28895

MEAN VALUE ALGORITHMS AND HEURISTICS FOR QUEUEING NETWORKS

A Master's Thesis
Presented by
Rifat Aykut ARAPOĞLU

to
the Graduate School of Natural and Applied Sciences
of Middle East Technical University
in Partial Fulfillment for the Degree of

MASTER OF SCIENCE

in

INDUSTRIAL ENGINEERING

MIDDLE EAST TECHNICAL UNIVERSITY
ANKARA
SEPTEMBER, 1993

Approval of the Graduate School of Natural and Applied Sciences.

[illegible]

Prof. Dr. İsmail TOSUN
Director

I certify that this thesis satisfies all the requirements as a thesis for the degree of Master of Science.

[illegible]

Assoc. Prof. Dr. Çağlar GÜVEN
Chairman of the Department

We certify that we have read this thesis and that in our opinion it is fully adequate, in scope and quality, as a thesis for the degree of Master of Science in Industrial Engineering.

[illegible]

Assoc. Prof. Dr. Çağlar GÜVEN
Supervisor

Examining Committee in Charge :

Prof. Dr. Nesim ERKİP [illegible]
Assoc. Prof. Dr. Çağlar GÜVEN [illegible]
Assist. Prof. Dr. Yasemin SERİN [illegible]

To my parents

ABSTRACT

MEAN VALUE ALGORITHMS AND HEURISTICS FOR QUEUEING NETWORKS

ARAPOĞLU, Rifat Aykut
M.S. in Industrial Engineering
Supervisor: Assoc. Prof. Dr. Çağlar GÜVEN
September, 1993, 118 pages.

In this study, Mean Value Analysis approach of product-form queueing networks is analyzed together with its heuristic extensions. Two exact computational algorithms, Mean Value Analysis (MVA) and MVA by Chain (MVAC), which are based on this approach are coded and implemented using the FORTRAN programming language on an IBM 3090 / 180 S mainframe. Each algorithm is implemented using two different strategies; regular implementations and those which exploit sparsity but require more CPU time due to a sequential search. The last implementation of MVAC avoids the need for any search in the case of networks with only three service centers. The algorithms and their implementations are compared in respect of CPU time and storage requirement.

Key Words: Product Form Queueing Networks, Exact Computational Algorithms, Mean Value Analysis.

Science Code: 605.02.02

ÖZ

KUYRUK AĞLARI İÇİN ORTALAMA DEĞER ALGORİTMALARI VE BULGUSAL YÖNTEMLERİ

ARAPOĞLU, Rifat Aykut
Yüksek Lisans Tezi, Endüstri Mühendisliği Anabilim Dalı
Tez Yöneticisi: Doç. Dr. Çağlar GÜVEN
Eylül, 1993, 118 sayfa.

Bu çalışmada, çarpım formundaki kuyruk ağlarında Ortalama Değer Analizi yaklaşımı bulgusal yöntemleri ile birlikte incelenmiştir. Bu yaklaşımı temel alan tam sonuç veren işlemsel algoritmalardan Ortalama Değer (MVA) ve Zincirsel Ortalama Değer (MVAC) algoritmaları IBM 3090 / 180 S ana sisteminde FORTRAN programlama dili kullanılarak yazılmış ve uygulanmıştır. Her algoritma iki farklı strateji kullanılarak programlanmıştır. İlk uygulama programları düz programlar olup ikinciler bilgisayar hafızasını daha etkin kullanan bir çeşit seyrek (sparse) matris tekniği içermektedir. Ancak bu teknik, merkezi işlem süresinin artmasına yol açan bir arama yordamı kullanmaktadır. Zincirsel Ortalama Değer Algoritmasının son uygulaması ise sadece üç hizmet merkezi olan kuyruk ağlarında hiçbir aramaya ihtiyaç göstermemektedir. Algoritmalar ve uygulamaları, merkezi işlem zamanı ve hafıza ihtiyacı açısından karşılaştırılmıştır.

