Since M is continuous on $I\!R_+^{KS}$, there are intervals such that $\xi_i^{(1,j)}(\nu^{(0)}) \in [a_i^{(1,j)}, b_i^{(1,j)}]$ and $\xi_i^{(2,j)}(\nu^{(0)}) \in [a_i^{(2,j)}, b_i^{(2,j)}]$, where $[a_i^{(j)}, b_i^{(j)}] = [a_i^{(1,j)} a_i^{(2,j)}, b_i^{(1,j)} b_i^{(2,j)}]$. Therefore by definition of M, $M(\nu^{(0)}) \in E = [a_1^{(1)}, b_1^{(1)}] \times \dots \times [a_K^{(1)}, b_K^{(1)}] \times [a_1^{(2)}, b_1^{(2)}] \times \dots \times [a_K^{(2)}, b_K^{(2)}] \times \dots \times [a_1^{(S)}, b_1^{(S)}] \times \dots \times [a_K^{(S)}, b_K^{(S)}]$ and $M(\nu^{(0)}) \subseteq E$. Suppose for $\nu^{(r)}$ that there are intervals $\xi_i^{(1,j)}(\nu^{(r)}) \in [a_i^{(1,j)}, b_i^{(1,j)}]$ and $\xi_i^{(2,j)}(\nu^{(r)}) \in [a_i^{(2,j)}, b_i^{(2,j)}]$. Then by definition of $\xi_i^{(j)}$, $\xi_i^{(j)}(\nu^{(r)}) = \xi_i^{(1,j)}(\nu^{(r)}) \xi_i^{(2,j)}(\nu^{(r)})$ and this implies $\xi_i^{(j)}(\nu^{(r)}) \in [a_i^{(1,j)} a_i^{(2,j)}, b_i^{(1,j)} b_i^{(2,j)}]$. By continuity of M, this implies $\xi_i^{(j)}(\nu^{(r)}) \in$

$[a_i^{(j)}, b_i^{(j)}]$, where $[a_i^{(j)}, b_i^{(j)}]$ is defined by $[a_i^{(1,j)} a_i^{(2,j)}, b_i^{(1,j)} b_i^{(2,j)}]$. Since $M(\nu^{(r)}) = (\xi_1^{(1)}(\nu^{(r)}), \ldots, \xi_K^{(1)}(\nu^{(r)}), \xi_1^{(2)}(\nu^{(r)}), \ldots, \xi_K^{(2)}(\nu^{(r)}), \ldots, \xi_1^{(S)}(\nu^{(r)}), \ldots, \xi_K^{(S)}(\nu^{(r)}))$ and $M(\nu^{(r)}) \subseteq E$ for some $E = [a_1^{(1)}, b_1^{(1)}] \times \ldots \times [a_K^{(1)}, b_K^{(1)}] \times [a_1^{(2)}, b_1^{(2)}] \times \ldots \times [a_K^{(2)}, b_K^{(2)}] \times \ldots \times [a_1^{(S)}, b_1^{(S)}] \times \ldots \times [a_K^{(S)}, b_K^{(S)}]$.

The interval $[a_i^{(j)}, b_i^{(j)}]$ is closed and bounded in $I\!R_+$. Therefore, by the proposition in [34, p. 54], $[a_i^{(j)}, b_i^{(j)}]$ is compact. This implies by the theorem in [34, p. 58], E is also compact. Being a closed interval, $[a_i^{(j)}, b_i^{(j)}]$ is convex, and by the discussion in [35, p. 28], E is also convex. Hence, M satisfies all conditions of Brouwer's fixed–point theorem on E, and therefore, a fixed–point $\nu \in E$ of M exists.

4.2.2 Yao and Buzacott's Method

Let us define the fixed–point equation used by Yao and Buzacott's method. Let $\nu = (\nu_1(1), \ldots, \nu_1(c_1), \nu_2(1), \ldots, \nu_2(c_2), \ldots, \nu_J(1), \ldots, \nu_J(c_J)) \in I\!R_+^{\sum_{i=1}^J c_i}$, then we can write the fixed–point equation of Yao and Buzacott's method in (3.6) as

$$\nu = YB(\nu),$$

where $YB : I\!R_+^{\sum_{i=1}^J c_i} \to I\!R_+^{\sum_{i=1}^J c_i}$ is given by

$$YB(\nu) = (\zeta_1^{(1)}(\nu), \ldots, \zeta_{c_1}^{(1)}(\nu), \zeta_1^{(2)}(\nu), \ldots, \zeta_{c_2}^{(2)}(\nu), \ldots, \zeta_1^{(J)}(\nu), \ldots, \zeta_{c_J}^{(J)}(\nu)),$$

YB equals the elementwise product of two vector valued functions $YB_1 : I\!R_+^{\sum_{i=1}^J c_i} \to I\!R_+^{\sum_{i=1}^J c_i}$ and $YB_2 : I\!R_+^{\sum_{i=1}^J c_i} \to I\!R_+^{\sum_{i=1}^J c_i}$ such that

$$YB(\nu) = YB_1(\nu) \odot YB_2(\nu),$$

$$\begin{aligned} YB_1(\nu) &= (\zeta_1^{(1,1)}(\nu), \ldots, \zeta_{c_1}^{(1,1)}(\nu), \zeta_1^{(1,2)}(\nu), \ldots, \zeta_{c_2}^{(1,2)}(\nu), \ldots, \zeta_1^{(1,J)}(\nu), \ldots, \zeta_{c_J}^{(1,J)}(\nu)) \\ &= \Big(\nu_1(1)\frac{\pi_1^{exp}(1)}{\pi_1^{exp}(0)}, \ldots, \nu_1(c_1)\frac{\pi_1^{exp}(c_1)}{\pi_1^{exp}(c_1-1)}, \nu_2(1)\frac{\pi_2^{exp}(1)}{\pi_2^{exp}(0)}, \ldots, \nu_2(c_2)\frac{\pi_2^{exp}(c_2)}{\pi_2^{exp}(c_2-1)}, \ldots, \\ &\quad \nu_J(1)\frac{\pi_J^{exp}(1)}{\pi_J^{exp}(0)}, \ldots, \nu_J(c_J)\frac{\pi_J^{exp}(c_J)}{\pi_J^{exp}(c_J-1)}\Big), \end{aligned}$$

and

$$\begin{aligned} YB_2(\nu) &= (\zeta_1^{(2,1)}(\nu),\ldots,\zeta_{c_1}^{(2,1)}(\nu),\zeta_1^{(2,2)}(\nu),\ldots,\zeta_{c_2}^{(2,2)}(\nu),\ldots,\zeta_1^{(2,J)}(\nu),\ldots,\zeta_{c_J}^{(2,J)}(\nu)) \\ &= (\frac{\pi_1(0)}{\pi_1(1)},\ldots,\frac{\pi_1(c_1-1)}{\pi_1(c_1)},\frac{\pi_2(0)}{\pi_2(1)},\ldots,\frac{\pi_2(c_2-1)}{\pi_2(c_2)},\ldots,\frac{\pi_J(0)}{\pi_J(1)},\ldots,\frac{\pi_J(c_J-1)}{\pi_J(c_J)}). \end{aligned}$$

Proposition 4.2. *The fixed–point equation defined by $\nu = YB(\nu)$ for Yao and Buzacott's method has a fixed–point.*

Proof. The proof follows in the same way as Proposition 4.1.

The next chapter discusses implementation issues associated with the methods.

Chapter 5

Implementation

The software tool [27] is coded in MATLAB [14] and can be used with MATLAB versions 5.3 (R11) and later. The tool is capable of analyzing closed QNs with phase–type service distributions and arbitrary buffer sizes. In that respect, the tool possesses six different steady–state analysis methods from the literature. These are the ML method [11], Marie's method [26], Yao and Buzacott's method [45], Akyıldız's mean value analysis (MVABLO) [2], the convolution algorithm [13], and Power, JOR and SOR methods, which are classical iterative methods based on splittings [39]. Although vectorization of computations increases speed of program execution considerably, we refrain from using dense vectorization since we work in sparse storage and we do not want to increase memory usage. Algorithms coded in this way become much more self–descriptive within the simple coding environment of MATLAB. In the following paragraphs, we talk briefly about implementation details of some important m–files and variables in the software tool.

We first start by introducing the most important variables that are designed to be used with the ML method and smoothers. These are the structure `subNet`, the cell array `LLM`, the global cell arrays `X` and `dQ`, the global arrray `Y`, the global sparse array `CM`, the global array `dV`, and arrays `phv` and `bv`. Before the call to `mlDriver.m`, these variables are constructed and the memory needed to represent them is allocated through the initialization process of methods. The structure `subNet` consists of six arrays: `partX`, `iVec`, `jVec`, `convVec`, `table`, and `strtInd`. The two dimensional array `partX` of size $(J \times (|\mathcal{N}| + 1))$ holds the partitions of iteration vectors with respect to HLM states for each level of an ML cycle. The arrays `iVec` and `jVec` are used to exploit the nonzero structure of the HLM matrix. Thus, if we represent the number of nondiagonal transitions in the HLM matrix by *#ndt*, then array `iVec` of length *#ndt* holds row indices of nondiagonal nonzero blocks of the generator matrix Q in columnwise order, where indices are enumerated by the integers in $\{1, 2, \ldots, |\mathcal{N}|\}$. The indices that correspond to diagonal blocks of Q are not stored in `iVec`. The array `jVec` of length $|\mathcal{N}|$ points to the end of each column in the array `iVec`. The array `convVec` of size $|\mathcal{N}|$ holds the lexicographical orders of HLM states. The array `convVec` together with `iVec` are used in the m–file `index2vec.m` when constructing vectors that represent the HLM states in $\mathcal{N}$. The two–dimensional sparse array `table` of size $(\#ndt \times J)$ holds pointers to elements of `dV` which are used to define the blocks of aggregated matrices of Q through levels of an ML cycle as in (2.6). The array `strtInd` of length $(J - 1)$ points to the first pointers of each level, except the coarsest one, in the array `table`. The global array `dV` holds the values defined by (2.5). The two–dimensional cell array `LLM` of size $(J \times 5)$ contains vectors and matrices that represent the phase–type service distributions of queues. That is, if *tril* and *triu* represent strictly lower– and upper–triangular parts of a matrix, then for $i \in \{1, 2, \ldots, J\}$

$$\texttt{LLM}\{i,j\} = \begin{cases} \alpha & \text{if } j = 1 \\ T_0 & \text{if } j = 2 \\ tril(T) & \text{if } j = 3 \text{ and } p_{i,i} = 0 \\ tril(T + p_{i,i}(T_0 \times \alpha)) & \text{if } j = 3 \text{ and } p_{i,i} > 0 \\ T & \text{if } j = 4 \text{ and } p_{i,i} = 0 \\ T + p_{i,i}(T_0 \times \alpha) & \text{if } j = 4 \text{ and } p_{i,i} > 0 \\ triu(T) & \text{if } j = 5 \text{ and } p_{i,i} = 0 \\ triu(T + p_{i,i}(T_0 \times \alpha)) & \text{if } j = 5 \text{ and } p_{i,i} > 0 \end{cases}.$$

All matrices in `LLM` are held in sparse format. Matrices with all zero entries and identity matrices are not stored in `LLM`. The cell arrays `X` and `dQ` hold maximum length iteration vectors and row sum vectors for each level of an ML cycle, respectively. `X` and `dQ` have the same length and the lengths of the vectors they hold in each cell are also equal. It is important to note that the lengths of the vectors in cells reduce from $|\mathcal{S}|$ to $|\mathcal{N}|$ due to the aggregation procedure in the ML method. The array `Y` of length equal to `X{1}` is used as an auxiliary array in iterative methods. The two–dimensional sparse array `CM` of size $(|\mathcal{N}| \times |\mathcal{N}|)$ is used to hold the coarsest matrix Q_J at the coarsest level of an ML cycle. The arrays `phv` and `bv` hold phase sizes and buffer sizes of queues, respectively.

The m–file `calculateDiag.m` computes the row sums, represented by $D(n, n)$ for all $n \in \mathcal{N}$ in chapter 2, of a matrix represented by Kronecker products. The processing scheme in `calculateDiag.m` is important in that the iterative methods Power, JOR, and SOR use almost the same processing scheme with little differences. The three main steps of `calculateDiag.m` on nonzero blocks of the generator matrix Q can be described as follows. First, the m–file `index2vec.m` is used to obtain vectors describing the HLM states corresponding to a given nonzero block of Q. Second, the interaction between these states is revealed using the obtained vectors and the routing probability matrix P, and then the resulting data is provided to the m–file `inputData.m`. The m–file `inputData.m` returns pointers to the matrices to be used in the Kronecker product–vector multiplication and order of left and right identity matrices which are needed by the multiplication algorithm implemented in the m–file `kPvM.m`. Third, the m–file `kPvM.m` obtained by modifying the vector–Kronecker product multiplication algorithm (see Algorithm 3) computes the row sum in the block defined by the data obtained from `inputData.m` and returns the resulting vector. Although, one can compute the row sum of a matrix defined by Kronecker products by passing a vector of all ones with appropriate size and data from `inputData.m` into `kPvM.m`, it is not efficient memorywise since the vector of all ones occupies memory. In practice, `kPvM.m` does not take a vector as an input argument when computing the row sum of a Kronecker product, so the vector is not stored in memory. In `calculateDiag.m`, the elements of the resulting vector are added to the part of array `dQ` corresponding to the HLM state obtained from `index2vec.m`.

The iterative methods Power, JOR and SOR are coded in the m–files `kPM.m`, `kJOR.m`, and `kSOR.m`. The difference is in the step where the iterative methods use vector–Kronecker product multiplication algorithm in order to compute the next iteration vector. Although the Power and JOR methods can be implemented easily just by looking at their definition, implementation of the SOR method turns out to be more complicated. The difficulty arises from the fact that SOR method uses the new estimates as soon as they are computed (see [23, p. 339]). To this end, the m–file `kSOR.m` uses Algorithm 8 which is implemented in the m–file `newIterVec.m`. The solution through iterative methods is implemented in the m–file `methodITER.m`. The m–file `methodITER.m` use exactly the same m–files `kPM.m`, `kJOR.m`, and `kSOR.m` which provide the implementations for smoothers of the ML method. Thus, `methodITER.m` iterates only at the first level of an ML cycle until a predefined error tolerance for the residual vector is met.

The ML method implemented in the software tool is capable of aggregating LLMs in fixed or circular orders using V, F, or W cycles. It is similar to [11] and is composed of two m–files: the driver m–file `mlDriver.m` and the recursive m–file `ml.m`. As mentioned in chapter 3, the m–file `mlDriver.m` is implemented into the solution phases of Marie's method and Yao and Buzacott's method, which are coded in m–files `methodMARIE.m` and `methodYB.m`, respectively. Having introduced the ML method in chapter 2, we now briefly talk about the consequences of a call to the m–file `mlDriver.m` when the cycle type is V. When we call `mlDriver.m`, diagonal elements of the generator matrix are computed and assigned to `dQ{1}` by `calculateDiag.m`, `X{1}` is pre–smoothed using the chosen iterative method, and `ml.m` is called. The m–file `ml.m` proceeds as follows through an ML cycle. In the first step, the m–file `genXf2c.m` constructs the coarser iteration vector and assigns it to `X{2}`. In the second step, using the vectors in `X{1}` and `X{2}`, the m–file `calDiagVecs.m` computes the new elements of `dV` and assigns them to their corresponding locations in `dV` using `table`. In the third step, the m–file `calculateDiag.m` computes the diagonal elements of the coarser generator matrix Q_1 and assigns them to `dQ{2}`. In the fourth step, the vector in `X{2}` is smoothed using the specified iterative method. In the last step, `ml.m` is called for the next coarser level. At the coarsest level, after computing the diagonal elements of the coarsest generator matrix Q_J, the m–file `genCoarsest.m` constructs the coarsest generator matrix Q_J, replaces its last column with ones, and assigns it to the sparse array `CM`. Then the linear system (2.4) with this coefficient matrix and sparse left–hand side vector $(0, 0, \ldots, 0, 1)$ is solved using the backslash operator "\" of MATLAB if $|\mathcal{N}|$ is less than 500, or is solved using the built–in function `bicgstab` of MATLAB otherwise. Unfortunately, `bicgstab` method may not converge to a solution for a specified tolerance. In order to achieve convergence, one may need to employ a preconditioner with `bicgstab`. To this end, the preconditioner is provided from the incomplete LU factorization of the coefficient matrix and can be obtained by using the MATLAB built–in function `luinc` with a suitable tolerance. At this point, recursion starts to backtrack to finer levels. The m–file `ml.m` uses the m–file `genXc2f.m` to construct iteration vectors for finer levels and these vectors are post–smoothed using the specified iterative method. At the finest level, `ml.m` returns the new iteration vector `X{1}` to `mlDriver.m`. After another smoothing operation in `mlDriver.m`, the residual of `X{1}` is computed by the m–file `calResidual.m`. Then the 1–norm of the residual vector is computed and the m–file `mlDriver.m` either stops and returns the result if the predefined error tolerance is met, or makes another call to the recursive m–file `ml.m`.

The approximative decompositional methods described in chapter 3 are implemented in m–files `convolutionAlgo.m`, `methodMVA.m`, `methodMARIE.m`, and `methodYB.m`. Being direct methods to compute performance measures of closed QNs, the convolution algorithm, which is implemented in the m–file `convolutionAlgo.m`, and Akyildiz's mean value analysis tool MVABLO, which is implemented in the m–file `methodMVABLO.m`, are coded into the software tool using Algorithms 4 and 5, respectively. Being inspired by Marie's method, the implementation of Yao and Buzacott's method is almost the same as Marie's method. For that reason, first we explain the steps of the m–file `methodMARIE.m` and then explain the m–file `methodYB.m` using the steps in `methodMARIE.m`. The m–file `methodMARIE.m` starts by reading the data defined for the network from a given directory using `readP.m`, which reads the routing probability matrix, and then generates the cell array `LLM` using `genLLMs.m`. After that, using Algorithm 7, which is implemented in the m–file `netDecompose.m`, the m–file `genJ.m` decomposes the network into subnetworks. After the decomposition procedure, the data structure `subNet` is generated by the m–file `genSubNetStructs.m`. Maximum space needed for the variables `X`, `dQ`, and `CM` are computed for each subnetwork and allocated once. Finally, Marie's method is implemented as in [26, p. 534] together with the m–file `mlDriver.m`, which is used in Step 4 of Marie's method in Table 3.1. Implementation of Yao and Buzacott's method follows similar steps; however, there is one main difference. Yao and Buzacott's method decomposes the network into subnetworks of single queues and for that reason `genJ.m` does not need to be called in `methodYB.m`. Yao and Buzacott's method is implemented as in [45, p. 412] together with the m–file `mlDriver.m`, which is used in Step 4 of Yao and Buzacott's method in Table 3.1. The state dependent exponential network mentioned in Step 2 of Yao and Buzacott's method in Table 3.1 is constructed by the m–file `genEN.m` and assigned to the sparse array `CM` with ones in the last column. Then `CM` is analyzed for its steady–state vector using the backslash operator "\" with the left–hand side vector $(0, 0, \ldots, 0, 1)$ if $|\mathcal{N}|$ is less than 500, or the built–in function `bicgstab` otherwise.

In the next chapter, we present the results of numerical experiments on five problems.

Chapter 6

Numerical Results

Numerical experiments are performed on five problems using the implemented methods in the software tool. The methods are compared for their accuracy and efficiency. When used in a method, the ML method assumed a stopping tolerance of 10^{-15} on the residual 1–norm. Experiments are conducted using two configurations of the ML method. These are V cycle with fixed aggregation order and F cycle with circular aggregation order. The results of the configuration which performs a smaller number floating–point operations are reported. ML method uses SOR with relaxation parameter 1.0 as smoother and it performs 1 pre– and 1 post–smoothing in all of the experiments. We used approximation tolerance of 10^{-4} for both Marie's method and Yao and Buzacott's method in the experiments. We set the maximum number of iterations for Marie's and Yao and Buzacott's method to 50. Thus, the methods stop when either the tolerance is met or the number of iterations reach 50. When computing the steady–state vector of the state dependent exponential network in Yao and Buzacott's method and the steady–state vector of the coarsest matrix in ML method, we use Gaussian elimination if the matrices' order is less than 500, or BiCGstab with ILU preconditioning and a drop tolerance of 10^{-5} if the matrices' order is greater than 500. Exact solutions of the problems are obtained via the ML method and SOR method with relaxation parameter 1.0 for all problems. Both of the methods assume a tolerance of 10^{-15} on the residual norm, and the iteration is stopped if the desired tolerance is not obtained within 1,000 iterations or 100,000 seconds. The convolution algorithm and Akyildiz's mean value analysis have no stopping criterion since they are direct methods and they stop when the approximation finishes for all the queues in a problem. We have the results of utilizations and mean lengths of queues in each problem using the methods in the software tool. Results obtained by approximative methods are compared with results of the ML method and relative errors are provided using 1–norm. We ran all the experiments on a Pentium 3.0 GHz with 1 GB of memory. All the methods and algorithms in the software tool are implemented in m–files using MATLAB [14].

The problems that are used in the experiments are closed QNs with 3, 6, and 8 queues. Problems 2 to 4 are analyzed with 5, 6, 7, 8 customers and problem 5 is analyzed with 6, 7, 8, 9 customers. Since buffer sizes of queues in the closed QNs may be finite, we have different subnetwork topologies in Marie's method. With regards to this, we considered locally/globally balanced service demands in the subnetworks. Each problem is defined by its routing probabilities among queues and its queues' phase–type service distributions. Routing probabilities among queues in the problems are given on figures depicting the closed QNs topology. The problems considered assume three types of phase–type service distributions. These are hypoexponential and hyperexponential distributions with 2 phases and Erlang distribution with 5 phases. However, this should not be understood to mean that the software tool is able to work with only these numbers of phases.

Let the transition rate matrices of hyperexponential distribution with 2 phases, hypoexponential distribution with 2 phases, and Erlang distribution with 5 phases be represented respectively as

$$T^{(j)}_{Hyper} = \begin{pmatrix} -\mu^{(j)}_1 & 0 \\ 0 & -\mu^{(j)}_2 \end{pmatrix}, \qquad T^{(j)}_{Hypo} = \begin{pmatrix} -\mu^{(j)}_1 & \mu^{(j)}_1 \\ 0 & -\mu^{(j)}_2 \end{pmatrix}$$

and

$$T^{(j)}_{Erlang} = \begin{pmatrix} -\mu^{(j)} & \mu^{(j)} & 0 & 0 & 0 \\ 0 & -\mu^{(j)} & \mu^{(j)} & 0 & 0 \\ 0 & 0 & -\mu^{(j)} & \mu^{(j)} & 0 \\ 0 & 0 & 0 & -\mu^{(j)} & \mu^{(j)} \\ 0 & 0 & 0 & 0 & -\mu^{(j)} \end{pmatrix}$$

for some $j \in \{1, 2, \ldots, J\}$. Then Hyper$^{(j)}(\mu_1^{(j)}, \mu_2^{(j)}, \alpha^{(j)})$, Hypo$^{(j)}(\mu_1^{(j)}, \mu_2^{(j)})$, and Erlang$^{(j)}(\mu^{(j)}, t)$ represent hyperexponential and hypoexponential distributions with 2 phases and Erlang distribution with t phases, respectively. The initial distribution vector $\alpha^{(j)}$ is not given for hypoexponential and Erlang distributions since they have initial distribution vectors of the form $\alpha^{(j)} = (1, 0, \ldots, 0)$ with appropriate length. The two other parameters that can change for the closed QN are the number of customers in the network and buffer sizes of queues. The number of customers in the closed QN is defined by K and the buffer sizes of queues are defined by the vector b such that b_j denotes the buffer size of queue j for $j = \{1, 2, \ldots, N\}$. These parameters appear in the caption of the table, which presents the results of the corresponding problem.

Tables presenting the results of the problems consist of six rows and eight columns. Rows of the table correspond to convolution algorithm (CA), Akyildiz's mean value analysis (MVABLO), Marie's method (M), Yao and Buzacott's (YB), ML method (ML), and SOR method (SOR), respectively. The parantheses by the method names include information about parameters defined for the methods. The parameters for the ML, Marie's, and Yao and Buzacott's methods indicate cycle type and aggregation order, respectively. Columns correspond to number of iterations performed by methods (oIter), average number of iterations performed by ML method for M and YB methods and average number of smoothings at the finest level for the ML method (iIter), time taken by methods in seconds (T), number of floating point operations in megaflops (MF) performed by methods, memory requirement of the methods in megabytes (MB), relative error of utilization of queues (RE(ρ)), relative error of mean queue lengths (RE(E[X])), and 1-norm of the residual vector for the ML and SOR methods, respectively. An asterisk by oIter or T values indicate that predefined upper bounds for these values are reached. Timing results of methods are given for demonstration purposes only. Indeed, experiments in MATLAB should not be expected to run in times that are consistent with flop count analyses. One can find a detailed explanation of this situation in [20].

6.1 Problem 1 (Marie's Example)

Our first problem is the first example in Marie's paper [26, p. 536]. This small problem consist of 3 queues having two Erlang service distributions with two phases and a hyperexponential service distribution with two phases. This problem is included to demonstrate that our extension of M and YB to include phase–type service distributions works correctly. The number of customers in the network is $K = 6$ and queues have infinite buffer sizes with $b = (6, 6, 6)$. Because of the fact that queues have infinite buffer sizes, decomposition in the closed QN is maximal and every queue is treated as a subnetwork in M and YB methods. The topology of the closed QN can be seen in Figure 6.1, and under this setting, the network has 28 HLM states with a state space size of 146. The number of nonzero elements needed for the sparse representation of Q is 820. Results of the methods for this problem are given in Tables 6.1, 6.2, and 6.3. In Table 6.1, we present the results for the problem without introducing any modifications on the example's real parameters. In Tables 6.2 and 6.3, we provide the results obtained for the balanced and unbalanced cases, respectively.

When we use the M and YB methods, we obtain 3 digits of accuracy results for the utilization values and 2 digits of accuracy for the mean queue length values in the closed QN. We see that CA and MVA cannot beat the results of M and YB, since they hardly attain 1 digit of accuracy for both of the performance measures. Thus, the M and YB methods compute the two performance measures more accurately than CA and MVA. On the other hand, when we compare the efficiency of the methods, we see that none of the methods M, YB, and SOR present better flop counts than that of the ML method. In that respect, the ML

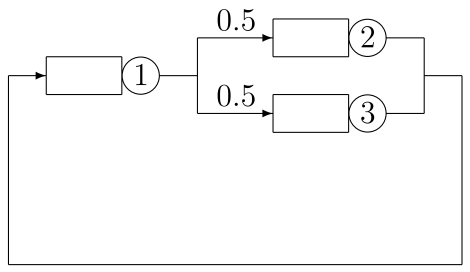

Figure 6.1: Problem 1 (Marie's example).

method is the most efficient method among the four iterative methods since it attains a residual norm close to machine precision in a smaller number of flops.

Table 6.1: $K = 6$, $b = (6,6,6)$, Hyper$^{(1)}(1.990049503712809, 9.950496287190580e–3, (9.950247518564047e-1, 4.975248143595290e-3))$, Erlang$^{(2)}(1.0, 2)$, Erlang$^{(3)}(2.0, 2)$

	oIter	iIter	T	MF	MB	RE(ρ)	RE(E[X])	Res
CA	n/a	n/a	0	0	0.0	2e−1	8e−2	n/a
MVABLO	n/a	n/a	0	0	0.0	2e−1	8e−2	n/a
M(F,CIRCULAR)	9	9	8	2	0.0	2e−3	1e−2	n/a
YB(F,CIRCULAR)	9	9	8	2	0.0	2e−3	1e−2	n/a
ML(F,CIRCULAR)	11	2	7	1	0.0	n/a	n/a	1e−16
SOR	252	n/a	21	5	0.0	n/a	n/a	1e−15

When we look at the results of the balanced case in Table 6.2, we see that the M and YB methods produce 2 digits of accuracy, whereas CA and MVABLO hardly attain 1 digit of accuracy for both of the performance measures. Although these results show that the accuracy of the M and Y methods is still better, the ML method is to be preferred over the other three iterative methods.

The results of the unbalanced case are presented in Table 6.3. For this case, the M and YB methods approximate the performance measures more accurately than CA and MVABLO. Yet, SOR performs much better than the ML method, which is better than the M and YB methods. Thus, this makes SOR the most accurate and efficient iterative method for the unbalanced case.

Table 6.2: $K = 6$, $b = (6{,}6{,}6)$, $\text{Hyper}^{(1)}(1.990049503712809, 9.950496287190580\text{e–}3, (9.950247518564047\text{e} - 1, 4.975248143595290\text{e} - 3))$, $\text{Erlang}^{(2)}(1.0, 2)$, $\text{Erlang}^{(3)}(1.0, 2)$

	oIter	iIter	T	MF	MB	RE(ρ)	RE(E[X])	Res
CA	n/a	n/a	0	0	0.0	2e−1	8e−2	n/a
MVABLO	n/a	n/a	0	0	0.0	3e−1	8e−2	n/a
M(F,CIRCULAR)	7	10	8	1	0.0	1e−2	3e−2	n/a
YB(F,CIRCULAR)	7	10	7	1	0.0	1e−2	3e−2	n/a
ML(F,CIRCULAR)	12	2	6	1	0.0	n/a	n/a	3e−16
SOR	107	n/a	9	2	0.0	n/a	n/a	8e−16

Table 6.3: $K = 6$, $b = (6{,}6{,}6)$, $\text{Hyper}^{(1)}(1.990049503712809, 9.950496287190580\text{e–}3, (9.950247518564047\text{e} - 1, 4.975248143595290\text{e} - 3))$, $\text{Erlang}^{(2)}(10.0, 2)$, $\text{Erlang}^{(3)}(0.1, 2)$

	oIter	iIter	T	MF	MB	RE(ρ)	RE(E[X])	Res
CA	n/a	n/a	0	0	0.0	2e−2	5e−2	n/a
MVABLO	n/a	n/a	0	0	0.0	2e−2	5e−2	n/a
M(F,CIRCULAR)	18	34	71	13	0.0	7e−3	3e−3	n/a
YB(F,CIRCULAR)	18	34	71	13	0.0	7e−3	3e−3	n/a
ML(F,CIRCULAR)	42	2	21	5	0.0	n/a	n/a	3e−16
SOR	81	n/a	7	2	0.0	n/a	n/a	8e−16

Because of the small size of the problem, neither M nor YB methods can be viewed as efficient or useful in any version of the problem. Yet, we will see that the M and YB methods turn out to be more valuable as the size of the problem increases.

6.2 Problem 2

The second problem consists of 6 tandem queues and has the topology in Figure 6.2. The closed QN in this problem possesses two hypoexponential service distributions each with two phases, two hyperexponential service distributions each with two phases, and two Erlang service distribution each with five phases. Experiments are performed with 5, 6, 7, 8 customers, and we obtained different number of subnetworks possessing different number of queues in the M method.

Buffer sizes of queues are given by $b = (8, 5, 8, 6, 8, 7)$. Thus for 5, 6, 7, 8 customers, the M method decomposes the QN into subnetworks given by the sets {{1},{2},{3},{4},{5},{6}}, {{1,2},{3},{4},{5},{6}}, {{1,2},{3,4},{5},{6}} and {{1,2},{3,4},{5,6}}, respectively. With this setting for 5, 6, 7, 8 customers, the closed QN has state space sizes of 8,070, 19,938, 43,320, 85,102 and HLM number of states of 252, 461, 785, 1,259, respectively. The number of nonzero elements needed for the sparse representation of networks for 5, 6, 7, 8 customers are 41,604, 111,809, 258,996, and 535,035, respectively. Results of the methods for this closed QN are presented in Tables 6.4, 6.5, 6.6, 6.7 for the balanced case and in Tables 6.8, 6.9, 6.10, 6.11 for the unbalanced case.

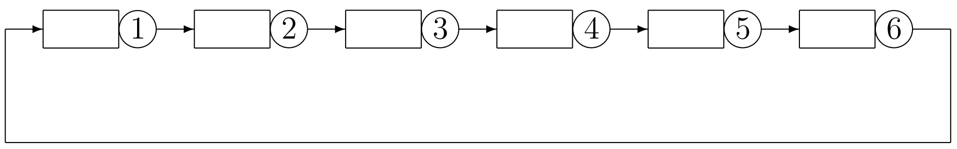

Figure 6.2: Problem 2.

In Tables 6.4, 6.5, 6.6, 6.7 we see that M and YB methods provide roughly 1.5 digits of accuracy for both performance measures. On the other hand, for the same measures, CA and MVABLO achieve less accuracy. Having performed less than one third of the number of flops compared to YB, M becomes the most accurate and efficient method between the M and YB methods for the balanced case. When we compare the results of ML and SOR methods, we see that ML performs better than SOR in terms of accuracy and efficiency by meeting the tolerance of 10^{-15} in at most 20 outer iterations. In all cases, SOR is not able to convergence within 1,000 iterations. In this set of problems, it is worthwhile to use the M and YB methods since they yield a solution (albeit in less accuracy) in shorter time and less space than the ML method. It is important to note that the outer iteration counts of the ML method increases linearly by increasing number of customers. Although outer iteration counts of M and YB methods do not change for $K \in \{5, 6, 7, 8\}$, a similar behavior to that of ML can be seen in the inner iteration counts of M and YB methods. Hence for this case of the problem, subnetworks generate more difficult problems to solve for increasing number of customers.

Table 6.4: $K = 5$, $b = (8, 5, 8, 6, 8, 7)$, $\text{Hypo}^{(1)}(3.0, 1.5)$, $\text{Erlang}^{(2)}(5.0, 5)$, $\text{Hypo}^{(3)}(1.2, 4.0)$, $\text{Hyper}^{(4)}(1.0, 1.0, (0.25, 0.75))$, $\text{Erlang}^{(5)}(5.0, 5)$, $\text{Hyper}^{(6)}(0.8, 0.8, (0.9, 0.1))$

	oIter	iIter	T	MF	MB	RE(ρ)	RE(E[X])	Res
CA	n/a	n/a	0	0	0.0	9e−2	7e−2	n/a
MVABLO	n/a	n/a	0	0	0.0	9e−2	7e−2	n/a
M(F,CIRCULAR)	5	4	6	1	0.0	4e−2	3e−2	n/a
YB(F,CIRCULAR)	5	4	8	5	0.1	4e−2	3e−2	n/a
ML(V,FIXED)	15	2	190	68	0.5	n/a	n/a	1e−16
SOR	1,000*	n/a	2,929	1,076	0.2	3e−4	4e−4	1e−3

Table 6.5: $K = 6$, $b = (8, 5, 8, 6, 8, 7)$, $\text{Hypo}^{(1)}(3.0, 1.5)$, $\text{Erlang}^{(2)}(5.0, 5)$, $\text{Hypo}^{(3)}(1.2, 4.0)$, $\text{Hyper}^{(4)}(1.0, 1.0, (0.25, 0.75))$, $\text{Erlang}^{(5)}(5.0, 5)$, $\text{Hyper}^{(6)}(0.8, 0.8, (0.9, 0.1))$

	oIter	iIter	T	MF	MB	RE(ρ)	RE(E[X])	Res
CA	n/a	n/a	0	0	0.0	1e−1	6e−2	n/a
MVABLO	n/a	n/a	0	0	0.0	9e−2	8e−2	n/a
M(F,CIRCULAR)	5	5	24	6	0.0	3e−2	2e−2	n/a
YB(F,CIRCULAR)	5	5	12	21	0.1	4e−2	3e−2	n/a
ML(V,FIXED)	16	2	456	118	1.2	n/a	n/a	8e−16
SOR	1,000*	n/a	6,980	2,761	0.5	1e−4	2e−4	3e−4

Table 6.6: $K = 7$, $b = (8, 5, 8, 6, 8, 7)$, $\text{Hypo}^{(1)}(3.0, 1.5)$, $\text{Erlang}^{(2)}(5.0, 5)$, $\text{Hypo}^{(3)}(1.2, 4.0)$, $\text{Hyper}^{(4)}(1.0, 1.0, (0.25, 0.75))$, $\text{Erlang}^{(5)}(5.0, 5)$, $\text{Hyper}^{(6)}(0.8, 0.8, (0.9, 0.1))$

	oIter	iIter	T	MF	MB	RE(ρ)	RE(E[X])	Res
CA	n/a	n/a	0	0	0.0	1e−1	1e−1	n/a
MVABLO	n/a	n/a	0	0	0.0	9e−2	9e−2	n/a
M(F,CIRCULAR)	5	6	45	11	0.1	3e−2	3e−2	n/a
YB(V,FIXED)	5	8	20	33	0.1	4e−2	3e−2	n/a
ML(V,FIXED)	18	2	1,160	425	2.4	n/a	n/a	8e−16
SOR	1,000*	n/a	14,740	6,159	1.1	3e−5	4e−5	9e−5

Table 6.7: $K = 8$, $b = (8, 5, 8, 6, 8, 7)$, Hypo$^{(1)}(3.0, 1.5)$, Erlang$^{(2)}(5.0, 5)$, Hypo$^{(3)}(1.2, 4.0)$, Hyper$^{(4)}(1.0, 1.0, (0.25, 0.75))$, Erlang$^{(5)}(5.0, 5)$, Hyper$^{(6)}(0.8, 0.8, (0.9, 0.1))$

	oIter	iIter	T	MF	MB	RE(ρ)	RE(E[X])	Res
CA	n/a	n/a	0	0	0.0	1e−1	2e−1	n/a
MVABLO	n/a	n/a	0	0	0.0	9e−2	9e−2	n/a
M(F,CIRCULAR)	5	9	84	21	0.1	2e−2	4e−2	n/a
YB(V,FIXED)	5	8	32	199	0.2	4e−2	4e−2	n/a
ML(V,FIXED)	20	2	2,142	919	4.6	n/a	n/a	7e−16
SOR	1,000*	n/a	28,339	12,328	2.1	2e−5	3e−5	4e−5

For the unbalanced case, M and YB methods achieve better results than CA and MVABLO by converging within 3 iterations and yielding at least 2.5 digits of accuracy for utilization and mean queue length values. This is at least 1 digit better than the results obtained with CA and MVABLO. Between M and YB, M is faster, but YB is more accurate except for $K = 8$. If we consider the results of SOR and ML methods, we see that SOR does not converge within 1,000 iterations, while ML method convergences within 6 outer iterations for all numbers of customers. The increase in the number of customers almost has no effect on the outer iterations done by the ML method. This behavior also can be seen in the inner iteration counts of M and YB methods. Again, it is worthwhile to use the M and YB methods.