Anahtar Kelimeler: Çarpım Formundaki Kuyruk Ağları, Tam Sonuçlu İşlemsel Algoritmalar, Ortalama Değer Analizi.

Bilim Dalı Sayısal Kodu: 605.02.02

ACKNOWLEDGEMENTS

I would like to express my gratitude to my supervisor, Assoc. Prof. Dr. Çağlar Güven, for his careful supervision and guidance throughout this study.

I owe special thanks to my parents, Gülseren Arapoğlu and Uslu Arapoğlu for their understanding and continuous support during this period. I am also thankful to my brother Rifat and my sister Ayça for their encouragements.

I am grateful to my friends İ. Hakkı Yavuz and Adalet Öner for their kind help and patience in typing the heavy notation. I also thank to my friend Aydın Kaynarca for his immediate helps.

TABLE OF CONTENTS

LIST OF TABLES

Page

LIST OF FIGURES

CHAPTER I
INTRODUCTION

Increase in world population and pervasive use of new technologies give rise to the problem of sharing limited resources among users as efficiently and effectively as possible. Resources may be CPU time for a time sharing computer or communication lines connecting different parts of the world or traffic roads used by vehicles. During the last 30 years, as the efficient use of the resources gained priority, the determination of long-run performance measures associated with expensive systems (such as throughputs, mean queue lengths, mean waiting times and utilizations) became an important issue in the performance evaluation of these systems. Queueing networks are widely used in modelling such systems.

There exist two approaches widely used to analyse queueing networks. The first one is Monte Carlo simulation which is suitable for complex systems that can not be analysed through analytical methods. However, the construction of a simulation model often requires a large effort and the output of the model needs to be carefully analysed to avoid possible misleading results, making this approach costly. The second approach is the analytical approach which is based on the mathematical solution of a set of equations known as 'balance equations' written for the states of the network. The solution of balance equations is not easy and special algorithms have been developed for the exact analysis of only a class of queueing networks called 'product-form' networks.

Product-form queueing networks were first observed by R.R.P. Jackson(1954) on tandem queues and were later extended in the work of J.R.Jackson (1957,1963) to a more general network allowing dependent arrival and service rates. Gordon and Newell (1967) studied closed queueing networks independently. All of these network models assume a single type of customer, first come-first served service centers and exponential service times. Baskett et al. (1975) introduced BCMP type product-form queueing networks relaxing some of the above assumptions. They allowed multiple types of customers and more general service time distributions at some centers while keeping exponential time assumption at others.

Besides the above developments, several exact computational algorithms have also been devised to compute mean performance measures of product-form networks in an efficient manner. The first algorithm was the convolution algorithm due to Buzen (1973) which obtains the mean performance measures in terms of the normalization constants for networks containing a single type of customers. Later Reiser and Kobayashi (1975) extended this algorithm to cover BCMP type networks. Reiser and Lavenberg (1980) proposed a new approach called Mean Value Analysis (MVA) which does not compute normalization constant and therefore avoids problems associated with the computation of the normalization constant. Conway and Georganas (1986) introduced a new algorithm called Recursion by Chain Algorithm (RECAL) which computes the normalization constant using a different recursion over the chains of the network. In 1989, Conway et al. (1989) combined the relative advantages of the MVA and RECAL into a single computational algorithm called the Mean Value Analysis by Chain (MVAC) algorithm. Besides these, there are other exact computational algorithms.

Exact algorithms for product form queueing networks can be classified into two main groups: algorithms which compute normalization constant and those which do not. Örmeci (1993) considered the two normalization constant based algorithms namely, the convolution and the RECAL algorithms. This study deals with the MVA approach which does not compute normalization constant at any time. The purpose of this study, is to examine and evaluate the MVA approach in depth and provide a comprehensive survey of related work including heuristics which derive from it. In this study, we focus on two MVA based exact algorithms namely the MVA and the MVAC algorithms. Different implementations of both algorithms are presented. Each implementation is devised to overcome the difficulties faced with the previous implementation. However, queueing networks that can be exactly analysed form a very small part of product-form networks because of its computational and/or storage requirements. Larger networks without special properties, can only be analysed using approximate algorithms. The Linearizer algorithm is one of the best performing heuristic known in this area but it still needs some further investigation such as its behavior under large number of customer types and determination of the error bounds.