Table 6.8: $K = 5$, $b = (8,5,8,6,8,7)$, Hypo$^{(1)}(3.0, 0.1)$, Erlang$^{(2)}(25.0, 5)$, Hypo$^{(3)}(70.0, 60.0)$, Hyper$^{(4)}(0.7, 0.7, (0.25, 0.75))$, Erlang$^{(5)}(0.1, 5)$, Hyper$^{(6)}(0.001, 0.001, (0.9, 0.1))$

	oIter	iIter	T	MF	MB	RE(ρ)	RE(E[X])	Res
CA	n/a	n/a	0	0	0.0	4e−7	4e−4	n/a
MVABLO	n/a	n/a	0	0	0.0	4e−7	4e−4	n/a
M(F,CIRCULAR)	3	1	2	0	0.0	5e−10	1e−6	n/a
YB(F,CIRCULAR)	3	1	3	4	0.1	5e−10	1e−6	n/a
ML(V,FIXED)	5	2	64	23	0.5	n/a	n/a	2e−16
SOR	1,000*	n/a	2,929	1,076	0.2	4e−3	2e−3	7e−5

Table 6.9: $K = 6$, $b = (8,5,8,6,8,7)$, $\text{Hypo}^{(1)}(3.0, 0.1)$, $\text{Erlang}^{(2)}(25.0, 5)$, $\text{Hypo}^{(3)}(70.0, 60.0)$, $\text{Hyper}^{(4)}(0.7, 0.7, (0.25, 0.75))$, $\text{Erlang}^{(5)}(0.1, 5)$, $\text{Hyper}^{(6)}(0.001, 0.001, (0.9, 0.1))$

	oIter	iIter	T	MF	MB	RE(ρ)	RE(E[X])	Res
CA	n/a	n/a	0	0	0.0	2e−8	4e−4	n/a
MVABLO	n/a	n/a	0	0	0.0	2e−8	4e−4	n/a
M(F,CIRCULAR)	2	2	3	1	0.0	7e−11	7e−5	n/a
YB(F,CIRCULAR)	3	1	5	10	0.1	1e−11	1e−6	n/a
ML(V,FIXED)	5	2	145	57	1.2	n/a	n/a	2e−16
SOR	1,000*	n/a	6,977	2,761	0.5	2e−3	7e−4	4e−5

Table 6.10: $K = 7$, $b = (8,5,8,6,8,7)$, $\text{Hypo}^{(1)}(3.0, 0.1)$, $\text{Erlang}^{(2)}(25.0, 5)$, $\text{Hypo}^{(3)}(70.0, 60.0)$, $\text{Hyper}^{(4)}(0.7, 0.7, (0.25, 0.75))$, $\text{Erlang}^{(5)}(0.1, 5)$, $\text{Hyper}^{(6)}(0.001, 0.001, (0.9, 0.1))$

	oIter	iIter	T	MF	MB	RE(ρ)	RE(E[X])	Res
CA	n/a	n/a	0	0	0.0	1e−9	3e−4	n/a
MVABLO	n/a	n/a	0	0	0.0	9e−10	3e−4	n/a
M(V,FIXED)	2	3	5	1	0.0	2e−12	4e−5	n/a
YB(F,CIRCULAR)	3	1	8	71	0.2	4e−13	9e−7	n/a
ML(V,FIXED)	5	2	297	349	2.4	n/a	n/a	2e−16
SOR	1,000*	n/a	14,743	6,159	1.1	8e−4	3e−4	2e−5

Table 6.11: $K = 8$, $b = (8,5,8,6,8,7)$, $\text{Hypo}^{(1)}(3.0, 0.1)$, $\text{Erlang}^{(2)}(25.0, 5)$, $\text{Hypo}^{(3)}(70.0, 60.0)$, $\text{Hyper}^{(4)}(0.7, 0.7, (0.25, 0.75))$, $\text{Erlang}^{(5)}(0.1, 5)$, $\text{Hyper}^{(6)}(0.001, 0.001, (0.9, 0.1))$

	oIter	iIter	T	MF	MB	RE(ρ)	RE(E[X])	Res
CA	n/a	n/a	0	0	0.0	5e−1	1e−1	n/a
MVABLO	n/a	n/a	0	0	0.0	5e−1	1e−1	n/a
M(F,CIRCULAR)	2	3	11	3	0.1	2e−4	5e−6	n/a
YB(F,CIRCULAR)	3	2	16	151	0.2	2e−3	6e−3	n/a
ML(V,FIXED)	6	2	658	770	4.6	n/a	n/a	1e−16
SOR	1,000*	n/a	28,334	12,328	2.1	5e−5	1e−5	4e−6

Tables 6.12 through 6.19 present the results of experiments which are conducted with 5, 6, 7, 8 customers and therefore fixed number of subnetworks in the M method. Buffer sizes of queues are given by $b = (8, 4, 8, 4, 8, 4)$. Thus the subnetworks resulting from the decomposition of the QN in method M is {{1,2},{3,4},{5,6}} for all numbers of customers. In this way, we investigate the behavior of M under the same decomposition into subnetworks for increasing number of customers. With this setting, for 5, 6, 7, 8 customers the closed QN has state space sizes of 8,061, 19,805, 42,417, 81,201 and HLM number of states of 249, 444, 729, 1,119, respectively. The number of nonzero elements needed for the sparse representation of Q for 5, 6, 7, 8 customers, are 41,573, 111,223, 254,311, and 512,347, respectively.

The results of the balanced case can be seen in Tables 6.12, 6.13, 6.14, and 6.15. In this case, M and YB methods present at most 2 digits of accuracy for both performance measures. Although CA and MVABLO sustain 1 digit of accuracy for utilization values, MVABLO has almost 1.5 digit accurate results for mean queue length values. As the number of customers increases, the M method performs less flops than the YB method, which implies that M becomes more efficient than YB. Indeed, for this case of the problem, M can be chosen for approximating utilizations and MVABLO can be chosen for approximating mean queue lengths. It is also important to notice that the ML method converges within 16 iterations and thus performs much better than SOR while calculating steady–state results of queues. Indeed, SOR does not converge within 1,000 iterations for this case of the problem. ML does not take more outer iterations when compared to the previous balanced case where the subnetworks changed for increasing number of customers. Yet, M yields higher inner iteration counts in this case. This implies that the problem is not difficult to solve than the previous case, but the subnetworks resulting from the decomposition procedure of M generates more difficult problems for the ML method for increasing number of customers.

Table 6.12: $K = 5$, $b = (8, 4, 8, 4, 8, 4)$, Hypo$^{(1)}(3.0, 1.5)$, Erlang$^{(2)}(5.0, 5)$, Hypo$^{(3)}(1.2, 4.0)$, Hyper$^{(4)}(1.0, 1.0, (0.25, 0.75))$, Erlang$^{(5)}(5.0, 5)$, Hyper$^{(6)}(0.8, 0.8, (0.9, 0.1))$

	oIter	iIter	T	MF	MB	RE(ρ)	RE(E[X])	Res
CA	n/a	n/a	0	0	0.0	1e−1	1e−1	n/a
MVABLO	n/a	n/a	0	0	0.0	9e−2	6e−2	n/a
M(F,CIRCULAR)	5	6	27	6	0.0	2e−2	3e−2	n/a
YB(F,CIRCULAR)	5	4	7	4	0.1	4e−2	3e−2	n/a
ML(V,FIXED)	15	2	189	65	0.5	n/a	n/a	1e−16
SOR	1,000*	n/a	2,921	1,074	0.2	5e−4	7e−4	1e−3

Table 6.13: $K = 6$, $b = (8, 4, 8, 4, 8, 4)$, Hypo$^{(1)}(3.0, 1.5)$, Erlang$^{(2)}(5.0, 5)$, Hypo$^{(3)}(1.2, 4.0)$, Hyper$^{(4)}(1.0, 1.0, (0.25, 0.75))$, Erlang$^{(5)}(5.0, 5)$, Hyper$^{(6)}(0.8, 0.8, (0.9, 0.1))$

	oIter	iIter	T	MF	MB	RE(ρ)	RE(E[X])	Res
CA	n/a	n/a	0	0	0.0	2e−1	2e−1	n/a
MVABLO	n/a	n/a	0	0	0.0	1e−1	6e−2	n/a
M(F,CIRCULAR)	5	7	41	9	0.0	2e−2	3e−2	n/a
YB(F,CIRCULAR)	5	5	11	18	0.1	3e−2	3e−2	n/a
ML(V,FIXED)	16	2	450	193	1.2	n/a	n/a	1e−15
SOR	1,000*	n/a	6,902	2,741	0.5	2e−4	1e−4	3e−4

Table 6.14: $K = 7$, $b = (8, 4, 8, 4, 8, 4)$, Hypo$^{(1)}(3.0, 1.5)$, Erlang$^{(2)}(5.0, 5)$, Hypo$^{(3)}(1.2, 4.0)$, Hyper$^{(4)}(1.0, 1.0, (0.25, 0.75))$, Erlang$^{(5)}(5.0, 5)$, Hyper$^{(6)}(0.8, 0.8, (0.9, 0.1))$

	oIter	iIter	T	MF	MB	RE(ρ)	RE(E[X])	Res
CA	n/a	n/a	0	0	0.0	2e−1	2e−1	n/a
MVABLO	n/a	n/a	0	0	0.0	1e−1	5e−2	n/a
M(F,CIRCULAR)	5	9	58	14	0.0	2e−2	3e−2	n/a
YB(F,CIRCULAR)	6	5	20	108	0.2	3e−2	4e−2	n/a
ML(F,CIRCULAR)	11	2	1,233	528	2.6	n/a	n/a	4e−16
SOR	1,000*	n/a	14,325	6,028	1.1	6e−5	7e−5	2e−4

Table 6.15: $K = 8$, $b = (8, 4, 8, 4, 8, 4)$, Hypo$^{(1)}(3.0, 1.5)$, Erlang$^{(2)}(5.0, 5)$, Hypo$^{(3)}(1.2, 4.0)$, Hyper$^{(4)}(1.0, 1.0, (0.25, 0.75))$, Erlang$^{(5)}(5.0, 5)$, Hyper$^{(6)}(0.8, 0.8, (0.9, 0.1))$

	oIter	iIter	T	MF	MB	RE(ρ)	RE(E[X])	Res
CA	n/a	n/a	0	0	0.0	2e−1	3e−1	n/a
MVABLO	n/a	n/a	0	0	0.0	1e−1	5e−2	n/a
M(F,CIRCULAR)	5	10	77	18	0.1	2e−2	4e−2	n/a
YB(F,CIRCULAR)	6	5	31	485	0.3	3e−2	6e−2	n/a
ML(F,CIRCULAR)	12	2	2,133	1,014	4.8	n/a	n/a	4e−16
SOR	1,000*	n/a	26,775	11,747	2.0	3e−5	3e−5	1e−4

The results of the unbalanced case can be seen in Tables 6.16, 6.17, 6.18, and 6.19. In the unbalanced case, the performance measures approximated by M appear to be at least 1 more digit accurate than those with YB. Also, the results obtained with M and YB appear to be 1.5 digits more accurate than those obtained with CA and MVABLO. For this reason, M, having performed less flops than YB, becomes the more accurate and efficient approximative method between the two. ML performs at most 6 outer iterations until convergence and significantly beats SOR. These experiments also show that there are cases in which one can use the M and YB methods.

Table 6.16: $K = 5$, $b = (8,4,8,4,8,4)$, Hypo$^{(1)}(3.0, 0.1)$, Erlang$^{(2)}(25.0, 5)$, Hypo$^{(3)}(70.0, 60.0)$, Hyper$^{(4)}(0.7, 0.7, (0.25, 0.75))$, Erlang$^{(5)}(0.1, 5)$, Hyper$^{(6)}(0.001, 0.001, (0.9, 0.1))$

	oIter	iIter	T	MF	MB	RE(ρ)	RE(E[X])	Res
CA	n/a	n/a	0	0	0.0	5e−1	2e−1	n/a
MVABLO	n/a	n/a	0	0	0.0	5e−1	2e−1	n/a
M(F,CIRCULAR)	2	3	5	1	0.0	2e−4	8e−6	n/a
YB(F,CIRCULAR)	3	2	3	2	0.1	2e−3	1e−2	n/a
ML(V,FIXED)	6	2	77	26	0.5	n/a	n/a	0e+0
SOR	1,000*	n/a	2,922	1,074	0.2	5e−4	2e−4	4e−5

Table 6.17: $K = 6$, $b = (8,4,8,4,8,4)$, Hypo$^{(1)}(3.0, 0.1)$, Erlang$^{(2)}(25.0, 5)$, Hypo$^{(3)}(70.0, 60.0)$, Hyper$^{(4)}(0.7, 0.7, (0.25, 0.75))$, Erlang$^{(5)}(0.1, 5)$, Hyper$^{(6)}(0.001, 0.001, (0.9, 0.1))$

	oIter	iIter	T	MF	MB	RE(ρ)	RE(E[X])	Res
CA	n/a	n/a	0	0	0.0	5e−1	3e−1	n/a
MVABLO	n/a	n/a	0	0	0.0	5e−1	3e−1	n/a
M(F,CIRCULAR)	2	3	6	1	0.0	8e−7	1e−7	n/a
YB(F,CIRCULAR)	3	2	4	11	0.1	3e−6	8e−3	n/a
ML(V,FIXED)	6	2	171	72	1.2	n/a	n/a	0e+0
SOR	1,000*	n/a	6,901	2,742	0.5	3e−5	7e−5	2e−5

Table 6.18: $K = 7$, $b = (8,4,8,4,8,4)$, Hypo$^{(1)}(3.0, 0.1)$, Erlang$^{(2)}(25.0, 5)$, Hypo$^{(3)}(70.0, 60.0)$, Hyper$^{(4)}(0.7, 0.7, (0.25, 0.75))$, Erlang$^{(5)}(0.1, 5)$, Hyper$^{(6)}(0.001, 0.001, (0.9, 0.1))$

	oIter	iIter	T	MF	MB	RE(ρ)	RE(E[X])	Res
CA	n/a	n/a	0	0	0.0	5e−1	4e−1	n/a
MVABLO	n/a	n/a	0	0	0.0	5e−1	4e−1	n/a
M(F,CIRCULAR)	2	3	7	2	0.0	1e−9	7e−8	n/a
YB(F,CIRCULAR)	3	2	7	54	0.2	2e−7	7e−3	n/a
ML(F,CIRCULAR)	5	2	535	801	2.6	n/a	n/a	1e−15
SOR	1,000*	n/a	14,342	6,028	1.1	5e−5	3e−5	9e−6

Table 6.19: $K = 8$, $b = (8,4,8,4,8,4)$, $\text{Hypo}^{(1)}(3.0, 0.1)$, $\text{Erlang}^{(2)}(25.0, 5)$, $\text{Hypo}^{(3)}(70.0, 60.0)$, $\text{Hyper}^{(4)}(0.7, 0.7, (0.25, 0.75))$, $\text{Erlang}^{(5)}(0.1, 5)$, $\text{Hyper}^{(6)}(0.001, 0.001, (0.9, 0.1))$

	oIter	iIter	T	MF	MB	RE(ρ)	RE(E[X])	Res
CA	n/a	n/a	0	0	0.0	5e−1	5e−1	n/a
MVABLO	n/a	n/a	0	0	0.0	5e−1	5e−1	n/a
M(F,CIRCULAR)	2	3	9	2	0.1	2e−10	5e−8	n/a
YB(F,CIRCULAR)	3	2	12	246	0.3	2e−7	6e−3	n/a
ML(V,FIXED)	6	2	609	661	4.3	n/a	n/a	0e+0
SOR	1,000*	n/a	26,763	11,747	2.0	3e−5	1e−5	7e−6

6.3 Problem 3

The third problem is a central server type closed QN which consists of 6 queues. The topology of the network is depicted in Figure 6.3. Service distributions of queues are represented by two hyperexponential distributions each with 2 phases, two hypoexponential distributions each with 2 phases and two Erlang distributions each with 5 phases. The buffer sizes of queues are given by $b = (8, 5, 7, 8, 8, 6)$. Therefore, for 5, 6, 7, 8 customers, we obtain cases of the problem with state space sizes of 8,070, 19,938, 43,320, 85,102, and HLM number of states of 252, 461, 785, 1,259 for which the numbers of nonzero elements needed for the sparse representation of Q are 50,682, 136,838, 317,787, 657,482, respectively. For 5, 6, 7, 8 customers the M method decomposes the QN into subnetworks given by the sets {{1},{2},{3},{4},{5},{6}}, {{1,2},{3},{4},{5},{6}}, {{1,2,6},{3},{4},{5}} and {{1,2,3,6},{4},{5}}, respectively. Tables 6.20 through 6.27 present the results for this network.

The results of the balanced case for 5, 6, 7, 8 customers are presented in Tables 6.20, 6.21, 6.22, and 6.23, respectively. Therein, we see that CA and MVABLO produce at most 1.5 digit accurate results for utilization values and mean queue length values. On the other hand, M and YB methods yield 2 to 3.5 digits accurate results for utilization values and almost 2 digits accurate results for mean queue length values for all numbers of customer. For 7 and 8 customers, YB performs less flops than that of M, but YB is unable to approximate the performance measures more accurately than M. Also, the inner iteration counts of M increases faster than the inner iteration counts of YB for increasing number

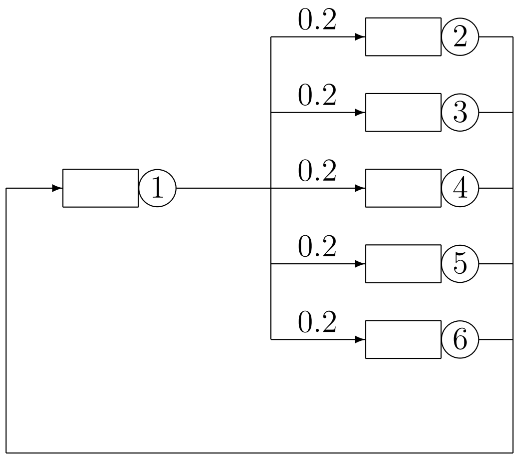

Figure 6.3: Problem 3.

of customers. This is mainly because of the fact that decomposition procedure of M generates inconvenient partitions of queues for increasing number of customers. Indeed, this case shows how the decomposition procedure can adversely affect the flop counts of M for increasing number of customers. When we compare SOR and ML, we see that the flop counts of ML are less than the flop counts of SOR for all numbers of customers.

Table 6.20: $K = 5$, $b = (8, 5, 7, 8, 8, 6)$, $\text{Hypo}^{(1)}(3.0, 1.5)$, $\text{Erlang}^{(2)}(1.0, 5)$, $\text{Hypo}^{(3)}(0.2, 2.0)$, $\text{Hyper}^{(4)}(0.2, 0.2, (0.25, 0.75))$, $\text{Erlang}^{(5)}(1.0, 5)$, $\text{Hyper}^{(6)}(0.2, 0.2, (0.9, 0.1))$

	oIter	iIter	T	MF	MB	RE(ρ)	RE(E[X])	Res
CA	n/a	n/a	0	0	0.0	5e−2	6e−2	n/a
MVABLO	n/a	n/a	0	0	0.0	5e−2	6e−2	n/a
M(F,CIRCULAR)	5	4	6	1	0.0	2e−3	1e−2	n/a
YB(F,CIRCULAR)	5	4	8	14	0.1	2e−3	1e−2	n/a
ML(V,FIXED)	13	2	216	95	0.6	n/a	n/a	2e−16
SOR	169	n/a	743	265	0.2	n/a	n/a	9e−16

Table 6.21: $K = 6$, $b = (8, 5, 7, 8, 8, 6)$, Hypo$^{(1)}(3.0, 1.5)$, Erlang$^{(2)}(1.0, 5)$, Hypo$^{(3)}(0.2, 2.0)$, Hyper$^{(4)}(0.2, 0.2, (0.25, 0.75))$, Erlang$^{(5)}(1.0, 5)$, Hyper$^{(6)}(0.2, 0.2, (0.9, 0.1))$

	oIter	iIter	T	MF	MB	RE(ρ)	RE(E[X])	Res
CA	n/a	n/a	0	0	0.0	6e−2	7e−2	n/a
MVABLO	n/a	n/a	0	0	0.0	5e−2	7e−2	n/a
M(F,CIRCULAR)	5	5	27	6	0.1	1e−3	8e−3	n/a
YB(F,CIRCULAR)	5	5	12	66	0.1	2e−3	1e−2	n/a
ML(V,FIXED)	14	2	489	359	1.3	n/a	n/a	5e−16
SOR	237	n/a	2,244	945	0.5	n/a	n/a	9e−16

Table 6.22: $K = 7$, $b = (8, 5, 7, 8, 8, 6)$, Hypo$^{(1)}(3.0, 1.5)$, Erlang$^{(2)}(1.0, 5)$, Hypo$^{(3)}(0.2, 2.0)$, Hyper$^{(4)}(0.2, 0.2, (0.25, 0.75))$, Erlang$^{(5)}(1.0, 5)$, Hyper$^{(6)}(0.2, 0.2, (0.9, 0.1))$

	oIter	iIter	T	MF	MB	RE(ρ)	RE(E[X])	Res
CA	n/a	n/a	0	0	0.0	8e−2	1e−1	n/a
MVABLO	n/a	n/a	0	0	0.0	5e−2	7e−2	n/a
M(V,FIXED)	5	9	177	55	0.1	9e−4	9e−3	n/a
YB(F,CIRCULAR)	5	5	20	27	0.2	1e−3	2e−2	n/a
ML(V,FIXED)	15	2	1,062	431	2.7	n/a	n/a	3e−16
SOR	315	n/a	6,289	2,783	1.1	n/a	n/a	1e−15

Table 6.23: $K = 8$, $b = (8, 5, 7, 8, 8, 6)$, Hypo$^{(1)}(3.0, 1.5)$, Erlang$^{(2)}(1.0, 5)$, Hypo$^{(3)}(0.2, 2.0)$, Hyper$^{(4)}(0.2, 0.2, (0.25, 0.75))$, Erlang$^{(5)}(1.0, 5)$, Hyper$^{(6)}(0.2, 0.2, (0.9, 0.1))$

	oIter	iIter	T	MF	MB	RE(ρ)	RE(E[X])	Res
CA	n/a	n/a	0	0	0.0	1e−1	2e−1	n/a
MVABLO	n/a	n/a	0	0	0.0	6e−2	8e−2	n/a
M(V,FIXED)	5	10	1,229	925	0.7	6e−4	9e−3	n/a
YB(V,FIXED)	5	8	32	52	0.3	7e−4	2e−2	n/a
ML(V,FIXED)	15	2	1,973	829	5.0	n/a	n/a	3e−16
SOR	401	n/a	15,351	7,052	2.1	n/a	n/a	1e−15

The results of the unbalanced case for 5, 6, 7, 8 customers are presented in Tables 6.24, 6.25, 6.26, and 6.27, respectively. For 5 and 6 customers, neither M nor YB are able to produce more accurate results than CA and MVABLO. For 7 and 8 customers, M and YB yield at least 2 digits more accurate results than CA and MVABLO. Having performed less flops than YB, M is the more efficient method between the two methods. For increasing number of customers, SOR and ML exhibit the same behavior as in the balanced case.

Table 6.24: $K = 5$, $b =$ (8,5,7,8,8,6), Hypo$^{(1)}(3.0, 0.1)$, Erlang$^{(2)}(25.0, 5)$, Hypo$^{(3)}(70.0, 60.0)$, Hyper$^{(4)}(0.7, 0.7, (0.25, 0.75))$, Erlang$^{(5)}(0.1, 5)$, Hyper$^{(6)}(0.001, 0.001, (0.9, 0.1))$

	oIter	iIter	T	MF	MB	RE(ρ)	RE(E[X])	Res
CA	n/a	n/a	0	0	0.0	6e−7	4e−4	n/a
MVABLO	n/a	n/a	0	0	0.0	6e−7	4e−4	n/a
M(F,CIRCULAR)	3	2	2	0	0.0	3e−7	2e−4	n/a
YB(F,CIRCULAR)	3	2	3	7	0.1	3e−7	2e−4	n/a
ML(V,FIXED)	10	2	156	65	0.6	n/a	n/a	4e−16
SOR	198	n/a	849	309	0.2	n/a	n/a	9e−16

Table 6.25: $K = 6$, $b =$ (8,5,7,8,8,6), Hypo$^{(1)}(3.0, 0.1)$, Erlang$^{(2)}(25.0, 5)$, Hypo$^{(3)}(70.0, 60.0)$, Hyper$^{(4)}(0.7, 0.7, (0.25, 0.75))$, Erlang$^{(5)}(0.1, 5)$, Hyper$^{(6)}(0.001, 0.001, (0.9, 0.1))$

	oIter	iIter	T	MF	MB	RE(ρ)	RE(E[X])	Res
CA	n/a	n/a	0	0	0.0	4e−8	3e−4	n/a
MVABLO	n/a	n/a	0	0	0.0	4e−8	3e−4	n/a
M(F,CIRCULAR)	2	2	4	1	0.1	2e−8	2e−4	n/a
YB(F,CIRCULAR)	3	2	5	35	0.1	2e−8	1e−4	n/a
ML(V,FIXED)	10	2	351	221	1.3	n/a	n/a	6e−16
SOR	213	n/a	2,018	849	0.5	n/a	n/a	9e−16

Table 6.26: $K = 7$ $b =$ (8,5,7,8,8,6), Hypo$^{(1)}(3.0, 0.1)$, Erlang$^{(2)}(25.0, 5)$, Hypo$^{(3)}(70.0, 60.0)$, Hyper$^{(4)}(0.7, 0.7, (0.25, 0.75))$, Erlang$^{(5)}(0.1, 5)$, Hyper$^{(6)}(0.001, 0.001, (0.9, 0.1))$

	oIter	iIter	T	MF	MB	RE(ρ)	RE(E[X])	Res
CA	n/a	n/a	0	0	0.0	5e−1	1e−1	n/a
MVABLO	n/a	n/a	0	0	0.0	7e−1	5e−1	n/a
M(V,FIXED)	3	4	54	17	0.1	1e−3	2e−3	n/a
YB(V,FIXED)	3	2	10	106	0.2	2e−3	2e−3	n/a
ML(V,FIXED)	84	2	5,926	4,542	2.7	n/a	n/a	9e−16
SOR	991	n/a	19,780	8,749	1.1	n/a	n/a	1e−15

Table 6.27: $K = 8$, $b = (8,5,7,8,8,6)$, $\text{Hypo}^{(1)}(3.0, 0.1)$, $\text{Erlang}^{(2)}(25.0, 5)$, $\text{Hypo}^{(3)}(70.0, 60.0)$, $\text{Hyper}^{(4)}(0.7, 0.7, (0.25, 0.75))$, $\text{Erlang}^{(5)}(0.1, 5)$, $\text{Hyper}^{(6)}(0.001, 0.001, (0.9, 0.1))$

	oIter	iIter	T	MF	MB	RE(ρ)	RE(E[X])	Res
CA	n/a	n/a	0	0	0.0	6e−1	3e−1	n/a
MVABLO	n/a	n/a	0	0	0.0	6e−1	6e−1	n/a
M(V,FIXED)	4	5	503	345	0.7	4e−4	1e−3	n/a
YB(F,CIRCULAR)	5	2	34	742	0.3	2e−3	2e−3	n/a
ML(V,FIXED)	116	2	15,203	11,040	5.0	n/a	n/a	9e−16
SOR	1,000*	n/a	38,272	17,580	2.1	3e−15	6e−15	6e−13

Another set of experiments for problem 3 is conducted by taking $b = (8, 4, 4, 8, 8, 4)$. In this way, we obtain fixed subnetwork topologies with Marie's method for $K \in \{5, 6, 7, 8\}$. The subnetworks resulting from the decomposition of the QN for method M is given by {{1,2,3,6},{4},{5}} for all numbers of customers. For 5, 6, 7, 8 customers the closed QN has state space sizes of 8,061, 19,805, 42,417, 81,201 and has HLM number of states of 249, 444, 729, 1,119, respectively. The number of nonzero elements needed for the sparse representation of Q for increasing number of customers are 50,653, 136,242, 312,747, and 632,206, respectively. Tables 6.28 through 6.35 show the results of this new configuration.

The results of the balanced case of this configuration for 5, 6, 7, 8 customers can be seen in Tables 6.28, 6.29, 6.30, and 6.31, respectively. Utilization results of queues in this case are at least 1.5 digits more accurate for M and YB than those for CA and MVABLO. Mean queue length results obtained with M and YB are at least 0.5 digit more accurate than the results obtained with CA and MVABLO. Again, we see the effect of an inconvenient partitioning introduced by the decomposition procedure of M. Because of the fact that inner iteration counts of M are almost twice those of YB for all number of customers, YB requires less flops than M and is more efficient while approximating utilizations values for $K \in \{5, 6\}$. On the other hand, the number of flops performed to obtain the solution of the state dependent exponential network at each iteration of YB increases significantly for $K = 8$. In that case, YB requires more flops than M. SOR and ML continue to exhibit the same behavior as in the previous configuration of the problem.

Table 6.28: $K = 5$, $b = (8, 4, 4, 8, 8, 4)$, Hypo$^{(1)}(3.0, 1.5)$, Erlang$^{(2)}(1.0, 5)$, Hypo$^{(3)}(0.2, 2.0)$, Hyper$^{(4)}(0.2, 0.2, (0.25, 0.75))$, Erlang$^{(5)}(1.0, 5)$, Hyper$^{(6)}(0.2, 0.2, (0.9, 0.1))$

	oIter	iIter	T	MF	MB	RE(ρ)	RE(E[X])	Res
CA	n/a	n/a	0	0	0.0	1e−1	1e−1	n/a
MVABLO	n/a	n/a	0	0	0.0	6e−2	5e−2	n/a
M(V,FIXED)	4	8	153	51	0.1	6e−4	6e−3	n/a
YB(F,CIRCULAR)	5	4	7	13	0.1	8e−4	1e−2	n/a
ML(V,FIXED)	12	2	187	87	0.6	n/a	n/a	7e−16
SOR	164	n/a	652	257	0.2	n/a	n/a	9e−16

Table 6.29: $K = 6$, $b = (8, 4, 4, 8, 8, 4)$, Hypo$^{(1)}(3.0, 1.5)$, Erlang$^{(2)}(1.0, 5)$, Hypo$^{(3)}(0.2, 2.0)$, Hyper$^{(4)}(0.2, 0.2, (0.25, 0.75))$, Erlang$^{(5)}(1.0, 5)$, Hyper$^{(6)}(0.2, 0.2, (0.9, 0.1))$

	oIter	iIter	T	MF	MB	RE(ρ)	RE(E[X])	Res
CA	n/a	n/a	0	0	0.0	2e−1	2e−1	n/a
MVABLO	n/a	n/a	0	0	0.0	7e−2	5e−2	n/a
M(V,FIXED)	4	9	307	118	0.2	5e−4	7e−3	n/a
YB(F,CIRCULAR)	5	5	11	56	0.1	1e−3	1e−2	n/a
ML(V,FIXED)	13	2	450	288	1.3	n/a	n/a	7e−16
SOR	206	n/a	1,932	817	0.5	n/a	n/a	1e−15

Table 6.30: $K = 7$, $b = (8, 4, 4, 8, 8, 4)$, Hypo$^{(1)}(3.0, 1.5)$, Erlang$^{(2)}(1.0, 5)$, Hypo$^{(3)}(0.2, 2.0)$, Hyper$^{(4)}(0.2, 0.2, (0.25, 0.75))$, Erlang$^{(5)}(1.0, 5)$, Hyper$^{(6)}(0.2, 0.2, (0.9, 0.1))$

	oIter	iIter	T	MF	MB	RE(ρ)	RE(E[X])	Res
CA	n/a	n/a	0	0	0.0	2e−1	2e−1	n/a
MVABLO	n/a	n/a	0	0	0.0	8e−2	5e−2	n/a
M(V,FIXED)	5	9	619	271	0.4	4e−4	8e−3	n/a
YB(F,CIRCULAR)	5	5	18	284	0.2	2e−3	2e−2	n/a
ML(F,CIRCULAR)	8	2	1,087	458	2.8	n/a	n/a	5e−16
SOR	236	n/a	5,111	2,277	1.1	n/a	n/a	9e−16

Table 6.31: $K = 8$, $b = (8, 4, 4, 8, 8, 4)$, Hypo$^{(1)}(3.0, 1.5)$, Erlang$^{(2)}(1.0, 5)$, Hypo$^{(3)}(0.2, 2.0)$, Hyper$^{(4)}(0.2, 0.2, (0.25, 0.75))$, Erlang$^{(5)}(1.0, 5)$, Hyper$^{(6)}(0.2, 0.2, (0.9, 0.1))$

	oIter	iIter	T	MF	MB	RE(ρ)	RE(E[X])	Res
CA	n/a	n/a	0	0	0.0	2e−1	3e−1	n/a
MVABLO	n/a	n/a	0	0	0.0	8e−2	7e−2	n/a
M(V,FIXED)	5	10	946	528	0.6	5e−4	1e−2	n/a
YB(F,CIRCULAR)	6	5	35	1,270	0.3	3e−3	2e−2	n/a
ML(F,CIRCULAR)	9	2	2,117	902	5.2	n/a	n/a	1e−16
SOR	339	n/a	12,262	5,689	2.0	n/a	n/a	1e−15

The results of the unbalanced case can be seen in Tables 6.32, 6.33, 6.34, and 6.35. In this case, M and YB yield at least 2 digits more accurate results than CA and MVABLO for both performance measures. CA and MVABLO provide very inaccurate results and they are hardly able to obtain 1 digit of accuracy for both performance measures. Although YB performs less flops than M for $K \in \{5, 6\}$, as the number of customers increases M approximates utilization values 1 digit more accurately than YB and performs less flops than YB; in this case, M and YB presents the same behavior as in the balanced case. Thus, this problem reveals cases in which YB performs better than M. Regarding SOR, it fails to converge to the predefined tolerance within 1,000 iterations for $K \in \{6, 7, 8\}$. Hence, M is to be recommended especially for increasing values of K in this problem.