In Chapter II, we present a review of related literature up to the introduction of the BCMP type networks. Mean Value Analysis (MVA) and Mean Value Analysis by Chain (MVAC) algorithms are given together with their theoretical basis. Chapter II ends with a review of approximate MVA algorithm for product-form networks. Chapter III, includes the implementational issues of both algorithms describing the possible strategies of implementation. Finally, the results of the experiments and a comparison of the algorithms are provided in Chapter IV.

CHAPTER II
LITERATURE SURVEY

Queueing network models have been used since the 1960's in diverse areas such as communications networks, computer time-sharing and multiprogramming systems, maintenance and repair facilities, production, air traffic control, assembly and inspection operations. They are important in the performance evaluation of the above systems as they provide the necessary analytic models. The purpose of this dissertation as noted in the Introduction, is to examine and evaluate in depth the Mean Value Analysis (MVA) approach which is one of a class of exact approaches to the steady state analysis of queueing networks. This treatment will also include heuristics based on the MVA algorithm. The survey given in this chapter therefore do not attempt to provide a complete account of all exact approaches but mention others only as far as they stand in relation to the MVA class of algorithms.

Network models were first reported in the 50's and early 60's but they were necessarily of restricted applicability: they depended invariably on the assumption of independent and exponential service time distributions and constant routing probabilities within the network. The last two decades have seen the algebraic solution of what have come to be known as BCMP type queueing networks through efficient computational algorithms. These represent a more general class and therefore the study of more realistic queues is made feasible.

In this chapter, historical developments starting with the work of Jackson (1957) are presented up to the introduction of BCMP type networks for which exact computational algorithms are devised. We then, investigate in detail the MVA approach together with its heuristic extensions.

2.1. Definitions

Buzen (1973) who was probably the first to introduce a practical algorithm for obtaining the limiting distribution of a queueing network defines such a network as follows :

"A Queueing network (QN) is a collection of service facilities organised in such a way that customers must proceed from one facility to another in order to satisfy their service requirements". A more precise definition is given by Kaylan (1990), who defines a QN as a set relation $\langle K, A \rangle$ where K is a collection of nodes and A is the set of arcs connecting these nodes. Each node represents a service system composed of a service mechanism and its associated waiting line. Customer flow takes place on the connecting arcs without any delay.

Queueing networks can be classified into two main groups as open and closed networks as shown in Figure 2.1. An open network allows outside customers to arrive and enter the network as well as departures from the network. Open networks can also be divided into subgroups as feedforward, feedback and overflow networks. In feedforward networks, customers are allowed to visit a node at most once thus, the flow is acyclic or unidirectional. A special case of feedforward networks in which all service centers are connected in series, i.e., in which the output of a center constitutes the input of the next center, is called a tandem network. A feedback network is one in which customers are allowed to visit a node more than once. In overflow networks each node has a single server with zero waiting line capacity. All arrivals occur first to server 1. Those customers that can not receive service at node 1 are lost to that server and they overflow arriving at the next server. The same rule is applied to all remaining nodes.