Table 6.32: $K = 5$, $b = (8,4,4,8,8,4)$, $\text{Hypo}^{(1)}(3.0, 0.1)$, $\text{Erlang}^{(2)}(25.0, 5)$, $\text{Hypo}^{(3)}(70.0, 60.0)$, $\text{Hyper}^{(4)}(0.7, 0.7, (0.25, 0.75))$, $\text{Erlang}^{(5)}(0.1, 5)$, $\text{Hyper}^{(6)}(0.001, 0.001, (0.9, 0.1))$

	oIter	iIter	T	MF	MB	RE(ρ)	RE(E[X])	Res
CA	n/a	n/a	0	0	0.0	5e−1	2e−1	n/a
MVABLO	n/a	n/a	0	0	0.0	7e−1	6e−1	n/a
M(V,FIXED)	2	5	57	18	0.1	1e−3	2e−3	n/a
YB(F,CIRCULAR)	3	2	3	7	0.1	2e−3	3e−3	n/a
ML(V,FIXED)	84	2	1,292	605	0.6	n/a	n/a	8e−16
SOR	976	n/a	3,878	1,524	0.2	n/a	n/a	1e−15

Table 6.33: $K = 6$, $b = (8,4,4,8,8,4)$, $\text{Hypo}^{(1)}(3.0, 0.1)$, $\text{Erlang}^{(2)}(25.0, 5)$, $\text{Hypo}^{(3)}(70.0, 60.0)$, $\text{Hyper}^{(4)}(0.7, 0.7, (0.25, 0.75))$, $\text{Erlang}^{(5)}(0.1, 5)$, $\text{Hyper}^{(6)}(0.001, 0.001, (0.9, 0.1))$

	oIter	iIter	T	MF	MB	RE(ρ)	RE(E[X])	Res
CA	n/a	n/a	0	0	0.0	6e−1	3e−1	n/a
MVABLO	n/a	n/a	0	0	0.0	6e−1	7e−1	n/a
M(V,FIXED)	4	5	174	67	0.2	4e−4	2e−3	n/a
YB(F,CIRCULAR)	5	2	8	66	0.1	2e−3	3e−3	n/a
ML(V,FIXED)	116	2	3,977	2,856	1.3	n/a	n/a	9e−16
SOR	1,000*	n/a	9,371	3,959	0.5	2e−15	4e−15	5e−13

Table 6.34: $K = 7$, $b = (8,4,4,8,8,4)$, $\text{Hypo}^{(1)}(3.0, 0.1)$, $\text{Erlang}^{(2)}(25.0, 5)$, $\text{Hypo}^{(3)}(70.0, 60.0)$, $\text{Hyper}^{(4)}(0.7, 0.7, (0.25, 0.75))$, $\text{Erlang}^{(5)}(0.1, 5)$, $\text{Hyper}^{(6)}(0.001, 0.001, (0.9, 0.1))$

	oIter	iIter	T	MF	MB	RE(ρ)	RE(E[X])	Res
CA	n/a	n/a	0	0	0.0	6e−1	4e−1	n/a
MVABLO	n/a	n/a	0	0	0.0	6e−1	7e−1	n/a
M(V,FIXED)	5	6	374	176	0.4	9e−5	2e−3	n/a
YB(F,CIRCULAR)	5	2	15	244	0.2	2e−3	4e−3	n/a
ML(F,CIRCULAR)	96	2	13,108	9,514	2.8	n/a	n/a	8e−16
SOR	1,000*	n/a	19,420	8,652	1.1	1e−15	1e−15	3e−13

Table 6.35: $K = 8$, $b = (8,4,4,8,8,4)$, $\text{Hypo}^{(1)}(3.0, 0.1)$, $\text{Erlang}^{(2)}(25.0, 5)$, $\text{Hypo}^{(3)}(70.0, 60.0)$, $\text{Hyper}^{(4)}(0.7, 0.7, (0.25, 0.75))$, $\text{Erlang}^{(5)}(0.1, 5)$, $\text{Hyper}^{(6)}(0.001, 0.001, (0.9, 0.1))$

	oIter	iIter	T	MF	MB	RE(ρ)	RE(E[X])	Res
CA	n/a	n/a	0	0	0.0	6e−1	5e−1	n/a
MVABLO	n/a	n/a	0	0	0.0	6e−1	7e−1	n/a
M(V,FIXED)	5	7	584	358	0.6	3e−4	1e−3	n/a
YB(F,CIRCULAR)	6	3	31	1,079	0.3	2e−3	5e−3	n/a
ML(F,CIRCULAR)	101	2	24,095	17,622	4.7	n/a	n/a	8e−16
SOR	1,000*	n/a	36,169	16,773	2.0	1e−15	1e−15	5e−12

6.4 Problem 4

The fourth problem consists of a closed QN with 6 queues which have arbitrary routing probabilities among them and have servers with two hyperexponential service distributions each with 2 phases, two hypoexponential service distributions each with 2 phases and two Erlang service distributions each with 5 phases. This problem is considered in order to investigate the behavior of methods under arbitrary routing. Buffer sizes of queues are defined by $b = (8, 5, 7, 8, 8, 6)$. In this case, the network has state space sizes of 8,070, 19,938, 43,320, 85,102, and HLM number of states of 252, 461, 785, 1,259 for $K \in \{5, 6, 7, 8\}$, respectively. The number of nonzero elements needed for the sparse representation of Q for $K \in \{5, 6, 7, 8\}$ is 57,489, 155,121, 360,054, and 744,515, respectively. The topology of the network is depicted in Figure 6.4 and Tables 6.36 through 6.43 give the results obtained with different methods while analyzing balanced and unbalanced cases of problem 4. Decomposition of the QN into subnetworks in the M method is given by the sets {{1},{2},{3},{4},{5},{6}}, {{1,2},{3},{4},{5},{6}}, {{1,2},{3},{4,5,6}}, {{1,2,3},{4,5,6}} for 5, 6, 7, 8 customers, respectively.

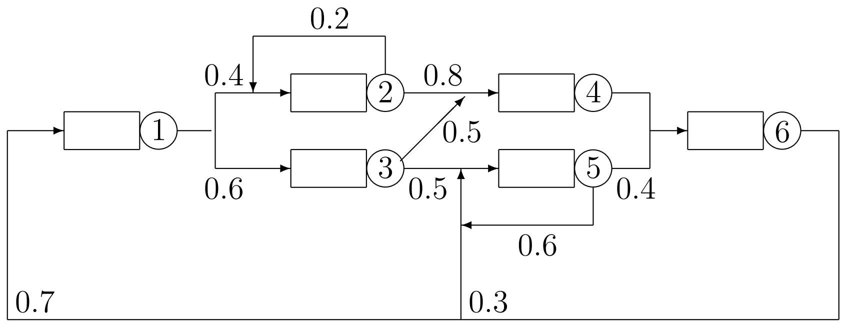

Figure 6.4: Problem 4.

Tables 6.36, 6.37, 6.38, and 6.39 present the results of the balanced network for 5, 6, 7, 8 customers, respectively. The results with YB are 2.5 digits accurate for utilization values. The utilization results with YB are 0.5 digits more accurate than the results with M for $K \in \{5, 6, 7\}$ and at least 1 digit more accurate than the results of MVABLO and CA. YB performs less flops than M for $K \in \{7, 8\}$. Also YB does almost two times less inner iterations than M for $K = 8$. Therefore YB is the more accurate method when approximating utilization values and the more efficient one for $K \in \{7, 8\}$ when compared with M. Mean queue length values obtained by MVABLO have at least 1.5 digits of accuracy for all numbers of customers. Indeed, M and YB are unable to approximate mean queue length values more than 0.5 digits better than MVABLO. This implies that by performing a negligible number of flops when compared with M and YB, MVABLO emerges as a reasonable method to approximate mean queue length values. When compared with SOR, ML performs less flops than SOR for all numbers of customers. It is important to note that ML has almost the same outer iteration counts for 5, 6, 7, and 8 customers. Thus the outer iteration counts of ML do not depend on the number of customers, whereas the iteration counts of SOR increases with increasing number of customers in this case.

Table 6.36: $K = 5$, $b = (8, 5, 7, 8, 8, 6)$, Hypo$^{(1)}(3.0, 1.5)$, Erlang$^{(2)}(2.5, 5)$, Hypo$^{(3)}(0.8, 2)$, Hyper$^{(4)}(0.7, 0.7, (0.25, 0.75))$, Erlang$^{(5)}(9.0, 5)$, Hyper$^{(6)}(1.4, 1.4, (0.9, 0.1))$

	oIter	iIter	T	MF	MB	RE(ρ)	RE(E[X])	Res
CA	n/a	n/a	0	0	0.0	5e−2	3e−2	n/a
MVABLO	n/a	n/a	0	0	0.0	5e−2	3e−2	n/a
M(F,CIRCULAR)	5	4	7	1	0.0	1e−2	9e−3	n/a
YB(F,CIRCULAR)	5	4	8	9	0.1	3e−3	3e−2	n/a
ML(V,FIXED)	16	2	357	104	0.5	n/a	n/a	5e−16
SOR	296	n/a	1,544	441	0.2	n/a	n/a	9e−16

Table 6.37: $K = 6$, $b = (8, 5, 7, 8, 8, 6)$, Hypo$^{(1)}(3.0, 1.5)$, Erlang$^{(2)}(2.5, 5)$, Hypo$^{(3)}(0.8, 2)$, Hyper$^{(4)}(0.7, 0.7, (0.25, 0.75))$, Erlang$^{(5)}(9.0, 5)$, Hyper$^{(6)}(1.4, 1.4, (0.9, 0.1))$

	oIter	iIter	T	MF	MB	RE(ρ)	RE(E[X])	Res
CA	n/a	n/a	0	0	0.0	6e−2	4e−2	n/a
MVABLO	n/a	n/a	0	0	0.0	6e−2	4e−2	n/a
M(F,CIRCULAR)	5	5	24	5	0.1	1e−2	1e−2	n/a
YB(F,CIRCULAR)	5	5	13	46	0.1	3e−3	4e−2	n/a
ML(V,FIXED)	15	2	547	330	1.2	n/a	n/a	9e−16
SOR	397	n/a	3,569	1,517	0.5	n/a	n/a	1e−15

Table 6.38: $K = 7$, $b = (8, 5, 7, 8, 8, 6)$, Hypo$^{(1)}(3.0, 1.5)$, Erlang$^{(2)}(2.5, 5)$, Hypo$^{(3)}(0.8, 2)$, Hyper$^{(4)}(0.7, 0.7, (0.25, 0.75))$, Erlang$^{(5)}(9.0, 5)$, Hyper$^{(6)}(1.4, 1.4, (0.9, 0.1))$

	oIter	iIter	T	MF	MB	RE(ρ)	RE(E[X])	Res
CA	n/a	n/a	0	0	0.0	6e−2	9e−2	n/a
MVABLO	n/a	n/a	0	0	0.0	6e−2	4e−2	n/a
M(F,CIRCULAR)	5	8	167	56	0.2	1e−2	2e−2	n/a
YB(V,FIXED)	6	8	25	35	0.2	3e−3	4e−2	n/a
ML(V,FIXED)	16	2	1,178	491	2.5	n/a	n/a	4e−16
SOR	506	n/a	9,620	4,307	1.1	n/a	n/a	1e−15

Table 6.39: $K = 8$, $b = (8, 5, 7, 8, 8, 6)$, Hypo$^{(1)}(3.0, 1.5)$, Erlang$^{(2)}(2.5, 5)$, Hypo$^{(3)}(0.8, 2)$, Hyper$^{(4)}(0.7, 0.7, (0.25, 0.75))$, Erlang$^{(5)}(9.0, 5)$, Hyper$^{(6)}(1.4, 1.4, (0.9, 0.1))$

	oIter	iIter	T	MF	MB	RE(ρ)	RE(E[X])	Res
CA	n/a	n/a	0	0	0.0	6e−2	1e−1	n/a
MVABLO	n/a	n/a	0	0	0.0	6e−2	5e−2	n/a
M(V,FIXED)	4	19	489	158	0.2	6e−3	1e−2	n/a
YB(V,FIXED)	6	9	38	65	0.3	3e−3	5e−2	n/a
ML(V,FIXED)	17	2	2,321	1,006	4.8	n/a	n/a	3e−16
SOR	615	n/a	22,468	10,464	2.1	n/a	n/a	1e−15

Tables 6.40, 6.41, 6.42, and 6.43 present the results of the unbalanced network for 5, 6, 7, 8 customers, respectively. The M method approximates utilization values by at least 1.5 digits and mean queue length values by at least 0.5 digits more accurate than CA and MVABLO. CA and MVABLO yield high accuracy for the performance measures of 5 and 6 customers. Yet, they fail to preserve this high accuracy in their approximations for 7 and 8 customers, where the number of queues subject to blocking is higher. YB performs less inner iterations than M for $K \in \{7, 8\}$, but the outer iteration counts of YB is more than M and this adversely affects the flop counts of YB significantly. Having performed less than one ninth of the flops of the YB method, M is the more accurate and efficient method for this case of the problem between the two methods. For none of the numbers of customers, SOR is able to perform less flops than ML.

Table 6.40: $K = 5$, $b = (8,5,7,8,8,6)$, $\text{Hypo}^{(1)}(3.0, 0.1)$, $\text{Erlang}^{(2)}(25.0, 5)$, $\text{Hypo}^{(3)}(70.0, 60.0)$, $\text{Hyper}^{(4)}(0.7, 0.7, (0.25, 0.75))$, $\text{Erlang}^{(5)}(0.1, 5)$, $\text{Hyper}^{(6)}(0.001, 0.001, (0.9, 0.1))$

	oIter	iIter	T	MF	MB	RE(ρ)	RE(E[X])	Res
CA	n/a	n/a	0	0	0.0	6e−7	3e−4	n/a
MVABLO	n/a	n/a	0	0	0.0	6e−7	3e−4	n/a
M(F,CIRCULAR)	2	2	2	0	0.0	2e−8	7e−5	n/a
YB(F,CIRCULAR)	3	1	3	6	0.0	5e−7	4e−4	n/a
ML(V,FIXED)	6	2	99	43	0.5	n/a	n/a	1e−16
SOR	103	n/a	389	154	0.2	n/a	n/a	9e−16

Table 6.41: $K = 6$, $b = (8,5,7,8,8,6)$, $\text{Hypo}^{(1)}(3.0, 0.1)$, $\text{Erlang}^{(2)}(25.0, 5)$, $\text{Hypo}^{(3)}(70.0, 60.0)$, $\text{Hyper}^{(4)}(0.7, 0.7, (0.25, 0.75))$, $\text{Erlang}^{(5)}(0.1, 5)$, $\text{Hyper}^{(6)}(0.001, 0.001, (0.9, 0.1))$

	oIter	iIter	T	MF	MB	RE(ρ)	RE(E[X])	Res
CA	n/a	n/a	0	0	0.0	4e−8	2e−4	n/a
MVABLO	n/a	n/a	0	0	0.0	4e−8	2e−4	n/a
M(F,CIRCULAR)	2	2	4	1	0.1	9e−10	4e−5	n/a
YB(F,CIRCULAR)	3	2	5	25	0.1	3e−8	4e−4	n/a
ML(V,FIXED)	6	2	222	127	1.2	n/a	n/a	1e−16
SOR	110	n/a	990	421	0.5	n/a	n/a	9e−16

Table 6.42: $K = 7$, $b = (8,5,7,8,8,6)$, $\text{Hypo}^{(1)}(3.0, 0.1)$, $\text{Erlang}^{(2)}(25.0, 5)$, $\text{Hypo}^{(3)}(70.0, 60.0)$, $\text{Hyper}^{(4)}(0.7, 0.7, (0.25, 0.75))$, $\text{Erlang}^{(5)}(0.1, 5)$, $\text{Hyper}^{(6)}(0.001, 0.001, (0.9, 0.1))$

	oIter	iIter	T	MF	MB	RE(ρ)	RE(E[X])	Res
CA	n/a	n/a	0	0	0.0	5e−1	1e−1	n/a
MVABLO	n/a	n/a	0	0	0.0	5e−1	2e−1	n/a
M(V,FIXED)	2	4	36	11	0.1	8e−3	3e−3	n/a
YB(F,CIRCULAR)	3	2	9	120	0.2	2e−2	5e−3	n/a
ML(V,FIXED)	6	2	453	575	2.5	n/a	n/a	5e−16
SOR	473	n/a	8,987	4,026	1.1	n/a	n/a	1e−15

Table 6.43: $K = 8$, $b = (8,5,7,8,8,6)$, $\text{Hypo}^{(1)}(3.0, 0.1)$, $\text{Erlang}^{(2)}(25.0, 5)$, $\text{Hypo}^{(3)}(70.0, 60.0)$, $\text{Hyper}^{(4)}(0.7, 0.7, (0.25, 0.75))$, $\text{Erlang}^{(5)}(0.1, 5)$, $\text{Hyper}^{(6)}(0.001, 0.001, (0.9, 0.1))$

	oIter	iIter	T	MF	MB	RE(ρ)	RE(E[X])	Res
CA	n/a	n/a	0	0	0.0	5e−1	3e−1	n/a
MVABLO	n/a	n/a	0	0	0.0	5e−1	1e−1	n/a
M(F,CIRCULAR)	2	6	177	76	0.3	3e−4	2e−4	n/a
YB(F,CIRCULAR)	4	2	22	686	0.3	1e−3	2e−2	n/a
ML(V,FIXED)	44	2	8,404	6,659	4.8	n/a	n/a	5e−16
SOR	875	n/a	31,980	14,886	2.1	n/a	n/a	1e−15

The experiments for problem 4 are repeated by setting $b = (8, 4, 4, 8, 8, 4)$. In this way, Marie's method is forced to possess fixed subnetwork topologies and queues 2, 3, 6 are blocking for $K \in \{5, 6, 7, 8\}$. Thus, subnetworks resulting from the decomposition procedure of M are given by the set {{1,2,3},{4,5,6}} for all numbers of customers. With this setting for 5, 6, 7, 8 customers, the closed QN has state space sizes of 8,070, 19,938, 43,320, 85,102 and has HLM number of states of 252, 461, 785, 1,259, respectively. The nonzero elements needed for the sparse representation of Q for increasing number of customers are 57,489, 155,121, 360,054, and 744,515, respectively. Tables 6.44 through 6.51 show the results of these experiments.