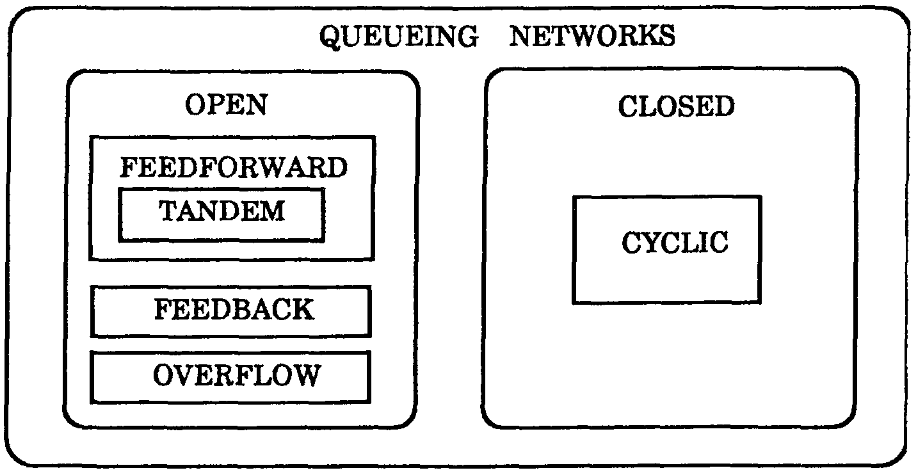


Figure 2.1. Classification of Queueing Networks with respect to the Routing Process

In a closed network no external customer arrivals or departures are permitted; thus the total number of customers in a closed network is constant. A cyclic network is basically a tandem network in which the outflow of the last service center is directed to the first center forming a closed network.

Another classification for QN's is possible with respect to the waiting line capacities or buffer size (restricted or unrestricted) and the total number of customers in the network (constant or variable). Hence we can have QN's which are

Open-unrestricted ,
Closed-unrestricted ,

Open-restricted , or
Closed-restricted.

2.2. Jackson Networks

After R.R.P. Jackson studied a tandem queue in 1954, J.R. Jackson (1957) first described an open queueing network in a systematic way. The type of network which is known as a Jackson Network consists of

M service centers each having s_i parallel servers and a waiting room of unlimited capacity. Customers are indistinguishable and arrive at a service center either from outside or from another service center in the network. Arrival times of customers from outside to service center i are Poisson random variables with mean rates λ_{oi}. Service time distributions are independent of arrivals and customers are served on a First Come First Served (FCFS) basis. All service times are exponentially distributed with mean rate μ_i. After each service completion at service center i, customers immediately join queue j with probability p_{ij} or leave the network with probability p_{io}, hence $p_{io} = 1 - \sum_{j=1}^{N} p_{ij}$. These probabilities are known as routing probabilities and are assumed to depend, independently of the state of the network, only on the center that the customer is leaving.

The equilibrium flow rate of customers through node i in the Jackson network described above can be found by solving the following system of linear equations :

$$e_i = \lambda_{oi} + \sum_{k=1}^{M} e_k \, p_{ki} \qquad i = 1,\dots,M. \qquad (2.1)$$

where e_i denotes the equilibrium flow rate of customers at node i from any other node, inside or outside of the network. e_i's are often known as the throughput rate.

A closer look at the Jackson network reveals that it looks like consisting of M individual M / M / s queueing systems except that the arrival process to a service center is not Markovian. This is due to the presence of feedback arrivals at the centers.It is shown in Burke (1956) that the output process of an M / M / s queue in equilibrium is Poisson with the same rate as the input process. However in the case of networks of queues service completion times at a center and the feedback arrival times to that center become dependent which implies that the arrival process is not Poisson. Still, we see later in this chapter that the centers behave as if the input process were Poisson. This is the most remarkable property of a Jackson network.

The state of an M / M / s system is a random process P, $P = \{n(t)\ ,\ t \geq 0\}$ where n(t) denotes the number of customers waiting for or receiving service at time t. Because of the memoryless property of the arrival and service processes, the state of the system can be represented as a Markov process and then, the state of the queueing network can be expressed in terms of the states of the individual M / M / s queueing systems as follows: Assuming that there are M service centers, let $N = \{N_t = (\ n_1(t), n_2(t), \ldots, n_M(t)\ )\ ,\ t \geq 0\}$ be a random process where $n_i(t)$ is the number of customers waiting for or receiving service at node i at time t. Let $\mathbf{n} = (n_1, n_2, \ldots, n_M)$ be any state and $p_t(\mathbf{n}) = P\ \{\ N_t = \mathbf{n}\ \}$ be the state probability at time t for any state $\mathbf{n}\ \varepsilon N$.