Tables 6.44, 6.45, 6.46, and 6.47 present the results of the balanced network for 5, 6, 7, 8 customers, respectively. Results obtained with CA are at most 1 digit accurate for both performance measures. MVABLO approximates the performance measures at most 0.5 digits more accurate than CA. On the other hand, M and YB methods are able to achieve at least 1.5 digits of accuracy for both performance measures. Although M performs more inner iterations than

YB, YB performs more outer iterations than M. Flop counts of M do not increase as much as flop counts of YB for increasing number of customers. Hence, this makes M more efficient than YB. When we look at SOR and ML, we see that ML performs less flops than SOR method and is more efficient for all numbers of customers.

Table 6.44: $K = 5$, $b = (8, 4, 4, 8, 8, 4)$, $\text{Hypo}^{(1)}(3.0, 1.5)$, $\text{Erlang}^{(2)}(2.5, 5)$, $\text{Hypo}^{(3)}(0.8, 2)$, $\text{Hyper}^{(4)}(0.7, 0.7, (0.25, 0.75))$, $\text{Erlang}^{(5)}(9.0, 5)$, $\text{Hyper}^{(6)}(1.4, 1.4, (0.9, 0.1))$

	oIter	iIter	T	MF	MB	RE(ρ)	RE(E[X])	Res
CA	n/a	n/a	0	0	0.0	1e−1	1e−1	n/a
MVABLO	n/a	n/a	0	0	0.0	6e−2	4e−2	n/a
M(F,CIRCULAR)	4	7	96	26	0.1	6e−3	1e−2	n/a
YB(F,CIRCULAR)	5	4	7	8	0.1	3e−3	3e−2	n/a
ML(V,FIXED)	16	2	273	109	0.5	n/a	n/a	5e−16
SOR	293	n/a	1,214	434	0.2	n/a	n/a	1e−15

Table 6.45: $K = 6$, $b = (8, 4, 4, 8, 8, 4)$, $\text{Hypo}^{(1)}(3.0, 1.5)$, $\text{Erlang}^{(2)}(2.5, 5)$, $\text{Hypo}^{(3)}(0.8, 2)$, $\text{Hyper}^{(4)}(0.7, 0.7, (0.25, 0.75))$, $\text{Erlang}^{(5)}(9.0, 5)$, $\text{Hyper}^{(6)}(1.4, 1.4, (0.9, 0.1))$

	oIter	iIter	T	MF	MB	RE(ρ)	RE(E[X])	Res
CA	n/a	n/a	0	0	0.0	2e−1	2e−1	n/a
MVABLO	n/a	n/a	0	0	0.0	7e−2	5e−2	n/a
M(F,CIRCULAR)	4	7	159	47	0.1	7e−3	1e−2	n/a
YB(F,CIRCULAR)	5	5	11	48	0.1	4e−3	4e−2	n/a
ML(V,FIXED)	15	2	612	341	1.2	n/a	n/a	9e−16
SOR	382	n/a	3,377	1,443	0.5	n/a	n/a	1e−15

Table 6.46: $K = 7$, $b = (8, 4, 4, 8, 8, 4)$, $\text{Hypo}^{(1)}(3.0, 1.5)$, $\text{Erlang}^{(2)}(2.5, 5)$, $\text{Hypo}^{(3)}(0.8, 2)$, $\text{Hyper}^{(4)}(0.7, 0.7, (0.25, 0.75))$, $\text{Erlang}^{(5)}(9.0, 5)$, $\text{Hyper}^{(6)}(1.4, 1.4, (0.9, 0.1))$

	oIter	iIter	T	MF	MB	RE(ρ)	RE(E[X])	Res
CA	n/a	n/a	0	0	0.0	2e−1	2e−1	n/a
MVABLO	n/a	n/a	0	0	0.0	8e−2	7e−2	n/a
M(F,CIRCULAR)	4	8	243	76	0.2	8e−3	1e−2	n/a
YB(F,CIRCULAR)	6	5	21	257	0.2	6e−3	4e−2	n/a
ML(F,CIRCULAR)	9	2	1,178	514	2.8	n/a	n/a	2e−16
SOR	475	n/a	8,731	3,937	1.1	n/a	n/a	1e−15

Table 6.47: $K = 8$, $b = (8, 4, 4, 8, 8, 4)$, Hypo$^{(1)}(3.0, 1.5)$, Erlang$^{(2)}(2.5, 5)$, Hypo$^{(3)}(0.8, 2)$, Hyper$^{(4)}(0.7, 0.7, (0.25, 0.75))$, Erlang$^{(5)}(9.0, 5)$, Hyper$^{(6)}(1.4, 1.4, (0.9, 0.1))$

	oIter	iIter	T	MF	MB	RE(ρ)	RE(E[X])	Res
CA	n/a	n/a	0	0	0.0	2e−1	3e−1	n/a
MVABLO	n/a	n/a	0	0	0.0	8e−2	1e−1	n/a
M(F,CIRCULAR)	4	8	380	121	0.3	8e−3	2e−2	n/a
YB(F,CIRCULAR)	6	6	33	691	0.3	7e−3	4e−2	n/a
ML(F,CIRCULAR)	9	2	2,051	939	5.1	n/a	n/a	6e−16
SOR	586	n/a	20,099	9,438	2.0	n/a	n/a	1e−15

Tables 6.48, 6.49, 6.50, and 6.51 present the results of the unbalanced network for 5, 6, 7, 8 customers, respectively. For these cases, CA and MVABLO are not able to achieve more than 1 digit of accuracy in neither performance measure. M approximates both performance measures with higher accuracy than CA, MV-ABLO, and YB by acquiring almost 3.5 to 4 digits of accuracy. The number of inner iterations of M is more than the number of inner iterations of YB, and the number of outer iterations of YB is more than the number of the outer iterations of M for all numbers of customers. Having performed less flops than YB for $K > 5$, M method turns out to be the more accurate and efficient method for the unbalanced case between the two. Indeed, the subnetworks' sizes of M are bigger compared to the subnetworks' sizes of YB, and YB performs less flops to obtain solutions for the subnetworks than M. Thus obtaining a solution for the state dependent exponential network in YB requires more flops than obtaining solutions for the subnetworks in M using ML. SOR is unable to converge within 1,000 iterations for $K \in \{7, 8\}$ and performs more flops than ML. In that respect, ML is more accurate and efficient than SOR for all numbers of customers.

Table 6.48: $K = 5$, $b =$ (8,4,4,8,8,4), Hypo$^{(1)}(3.0, 0.1)$, Erlang$^{(2)}(25.0, 5)$, Hypo$^{(3)}(70.0, 60.0)$, Hyper$^{(4)}(0.7, 0.7, (0.25, 0.75))$, Erlang$^{(5)}(0.1, 5)$, Hyper$^{(6)}(0.001, 0.001, (0.9, 0.1))$

	oIter	iIter	T	MF	MB	RE(ρ)	RE(E[X])	Res
CA	n/a	n/a	0	0	0.0	5e−1	2e−1	n/a
MVABLO	n/a	n/a	0	0	0.0	5e−1	3e−1	n/a
M(F,CIRCULAR)	2	3	25	7	0.1	6e−4	3e−4	n/a
YB(F,CIRCULAR)	3	2	3	4	0.1	2e−2	7e−3	n/a
ML(V,FIXED)	6	2	98	39	0.5	n/a	n/a	3e−16
SOR	465	n/a	1,928	689	0.2	n/a	n/a	9e−16

Table 6.49: $K = 6$, $b = (8,4,4,8,8,4)$, $\text{Hypo}^{(1)}(3.0, 0.1)$, $\text{Erlang}^{(2)}(25.0, 5)$, $\text{Hypo}^{(3)}(70.0, 60.0)$, $\text{Hyper}^{(4)}(0.7, 0.7, (0.25, 0.75))$, $\text{Erlang}^{(5)}(0.1, 5)$, $\text{Hyper}^{(6)}(0.001, 0.001, (0.9, 0.1))$

	oIter	iIter	T	MF	MB	RE(ρ)	RE(E[X])	Res
CA	n/a	n/a	0	0	0.0	5e−1	3e−1	n/a
MVABLO	n/a	n/a	0	0	0.0	6e−1	3e−1	n/a
M(F,CIRCULAR)	2	6	80	24	0.1	3e−4	2e−4	n/a
YB(F,CIRCULAR)	4	2	6	39	0.1	7e−3	5e−3	n/a
ML(V,FIXED)	43	2	1,538	965	1.2	n/a	n/a	7e−16
SOR	895	n/a	7,918	3,380	0.5	n/a	n/a	1e−15

Table 6.50: $K = 7$, $b = (8,4,4,8,8,4)$, $\text{Hypo}^{(1)}(3.0, 0.1)$, $\text{Erlang}^{(2)}(25.0, 5)$, $\text{Hypo}^{(3)}(70.0, 60.0)$, $\text{Hyper}^{(4)}(0.7, 0.7, (0.25, 0.75))$, $\text{Erlang}^{(5)}(0.1, 5)$, $\text{Hyper}^{(6)}(0.001, 0.001, (0.9, 0.1))$

	oIter	iIter	T	MF	MB	RE(ρ)	RE(E[X])	Res
CA	n/a	n/a	0	0	0.0	5e−1	4e−1	n/a
MVABLO	n/a	n/a	0	0	0.0	5e−1	5e−1	n/a
M(F,CIRCULAR)	2	7	129	44	0.2	3e−4	2e−4	n/a
YB(F,CIRCULAR)	4	2	11	176	0.2	7e−3	4e−3	n/a
ML(F,CIRCULAR)	15	2	1,996	3,425	2.8	n/a	n/a	8e−16
SOR	1,000*	n/a	18,379	8,286	1.1	3e−11	2e−11	2e−14

Table 6.51: $K = 8$, $b = (8,4,4,8,8,4)$, $\text{Hypo}^{(1)}(3.0, 0.1)$, $\text{Erlang}^{(2)}(25.0, 5)$, $\text{Hypo}^{(3)}(70.0, 60.0)$, $\text{Hyper}^{(4)}(0.7, 0.7, (0.25, 0.75))$, $\text{Erlang}^{(5)}(0.1, 5)$, $\text{Hyper}^{(6)}(0.001, 0.001, (0.9, 0.1))$

	oIter	iIter	T	MF	MB	RE(ρ)	RE(E[X])	Res
CA	n/a	n/a	0	0	0.0	5e−1	5e−1	n/a
MVABLO	n/a	n/a	0	0	0.0	5e−1	5e−1	n/a
M(F,CIRCULAR)	2	7	169	59	0.3	3e−4	2e−4	n/a
YB(F,CIRCULAR)	4	2	18	629	0.3	7e−3	4e−3	n/a
ML(F,CIRCULAR)	17	2	5,553	10,555	5.2	n/a	n/a	2e−16
SOR	1,000*	n/a	34,301	16,103	2.0	6e−10	4e−10	4e−13

6.5 Problem 5

The fifth problem consists of a closed QN with 8 queues and it has arbitrary routing probabilities among queues. The problem is considered in order to investigate the behavior of methods on larger problems. Indeed, the problem includes cases with state space sizes over one million. The queues have servers with three hyperexponential service distributions each with 2 phases, three hypoexponential service distributions each with 2 phases, and two Erlang service distributions each with 5 phases. Buffer sizes of queues are defined by $b = (8, 9, 9, 6, 6, 9, 9, 7)$. Subnetworks resulting from the decomposition procedure of the M method is given by the sets {{1},{2},{3},{4},{5},{6},{7},{8}}, {{1},{2},{3,4,5},{6},{7},{8}}, {{1},{2},{3,4,5},{6,7,8}}, and {{1,2},{3,4,5},{6,7,8}} for 6, 7, 8, 9 customers, respectively. In this case, the QN has state space sizes of 85,991, 236,172, 578,592, 1,291,130, and HLM number of states of 1,716, 3,430, 6,418, 11,359 for 6, 7, 8, 9 customers, respectively. The number of nonzero elements needed for the sparse representation of Q for 6, 7, 8, 9 customers are 701,763, 2,087,734, 5,450,599, and 12,807,256, respectively. The topology of the network is depicted in Figure 6.5 and Tables 6.52 through 6.59 give the results obtained with different methods while analyzing balanced and unbalanced cases of problem 5.

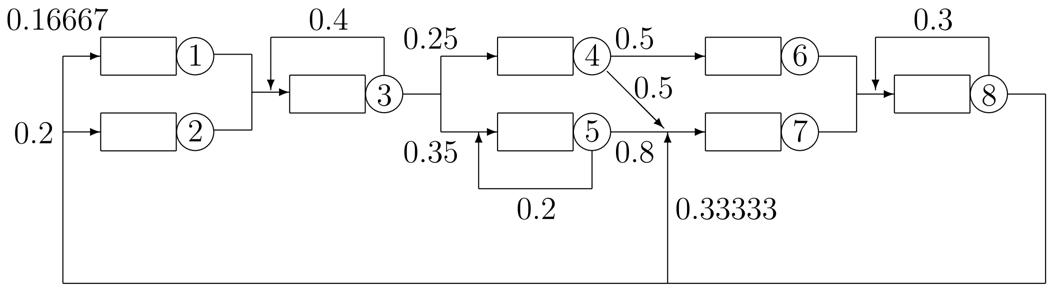

Figure 6.5: Problem 5.

Tables 6.52, 6.53, 6.54, and 6.55 present the results of the balanced network for 6, 7, 8, 9 customers, respectively. For all numbers of customers, MVABLO yields 1.5 digits accurate results for both performance measures. Although CA provides at most 1.5 digits of accuracy for both performance measures, its results are not better than those of MVABLO. YB method's approximations for utilization values are 0.5 digits more accurate than the results of M. Yet, YB performs more flops than M. Moreover, YB performs more flops than ML for $K = \{7, 9\}$ and YB is not useful for these cases of the problem. By providing 2 to 2.5 digits of accuracy in the approximations, M becomes, if not the more accurate, the more efficient method when compared with YB. For none of the numbers of customers, SOR performs less flops than ML.

Table 6.52: $K = 6$, $b = (8, 9, 9, 6, 6, 9, 9, 7)$, $\text{Hypo}^{(1)}(1.5, 2.0)$, $\text{Hyper}^{(2)}(1.1, 1.1, (0.1, 0.9))$, $\text{Erlang}^{(3)}(18.0, 5)$, $\text{Hyper}^{(4)}(0.9, 0.9, (0.85, 0.15))$, $\text{Hypo}^{(5)}(1.5, 10.0)$, $\text{Hypo}^{(6)}(0.5, 4.0)$, $\text{Hyper}^{(7)}(3.5, 3.5, (0.25, 0.75))$, $\text{Erlang}^{(8)}(30, 5)$

	oIter	iIter	T	MF	MB	RE(ρ)	RE(E[X])	Res
CA	n/a	n/a	0	0	0.0	4e−2	4e−2	n/a
MVABLO	n/a	n/a	0	0	0.0	4e−2	4e−2	n/a
M(F,CIRCULAR)	5	4	39	9	0.1	8e−3	1e−2	n/a
YB(V,FIXED)	5	6	45	106	0.3	1e−3	2e−2	n/a
ML(F,CIRCULAR)	9	2	4,844	1,983	7.2	n/a	n/a	1e−16
SOR	612	n/a	22,526	10,616	2.2	n/a	n/a	1e−15

Table 6.53: $K = 7$, $b = (8, 9, 9, 6, 6, 9, 9, 7)$, $\text{Hypo}^{(1)}(1.5, 2.0)$, $\text{Hyper}^{(2)}(1.1, 1.1, (0.1, 0.9))$, $\text{Erlang}^{(3)}(18.0, 5)$, $\text{Hyper}^{(4)}(0.9, 0.9, (0.85, 0.15))$, $\text{Hypo}^{(5)}(1.5, 10.0)$, $\text{Hypo}^{(6)}(0.5, 4.0)$, $\text{Hyper}^{(7)}(3.5, 3.5, (0.25, 0.75))$, $\text{Erlang}^{(8)}(30, 5)$

	oIter	iIter	T	MF	MB	RE(ρ)	RE(E[X])	Res
CA	n/a	n/a	0	0	0.0	6e−2	8e−2	n/a
MVABLO	n/a	n/a	0	0	0.0	4e−2	4e−2	n/a
M(V,FIXED)	5	7	139	38	0.2	6e−3	9e−3	n/a
YB(V,FIXED)	5	6	131	5,931	0.7	1e−3	2e−2	n/a
ML(F,CIRCULAR)	10	2	15,645	5,413	18.5	n/a	n/a	0e+0
SOR	787	n/a	76,160	38,672	5.9	n/a	n/a	1e−15

Table 6.54: $K = 8$, $b = (8,9,9,6,6,9,9,7)$, $\text{Hypo}^{(1)}(1.5, 2.0)$, $\text{Hyper}^{(2)}(1.1, 1.1, (0.1, 0.9))$, $\text{Erlang}^{(3)}(18.0, 5)$, $\text{Hyper}^{(4)}(0.9, 0.9, (0.85, 0.15))$, $\text{Hypo}^{(5)}(1.5, 10.0)$, $\text{Hypo}^{(6)}(0.5, 4.0)$, $\text{Hyper}^{(7)}(3.5, 3.5, (0.25, 0.75))$, $\text{Erlang}^{(8)}(30, 5)$

	oIter	iIter	T	MF	MB	RE(ρ)	RE(E[X])	Res
CA	n/a	n/a	0	0	0.0	8e−2	1e−1	n/a
MVABLO	n/a	n/a	0	0	0.0	4e−2	4e−2	n/a
M(V,FIXED)	4	11	487	138	0.2	6e−3	7e−3	n/a
YB(V,FIXED)	5	7	232	566	1.3	1e−3	2e−2	n/a
ML(F,CIRCULAR)	10	2	25,258	12,430	43.1	n/a	n/a	1e−16
SOR	309	n/a	100,000*	38,052	14.4	1e−5	2e−5	6e−9

Table 6.55: $K = 9$, $b = (8,9,9,6,6,9,9,7)$, $\text{Hypo}^{(1)}(1.5, 2.0)$, $\text{Hyper}^{(2)}(1.1, 1.1, (0.1, 0.9))$, $\text{Erlang}^{(3)}(18.0, 5)$, $\text{Hyper}^{(4)}(0.9, 0.9, (0.85, 0.15))$, $\text{Hypo}^{(5)}(1.5, 10.0)$, $\text{Hypo}^{(6)}(0.5, 4.0)$, $\text{Hyper}^{(7)}(3.5, 3.5, (0.25, 0.75))$, $\text{Erlang}^{(8)}(30, 5)$

	oIter	iIter	T	MF	MB	RE(ρ)	RE(E[X])	Res
CA	n/a	n/a	0	0	0.0	1e−1	2e−1	n/a
MVABLO	n/a	n/a	0	0	0.0	5e−2	5e−2	n/a
M(F,CIRCULAR)	4	6	2,447	1,128	1.3	5e−3	8e−3	n/a
YB(F,CIRCULAR)	5	5	815	38,583	2.4	1e−3	2e−2	n/a
ML(F,CIRCULAR)	10	2	54,231	26,324	92.6	n/a	n/a	5e−16
SOR	200	n/a	100,000*	56,018	31.9	3e−4	7e−4	1e−7

The results obtained for the unbalanced case can be seen in Tables 6.56, 6.57, 6.58, and 6.59. For $K = 6$, CA and MVABLO's approximations yield 2 digits accurate results for utilization values and 2.5 digits accurate results for mean queue length values. By performing negligible number of floating–point operations, CA and MVABLO turn out to be the most accurate and efficient methods in this case. Yet, as the number of customers increases, M yields at least 3.5 digits accurate results for both performance measures, while CA and MVABLO are able to provide at most 1.5 digits of accuracy. YB preserves at least 2 digits of accuracy for utilization values and 2.5 digits of accuracy for mean queue length values for $K > 6$. Yet, the flop counts of M are less than the flop counts of YB, and thus, M emerges as the more accurate and efficient method between the two approximative methods. SOR does not converge to the predefined tolerance within the predefined iteration count or time bound.

Table 6.56: $K = 6$, $b = (8, 9, 9, 6, 6, 9, 9, 7)$, Hypo$^{(1)}(200.0, 90.0)$, Hyper$^{(2)}(1,000.0, 1,000.0, (0.1, 0.9))$, Erlang$^{(3)}(0.15, 5)$, Hyper$^{(4)}(0.008, 0.008, (0.85, 0.15))$, Hypo$^{(5)}(8,000.0, 5,000.0)$, Hypo$^{(6)}(0.1, 0.05)$, Hyper$^{(7)}(10.0, 10.0, (0.25, 0.75))$, Erlang$^{(8)}(30.0, 5)$

	oIter	iIter	T	MF	MB	RE(ρ)	RE(E[X])	Res
CA	n/a	n/a	0	0	0.0	1e−2	5e−3	n/a
MVABLO	n/a	n/a	0	0	0.0	1e−2	5e−3	n/a
M(F,CIRCULAR)	4	3	34	8	0.1	2e−2	8e−3	n/a
YB(V,FIXED)	4	4	55	1,846	0.3	4e−2	3e−2	n/a
ML(V,FIXED)	31	2	6,806	28,752	6.4	n/a	n/a	6e−16
SOR	1,000*	n/a	36,842	17,361	2.2	8e−9	3e−8	5e−12

Table 6.57: $K = 7$, $b = (8, 9, 9, 6, 6, 9, 9, 7)$, Hypo$^{(1)}(200.0, 90.0)$, Hyper$^{(2)}(1,000.0, 1,000.0, (0.1, 0.9))$, Erlang$^{(3)}(0.15, 5)$, Hyper$^{(4)}(0.008, 0.008, (0.85, 0.15))$, Hypo$^{(5)}(8,000.0, 5,000.0)$, Hypo$^{(6)}(0.1, 0.05)$, Hyper$^{(7)}(10.0, 10.0, (0.25, 0.75))$, Erlang$^{(8)}(30.0, 5)$

	oIter	iIter	T	MF	MB	RE(ρ)	RE(E[X])	Res
CA	n/a	n/a	0	0	0.0	6e−2	6e−2	n/a
MVABLO	n/a	n/a	0	0	0.0	5e−2	1e−1	n/a
M(V,FIXED)	4	4	116	32	0.2	7e−3	2e−3	n/a
YB(F,CIRCULAR)	4	3	117	5,663	0.7	2e−2	3e−3	n/a
ML(V,FIXED)	45	2	20,817	42,597	16.5	n/a	n/a	5e−16
SOR	1,000*	n/a	96,856	49,184	5.9	4e−7	2e−6	3e−10

Table 6.58: $K = 8$, $b = (8, 9, 9, 6, 6, 9, 9, 7)$, Hypo$^{(1)}(200.0, 90.0)$, Hyper$^{(2)}(1,000.0, 1,000.0, (0.1, 0.9))$, Erlang$^{(3)}(0.15, 5)$, Hyper$^{(4)}(0.008, 0.008, (0.85, 0.15))$, Hypo$^{(5)}(8,000.0, 5,000.0)$, Hypo$^{(6)}(0.1, 0.05)$, Hyper$^{(7)}(10.0, 10.0, (0.25, 0.75))$, Erlang$^{(8)}(30.0, 5)$

	oIter	iIter	T	MF	MB	RE(ρ)	RE(E[X])	Res
CA	n/a	n/a	0	0	0.0	5e−2	1e−1	n/a
MVABLO	n/a	n/a	0	0	0.0	5e−2	2e−1	n/a
M(F,CIRCULAR)	4	4	357	155	0.3	2e−4	6e−4	n/a
YB(F,CIRCULAR)	4	3	403	31,708	1.3	2e−2	3e−3	n/a
ML(V,FIXED)	46	2	48,429	123,691	38.5	n/a	n/a	3e−16
SOR	435	n/a	100,000*	53,554	14.4	5e−4	2e−3	4e−7

Table 6.59: $K = 9$, $b = (8, 9, 9, 6, 6, 9, 9, 7)$, $\text{Hypo}^{(1)}(200.0, 90.0)$, $\text{Hyper}^{(2)}(1,000.0, 1,000.0, (0.1, 0.9))$, $\text{Erlang}^{(3)}(0.15, 5)$, $\text{Hyper}^{(4)}(0.008, 0.008, (0.85, 0.15))$, $\text{Hypo}^{(5)}(8,000.0, 5,000.0)$, $\text{Hypo}^{(6)}(0.1, 0.05)$, $\text{Hyper}^{(7)}(10.0, 10.0, (0.25, 0.75))$, $\text{Erlang}^{(8)}(30.0, 5)$

	oIter	iIter	T	MF	MB	RE(ρ)	RE(E[X])	Res
CA	n/a	n/a	0	0	0.0	5e−2	1e−1	n/a
MVABLO	n/a	n/a	0	0	0.0	5e−2	3e−1	n/a
M(F,CIRCULAR)	4	5	1,859	1,884	1.9	2e−4	6e−4	n/a
YB(F,CIRCULAR)	4	3	674	34,848	2.4	2e−2	2e−3	n/a
ML(F,CIRCULAR)	12	2	67,760	241,214	92.6	n/a	n/a	1e−16
SOR	141	n/a	100,000*	39,515	31.9	2e−2	8e−2	2e−5

Another set of experiments for problem 5 is conducted by setting $b = (5, 9, 9, 5, 5, 9, 9, 5)$. With this setting for 6, 7, 8, 9 customers, the closed QN has state space sizes of 85,980, 235,960, 576,670, 1,279,970, and HLM numbers of states of 1,712, 3,400, 6,291, 10,960, respectively. The nonzero elements needed for the sparse representation of Q for 6, 7, 8, 9 customers are 701,703, 2,086,332, 5,435,949, and 12,712,452, respectively. Tables 6.60 through 6.67 show the results of these experiments.

The results obtained for the balanced network can be seen in Tables 6.60, 6.61, 6.62, and 6.63 for 6, 7, 8, 9 customers, respectively. In the balanced case, MVABLO yields 1.5 digits of accuracy for both performance measures in all cases. CA yields 1 digit of accuracy for average number of customer values in all cases and 1 digit of accuracy for utilization values in all cases except $K = 7$. We see that utilization values obtained using M yield 2.5 digits of accuracy. Those obtained with YB are equally good or slightly better. On the other hand, mean queue length values obtained using YB yield 2 digits of accuracy; those obtained with M are equally good or slightly better. Since ML converges within 10 iterations in all cases, YB ends up performing more flops than ML for $K > 6$, while M performs 5 to 20 times less flops than ML. Additionally, when we consider the accuracy of M and compare its flop counts with ML and YB, we conclude that M is the most efficient method.

Table 6.60: $K = 6$, $b = (5, 9, 9, 5, 5, 9, 9, 5)$, Hypo$^{(1)}(1.5, 2.0)$, Hyper$^{(2)}(1.1, 1.1, (0.1, 0.9))$, Erlang$^{(3)}(18.0, 5)$, Hyper$^{(4)}(0.9, 0.9, (0.85, 0.15))$, Hypo$^{(5)}(1.5, 10.0)$, Hypo$^{(6)}(0.5, 4.0)$, Hyper$^{(7)}(3.5, 3.5, (0.25, 0.75))$, Erlang$^{(8)}(30, 5)$

	oIter	iIter	T	MF	MB	RE(ρ)	RE(E[X])	Res
CA	n/a	n/a	0	0	0.0	9e−2	1e−1	n/a
MVABLO	n/a	n/a	0	0	0.0	4e−2	4e−2	n/a
M(V,FIXED)	4	12	806	258	0.3	4e−3	6e−3	n/a
YB(F,CIRCULAR)	5	4	45	767	0.3	1e−3	2e−2	n/a
ML(F,CIRCULAR)	9	2	4,793	1,980	7.2	n/a	n/a	1e−16
SOR	612	n/a	25,954	10,594	2.2	n/a	n/a	1e−15

Table 6.61: $K = 7$, $b = (5, 9, 9, 5, 5, 9, 9, 5)$, Hypo$^{(1)}(1.5, 2.0)$, Hyper$^{(2)}(1.1, 1.1, (0.1, 0.9))$, Erlang$^{(3)}(18.0, 5)$, Hyper$^{(4)}(0.9, 0.9, (0.85, 0.15))$, Hypo$^{(5)}(1.5, 10.0)$, Hypo$^{(6)}(0.5, 4.0)$, Hyper$^{(7)}(3.5, 3.5, (0.25, 0.75))$, Erlang$^{(8)}(30, 5)$

	oIter	iIter	T	MF	MB	RE(ρ)	RE(E[X])	Res
CA	n/a	n/a	0	0	0.0	6e−2	8e−2	n/a
MVABLO	n/a	n/a	0	0	0.0	4e−2	4e−2	n/a
M(V,FIXED)	4	13	1,393	697	0.5	5e−3	7e−3	n/a
YB(F,CIRCULAR)	5	4	122	5,445	0.7	1e−3	2e−2	n/a
ML(F,CIRCULAR)	10	2	11,157	5,306	18.5	n/a	n/a	0e+0
SOR	790	n/a	87,604	38,713	5.9	n/a	n/a	1e−15

Table 6.62: $K = 8$, $b = (5, 9, 9, 5, 5, 9, 9, 5)$, Hypo$^{(1)}(1.5, 2.0)$, Hyper$^{(2)}(1.1, 1.1, (0.1, 0.9))$, Erlang$^{(3)}(18.0, 5)$, Hyper$^{(4)}(0.9, 0.9, (0.85, 0.15))$, Hypo$^{(5)}(1.5, 10.0)$, Hypo$^{(6)}(0.5, 4.0)$, Hyper$^{(7)}(3.5, 3.5, (0.25, 0.75))$, Erlang$^{(8)}(30, 5)$

	oIter	iIter	T	MF	MB	RE(ρ)	RE(E[X])	Res
CA	n/a	n/a	0	0	0.0	9e−2	1e−1	n/a
MVABLO	n/a	n/a	0	0	0.0	5e−2	5e−2	n/a
M(V,FIXED)	4	14	2,373	1,318	0.7	5e−3	8e−3	n/a
YB(V,FIXED)	5	7	365	30,002	1.3	2e−3	2e−2	n/a
ML(F,CIRCULAR)	10	2	24,946	12,196	42.9	n/a	n/a	2e−16
SOR	438	n/a	100,000*	53,736	14.3	7e−7	1e−6	4e−10

Table 6.63: $K = 9$, $b = (5, 9, 9, 5, 5, 9, 9, 5)$, $\text{Hypo}^{(1)}(1.5, 2.0)$, $\text{Hyper}^{(2)}(1.1, 1.1, (0.1, 0.9))$, $\text{Erlang}^{(3)}(18.0, 5)$, $\text{Hyper}^{(4)}(0.9, 0.9, (0.85, 0.15))$, $\text{Hypo}^{(5)}(1.5, 10.0)$, $\text{Hypo}^{(6)}(0.5, 4.0)$, $\text{Hyper}^{(7)}(3.5, 3.5, (0.25, 0.75))$, $\text{Erlang}^{(8)}(30, 5)$

	oIter	iIter	T	MF	MB	RE(ρ)	RE(E[X])	Res
CA	n/a	n/a	0	0	0.0	1e−1	2e−1	n/a
MVABLO	n/a	n/a	0	0	0.0	5e−2	6e−2	n/a
M(F,CIRCULAR)	4	6	2,301	1,060	1.7	6e−3	9e−3	n/a
YB(F,CIRCULAR)	5	5	1,155	131,150	2.3	3e−3	2e−2	n/a
ML(F,CIRCULAR)	10	2	52,789	25,683	91.5	n/a	n/a	6e−16
SOR	203	n/a	100,000*	56,345	31.6	4e−4	9e−4	1e−7

The results obtained for the unbalanced network can be seen in Tables 6.64, 6.65, 6.66, and 6.67 for 6, 7, 8, 9 customers, respectively. In all cases CA and MVABLO yield 1.5 digits accuracy for utilization values and 0.5 to 1.5 digits accuracy for mean queue length values. Results with YB are 2 digits accurate in all cases. On the other hand, results with M provide 2.5 to 4 digits of accuracy for both performance measures and performs at least 5 times less flops than YB. Hence, M emerges again as the more accurate and efficient method between the two.

Table 6.64: $K = 6$, $b = (5, 9, 9, 5, 5, 9, 9, 5)$, $\text{Hypo}^{(1)}(200.0, 90.0)$, $\text{Hyper}^{(2)}(1,000.0, 1,000.0, (0.1, 0.9))$, $\text{Erlang}^{(3)}(0.15, 5)$, $\text{Hyper}^{(4)}(0.008, 0.008, (0.85, 0.15))$, $\text{Hypo}^{(5)}(8,000.0, 5,000.0)$, $\text{Hypo}^{(6)}(0.1, 0.05)$, $\text{Hyper}^{(7)}(10.0, 10.0, (0.25, 0.75))$, $\text{Erlang}^{(8)}(30.0, 5)$

	oIter	iIter	T	MF	MB	RE(ρ)	RE(E[X])	Res
CA	n/a	n/a	0	0	0.0	7e−2	7e−2	n/a
MVABLO	n/a	n/a	0	0	0.0	6e−2	1e−1	n/a
M(F,CIRCULAR)	4	4	359	169	0.4	4e−3	2e−3	n/a
YB(V,FIXED)	4	4	86	4,253	0.3	2e−2	1e−2	n/a
ML(V,FIXED)	45	2	8,633	36,852	6.4	n/a	n/a	6e−16
SOR	1,000*	n/a	42,446	17,307	2.2	4e−9	1e−8	4e−12

Table 6.65: $K = 7$, $b = (5, 9, 9, 5, 5, 9, 9, 5)$, $\text{Hypo}^{(1)}(200.0, 90.0)$, $\text{Hyper}^{(2)}(1,000.0, 1,000.0, (0.1, 0.9))$, $\text{Erlang}^{(3)}(0.15, 5)$, $\text{Hyper}^{(4)}(0.008, 0.008, (0.85, 0.15))$, $\text{Hypo}^{(5)}(8,000.0, 5,000.0)$, $\text{Hypo}^{(6)}(0.1, 0.05)$, $\text{Hyper}^{(7)}(10.0, 10.0, (0.25, 0.75))$, $\text{Erlang}^{(8)}(30.0, 5)$

	oIter	iIter	T	MF	MB	RE(ρ)	RE(E[X])	Res
CA	n/a	n/a	0	0	0.0	6e−2	1e−1	n/a
MVABLO	n/a	n/a	0	0	0.0	7e−2	2e−1	n/a
M(V,FIXED)	4	11	934	451	0.5	2e−4	7e−4	n/a
YB(F,CIRCULAR)	4	3	96	2,883	0.7	2e−2	1e−2	n/a
ML(V,FIXED)	45	2	20,742	41,487	16.5	n/a	n/a	7e−16
SOR	1,000*	n/a	96,834	49,000	5.9	4e−9	1e−8	4e−12

Table 6.66: $K = 8$, $b = (5, 9, 9, 5, 5, 9, 9, 5)$, $\text{Hypo}^{(1)}(200.0, 90.0)$, $\text{Hyper}^{(2)}(1,000.0, 1,000.0, (0.1, 0.9))$, $\text{Erlang}^{(3)}(0.15, 5)$, $\text{Hyper}^{(4)}(0.008, 0.008, (0.85, 0.15))$, $\text{Hypo}^{(5)}(8,000.0, 5,000.0)$, $\text{Hypo}^{(6)}(0.1, 0.05)$, $\text{Hyper}^{(7)}(10.0, 10.0, (0.25, 0.75))$, $\text{Erlang}^{(8)}(30.0, 5)$

	oIter	iIter	T	MF	MB	RE(ρ)	RE(E[X])	Res
CA	n/a	n/a	0	0	0.0	6e−2	2e−1	n/a
MVABLO	n/a	n/a	0	0	0.0	7e−2	3e−1	n/a
M(V,FIXED)	4	11	1,453	817	0.7	2e−4	6e−4	n/a
YB(V,FIXED)	4	4	259	11,322	1.3	2e−2	1e−2	n/a
ML(V,FIXED)	45	2	47,105	128,651	38.2	n/a	n/a	7e−16
SOR	438	n/a	100,000*	53,736	14.3	7e−5	2e−4	7e−8

Table 6.67: $K = 9$, $b = (5, 9, 9, 5, 5, 9, 9, 5)$, $\text{Hypo}^{(1)}(200.0, 90.0)$, $\text{Hyper}^{(2)}(1,000.0, 1,000.0, (0.1, 0.9))$, $\text{Erlang}^{(3)}(0.15, 5)$, $\text{Hyper}^{(4)}(0.008, 0.008, (0.85, 0.15))$, $\text{Hypo}^{(5)}(8,000.0, 5,000.0)$, $\text{Hypo}^{(6)}(0.1, 0.05)$, $\text{Hyper}^{(7)}(10.0, 10.0, (0.25, 0.75))$, $\text{Erlang}^{(8)}(30.0, 5)$

	oIter	iIter	T	MF	MB	RE(ρ)	RE(E[X])	Res
CA	n/a	n/a	0	0	0.0	5e−2	2e−1	n/a
MVABLO	n/a	n/a	0	0	0.0	7e−2	4e−1	n/a
M(V,FIXED)	4	12	2,130	1,636	1.1	2e−4	5e−4	n/a
YB(F,CIRCULAR)	4	3	648	35,245	2.3	2e−2	9e−3	n/a
ML(F,CIRCULAR)	13	2	70,752	158,032	91.5	n/a	n/a	2e−16
SOR	203	n/a	100,000*	56,345	31.6	4e−3	1e−2	4e−6

In conclusion, CA and MVABLO produce acceptable results for problems which have queues with balanced service distributions and have small number of queues subject to blocking. On the other hand, although M and YB present relatively more accurate results for all problems, they present results with at least 2 digits accuracy for unbalanced cases. Also, unlike the results obtained with CA and MVABLO, an increase in the number of queues subject to blocking causes little or no effect in the results obtained with M and YB. Therefore, M and YB arise as better methods than CA and MVABLO for analyzing problems with unbalanced service demands and many queues subject to blocking. When we compare the accuracies of M and YB, especially in the problems with unbalanced service demands and many queues subject to blocking, we see that M can produce at least 0.5 digits more accurate results for utilization values than YB. When we compare the efficiencies of M and YB, we see that the number of flops performed by YB to compute arrival rates of queues mostly depends on the number of flops performed for obtaining the solution of the exponential network generated in each iteration step. Therefore, for problems which require small number of flops for the solution of the exponential network, YB executes less flops than M. Also, for problems which result in subnetworks with large number of queues for M, YB may end up performing less flops than M through its approximation process. Consequently, efficiencies of M and YB depend heavily on the particular problem. When we compare ML and SOR methods, we see that ML achieves convergence within 100 iterations in all problems. Yet, SOR does not converge in 1,000 iterations or 100,000 seconds for some of the problems. Clearly, number of iterations determines the number of flops performed by the methods, and ML performs less flops than SOR in most problems. Even though SOR method takes less space in memory than ML, in most of the problems ML requires less memory than the corresponding sparse representation, thereby, being capable of solving variants of the problems with larger number of customers. Since M and YB are based on decomposition, the space requirements of M and YB are smaller than that of ML and SOR for big problems. Actually, the usage of ML in M and YB introduces another dimension of scalability to the space requirements of the two methods. A summary of the iteration counts and relative accuracies obtained for all problems using the methods CA, MVABLO, M, and YB are given in Table

6.68.

Table 6.68: Average number of outer iterations performed by M and YB, and average accuracy for utilization and mean queue length values in all problems.

	oIter	RE(ρ)	RE(E[X])
CA	n/a	2e−1	2e−1
MVABLO	n/a	2e−1	2e−1
M	4	6e−3	8e−3
YB	5	9e−3	2e−2

Chapter 7

Conclusion

In this thesis, we consider two approximative iterative methods based on decomposition from the literature, namely Marie's method and Yao and Buzacott's method. It is shown that these methods can be used for analyzing closed QNs with phase type service distributions and arbitrary buffer sizes. While analyzing such closed QNs, subnetworks resulting from decomposition can be represented using Kronecker products. This is shown to add another level of scalability to the methods by requiring less space than the ordinary sparse representation of subnetworks. Furthermore, the Kronecker representation of subnetworks enables the use of a multilevel method in the solution procedures of Marie's method and Yao and Buzacott's method.

The effect of using the multilevel method is analyzed through a set of numerical experiments, which show that the number of iterations and floating point operations taken by the multilevel method are generally much smaller than those of the SOR method. Thus, the employment of the multilevel method within Marie's method and Yao and Buzacott's method makes the methods more efficient. The methods are also compared with two analytical methods from the literature, namely the convolution algorithm and Akyildiz's mean value analysis, on a number of examples and the cases in which Marie's method and Yao and Buzacott's method yield better approximations are identified. Indeed, Marie's method and Yao and Buzacott's method yield results with relative errors smaller than 10^{-5} for unbalanced cases of problems 2, 3, and 4. We see that Marie's method and Yao and Buzacott's method yield more accurate results for relatively crowded closed QNs with unbalanced service demands. In general, Marie's method approximates the performance measures of utilization and average number of customers in queues at least half a digit better than Yao and Buzacott's method. The efficiency of the algorithms may present different behaviour for different types of networks. For instance, an unbalanced decomposition of the network in Marie's method may cause Marie's method to be less efficient than Yao and Buzacott's method, whereas Yao and Buzacott's method may be less efficient than Marie's method for problems which have a large high level model tieing together the low level models in the Kronecker representation of the closed QN and performs an extensive number of floating point operations to obtain the steady–state solution of the state–dependent exponential network.

Being fixed point iterations, Marie's method and Yao and Buzacott's method are analyzed for the existence of a fixed–point and it is proved that a fixed–point exists for each method. Complexity analysis of the methods for one iteration is also given together with the complexity analysis of the other methods used.

As an extension, numerical experiments can be conducted for multiple server queues by modifying the software to include multiple servers. Other than using aggregation on phases, a new partitioning of the state space for phase–type service distributions can be investigated for the ML method. This may introduce a reduction in computational complexity and therefore time for the ML method. As future work, the uniqueness of the fixed–point in Marie's method and Yao and Buzacott's method can be investigated. Necessary and sufficient conditions for the convergence of iterative methods based on decomposition can be given. Consequently, problem types that possess these features can be considered as case studies.

Bibliography

[1] Akyildiz, I.F. (1988) "On the Exact and Approximate Throughput Analysis of Closed Queueing Networks with Blocking." *IEEE Transactions on Software Engineering*, 14, 62–71.

[2] Akyildiz, I.F. (1988) "Mean Value Analysis for Blocking Queueing Networks." *IEEE Transactions on Software Engineering*, 14, 418–428.

[3] Altiok, T. (1989) "Approximate Analysis of Queues in Series with Phase–Type Service Time and Blocking." *Operations Research*, 37, 601–610.

[4] Balsamo, S. (2000). "Closed Queueing Networks with Finite Capacity Queues: Approximate Analysis." In R. Van Landeghem (ed.), *European Simulation Multiconnference - Simulation and Conference: Enablers for a Better Quality of Life*. Ghent, Belgium: SCS Europe, 593–600.

[5] Baskett, F., Chandy, K., Muntz, R., and Palacios, F. (1975). "Open, Closed, and Mixed Networks of Queues with Different Classes of Customers." *Journal of the ACM*, 22, 248–260.

[6] Baynat, B. and Dallery, Y. (1993) "A Unified View of Product-Form Approximation Techniques for General Closed Queueing Networks." *Performance Evaluation*, 18, 205–224.

[7] Baynat, B. and Dallery, Y. (1993). "Approximate Techniques for General Closed Queueing Networks with Subnetworks having Population Constraints." *European Journal of Operational Research*, 69, 250–264.

[8] Buchholz, P. (1994). "A Class of Hierarchical Queueing Networks and Their Analysis." *Queueing Systems*, 15, 59–80.

[9] Buchholz, P. (1998). "An Adaptive Aggregation/Disaggregation Algorithm for Hierarchical Markovian Models." *European Journal of Operational Research*, 116, 545–564.

[10] Buchholz, P. (1999). "Hierarchical Structuring of Superposed GSPNs." *IEEE Transactions on Software Engineering*, 25, 166–181.

[11] Buchholz, P. and Dayar, T. (2004). "Comparison of Multilevel Methods for Kronecker–based Markovian Representations." *Computing*, 73, 349–371.

[12] Buchholz, P. and Dayar, T. (2007). "On The Convergence of a Class of Multilevel Methods for Large, Sparse Markov Chains." to appear in *SIAM Journal on Matrix Analysis and Applications.*

[13] Buzen, J.P. (1973). "Computational Algorithms for Closed Queueing Networks with Exponential Servers." *Communications of the ACM*, 16, 527–531.

[14] Chapman, S.J. (2002). *MATLAB Programming for Engineers*. Brooks/Cole Pacific Grove.

[15] Dallery, Y. and Gershwin, S.B. (1992). "Manufacturing Flow Line Systems: A Review of Models and Analytical Results." *Queueing Systems*, 12, 3–94.

[16] Davio, M. (1981). "Kronecker Products and Shuffle Algebra." *IEEE Transactions on Computers*, C-30, 116–125.

[17] Dayar, T. (2006). "Analyzing Markov Chains Based on Kronecker Products." In A.N. Langville and W.J. Stewart (eds.), *MAM 2006: Markov Anniversary Meeting*. Raleigh, North Carolina: Boson Books, 279–300.

[18] Fernandes, P., Plateau, B., and Stewart, W.J. (1998). "Efficient Descriptor-Vector Multiplications in Stochastic Automata Networks." *Journal of the ACM*, 45, 381–414.

[19] Frein, Y. and Dallery, Y. (1989). 'Analysis of Cyclic Queueing Networks with Finite Buffers and Blocking Before Service." *Performance Evaluation*, 10, 197–210.

[20] Genz, A., Lin, Z., Jones, C., Luo, D. and Prenzel, T. (1991). "Fast Givens Goes Slow in MATLAB." *ACM SIGNUM Newsteller*, 26, 11–16.

[21] Gershwin, S.B. (1987). "An Efficient Decomposition Method for the Approximate Evaluation of Tandem Queues with Finite Storage Space and Blocking." *Operations Research*, 35, 291–305.

[22] Gordon, W.J. and Newell, G.F. (1967). "Closed Queueing Systems with Exponential Servers." *Operations Research*, 15, 252–267.

[23] Haverkort, B.R. (1998). *Performance of Computer Communication Systems: A Model-based Approach*. New York: John Wiley & Sons.

[24] Kühn, P.J. (1967). "Approximate Analysis of General Queueing Networks by Decomposition." *IEEE Transactions on Communications*, 27, 113–126.

[25] Mainkar, V. and Trivedi, K.S. (1996). "Sufficient Conditions for Existence of a Fixed Point in Stochastic Reward Net-Based Iterative Models." *IEEE Transactions on Software Engineering*, 22, 640–653.

[26] Marie, R. (1979). "An Approximate Analytical Method for General Queueing Networks." *IEEE Transactions on Software Engineering*, 5, 530–538.

[27] Meriç, A. *Software for Kronecker Representation and Decompositional Analysis of Closed Queueing Networks with Phase–Type Service Distributions and Arbitrary Buffer Sizes*, (2007). Available from `http://www.cs.bilkent.edu.tr/~tugrul/software.html`.

[28] Onvural, R. (1990). "Survey of Closed Queueing Networks with Blocking." *ACM Computing Surveys*, 22, 83–121.

[29] Onvural, R. and Perros, H.G. (1989). "Throughput Analysis in Cyclic Queueing Networks with Blocking." *IEEE Transactions on Software Engineering*, 15, 800–808.

[30] Ortega, J.M. and Rheinboldt, W.C. (1970) "Iterative Solution of Nonlinear Equations in Several Variables" *New York: Academic Press*

[31] Perros, H.G., Nilsson, A.A. and Liu, Y.C. (1988). "Approximate Analysis of Product-Form Type Queueing Networks with Blocking and Deadlock". *Performance Evaluation*, 8, 19–39.

[32] Reiser, M. (1981). "Mean Value Analysis and Convolution Method for Queue–dependent Servers in Closed Queueing Networks". *Performance Evaluation*, 1, 7–81.

[33] Reiser, M. and Lavenberg, S.S. (1980). "Mean Value Analysis of Closed Multichain Queueing Networks". *Journal of the ACM*, 22, 313–322.

[34] Rosenlicht, M. (1986). *Introduction to Analysis.* New York, Dover Publications, Inc.

[35] Rudin, W. (1964). *Principles of Mathematical Analysis.* McGraw–Hill.

[36] Saad, Y. (2003). *Iterative Methods for Sparse Linear Systems.* Philadelphia, SIAM.

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

[38] Suri, R. and Diehl, G.W. (1986). "A Variable Buffer Size Model and Its Use in Analytical Closed Queueing Networks with Blocking." *Management Science*, 32, 206–225.

[39] Uysal, E. and Dayar, T. (1998) "Iterative Methods based on Splittings for Stochastic Automata Networks." *European Journal of Operational Research*, 110, 166–186.

[40] Van Loan, C.F. (2000). "The Ubiquitous Kronecker Product." *Journal of Computational and Applied Mathematics*, 123, 85–100.

[41] Vroblefski, M., Ramesh, R. and Zionts, S. (2000). "General Open and Closed Queueing Networks with Blocking: A Unified Framework for Approximation." *INFORMS Journal on Computing*, 12, 299–316.

[42] Whitt, W. (1983). “The Queueing Network Analyzer.” *The Bell System Technical Journal*, 62, 2779–2815.

[43] Yao, D.D. and Buzacott, J.A. (1985). “Modelling a Class of State-Dependent Routing in Flexible Manufacturing Systems.” *Annals of Operations Research*, 3, 153–167.

[44] Yao, D.D. and Buzacott, J.A. (1985). “Queueing Models for a Flexible Machining Station Part II: The method of Coxian Phases.” *European Journal of Operational Research*, 19, 241–252.

[45] Yao, D.D. and Buzacott, J.A. (1986). “The Exponentialization Approach to Flexible Manufacturing Systems Models with General Processing Times.” *European Journal of Operational Research*, 24, 410–416.

Appendix A

Readme of software

```
1. A Closed QN is defined through text files located under a directory.
   The text files describing the queues are named as Txx.txt and
   ALFAxx.txt, where "xx" is the queue number. Routing probabilities
   among queues are defined by the text file P.txt.

   - Txx.txt represents phase transitions in the matrix T for service
     distribution of queue "xx".

   - ALFAxx.txt represents the initial distribution vector ALFA
     corresponding to T for queue "xx".

   Sample text files for a problem are given below:

   P.txt:
                       0  0.5 0.5
                       1    0   0
                       1    0   0

   ALFA1.txt:
             9.950247518564047e-001   4.975248143595290e-003
   T1.txt:
             -1.990049503712809e+000   0
              0                        -9.950496287190580e-003

   ALFA2.txt:
             1  0

   T2.txt:
             -1  1
              0 -1

   ALFA3.txt:
             1  0

   T3.txt:
             -2  2
              0 -2


   **The text files included here, i.e., within directory 'EXM',
     define a representation for the first example in Marie's paper.

2. Program starts with Main.m.

3. A sample program run:

   >> K = 6;
   >> bv = [6 6 6];
   >> dir = 'EXM'
   >> Main(K,bv,dir)
   >>
   >> 1. CONVOLUTION ALGORITHM.
   >> 2. AKYILDIZ'S MEAN VALUE ANALYSIS (MVABLO).
   >> 3. MARIE'S METHOD USING ML METHOD.
   >> 4. YAO & BUZACOTT'S METHOD USING ML METHOD.
   >> 5. SOLUTION THROUGH ML METHOD.
   >> 6. SOLUTION THROUGH POWER, JOR, OR SOR METHODS.
   >> Enter method number >
```

defines a closed QN which has 3 queueing stations with buffer sizes of 6, and 6 customers. 'EXM' is the directory name and the text files are T1.txt, T2.txt, T3.txt, ALFA1.txt, ALFA2.txt, and ALFA3.txt.

```
>> ...
>> Enter method number (0 (zero) to exit) > 3
>>
>> Cycle type for ML method:
>> 1. V-cycle
>> 2. F-cycle
>> 3. W-cycle
>> Enter your choice of cycle type > 2
```

defines the cycle type for ML method.

```
>> ...
>> Enter your choice of cycle type > 2
>>
>> Smoother type for ML method:
>> 1. Power Method
>> 2. JOR Method
>> 3. SOR method
>> Enter your choice of smoother type > 3
```

defines the iterative method to be used as smoother in the ML method.

```
>> ...
>> Enter your choice of smoother type> 3
>>
>> Enter relaxation parameter w (0<=w<=2) for smoother > 0.5
```

defines the relaxation parameter for JOR and SOR methods.

```
>> ...
>> Enter relaxation parameter w (0<=w<=2) for smoother > 0.5
>>
>> Enter number of pre and post smoothings for this smoother type:
>> pre > 1
>> post > 1
```

defines the number of pre and post smoothings in the ML method.

```
>> ...
>> pre > 1
>> post > 1
>>
>> Enter approximation accuracy > 10e-8
```

defines the solver's stopping tolerance on the approximate error.

```
>> ...
>> Enter approximation accuracy > 10e-8
>>
>> Enter the maximum number of cycles > 100
```

defines the maximum number of cycles that need to be executed.

```
>> ...
```

```
   >> Enter the maximum number of cycles > 100
   >>
   >> File type(s) to save results:
   >> 1. txt
   >> 2. xls
   >> 3. both
   >> Enter your choice of file type(s) for saving results > 3

   defines the file type(s) the program will save the results in.

4. Output:

   Output files can be txt or xls files. They include steady-state
   probability vectors, marginal queue length distributions
   of queueing stations or subnetworks, thruputs of queueing stations,
   and mean queue lengths of queueing stations.
```