Jackson obtained the joint steady-state probability distribution $p(\mathbf{n})$ of state $\mathbf{n}$ in the form of a product of functions which depend on the traffic intensity at center i. The intensity ρ_i is defined as $\rho_i = e_i / \mu_i$, with $\rho_i / s_i < 1$ where s_i is the number of servers at center i.

$$p(n_1, n_2, \ , n_M) = p_1(n_1)\ p_2(n_2) \ldots p_M(n_M) \tag{2.2}$$

where

$$p_i(k) = \begin{cases} p_i(0)\ \rho_i^k / k! & k = 0, 1, \ldots, s_i \\ p_i(0)\ \rho_i^k / (s_i!\ s_i^{k-s_i}\ ) & k = s_{i+1}, \ldots \end{cases} \tag{2.3}$$

and $p_i(0)$ must be obtained from the normalizing equation :

$$\sum_{k=0}^{\infty} p_i(k) = 1 \tag{2.4}$$

The proof of the above result was given by Jackson (1957) who showed by direct substitution that the product form solution satisfied the balance equations written for each service center of the network.

Jackson's result says that the individual M / M / s queues behave independently i.e., the number of customers at each center is independent of the queue lengths at the other centers, and that they act as if their arrival processes were Poisson with mean rate e_i.

Jackson later (1963) extended this model by allowing :

1. the mean arrival rate of outside customers to be dependent on the number of customers already present in the network.

2. the mean service rate of a server at a service center to be dependent on the number of customers present in that service center.

These extensions include "triggered arrivals" whereby customers are injected automatically into the network when the total population of the network falls below a prespecified limit, and service deletions or jumps, where a service is deleted when the queue-length at a service center exceeds a prespecified maximum.

2.3. Gordon - Newell Networks

Gordon and Newell without being aware of the results of Jackson, studied a closed Markovian network with N identical customers (fixed) circulating over M interconnected service centers. This corresponds to Jackson's model with $\lambda_{oi}=0$ and $p_{io}=0$ for all i. They solved the equilibrium equations for the joint probability distribution of customers by using a separation of variables technique and obtained a simpler notation for the equilibrium joint distribution of customers. The state space, S (N,M)

$$S(N,M) = \{(n_1,n_2,...,n_M) : \sum_{i=1}^{M} n_i = N , n_i \geq 0 \quad i=1,...,M. \} \tag{2.5}$$

is finite but clearly very large even for small values of N and M. The number of distinct states of a closed network with M service centers and N customers is given by C(N+M-1,M-1),where C(a,b) denotes the number of distinct combinations of a with b. The equilibrium flow rate of customers in a Gordon-Newell network can be found easily by dropping λ_{oi}'s from equation (2.1) i.e.,

$$e_i = \sum_{k=1}^{M} e_k \; p_{ki} \qquad i = 1,2,\ldots,M. \qquad (2.6)$$

The unique steady-state distribution of customers in the network was obtained by Gordon and Newell (1967) as :

$$p(n_1,n_2,\ldots,n_M) = \frac{1}{G(N)} \prod_{i=1}^{M} \alpha_i(n_i) \qquad (2.7)$$

where

$$\alpha_i(n_i) = \frac{e_i^{n_i}}{\prod_{j=1}^{n_i} \mu_i \min(j,s_i)}$$

G(N) is the normalization constant and defined as:

$$G(N) = \sum_{\mathbf{n} \,\varepsilon S\,(N,M)} \prod_{i=1}^{M} \alpha_i(n_i) \qquad (2.8)$$

Note that the computation of G(N) is not straightforward and requires a summation of product terms over the entire state space S(N,M). Although there has been attempts to derive simpler closed form expressions for the normalization constants (e.g.,Koenigsberg, 1958), this difficulty still goes on and led to the development of special algorithms devised to compute the normalization constants efficiently. On the other hand, open networks are easier to solve when compared to a corresponding closed network but they require more information than a closed network; i.e., outside arrival rates, λ_{oi}, are either known or estimated.

2.4. BCMP Networks

Baskett et al. (1975) introduced a more general class of QN (called BCMP networks after the initials of the authors) allowing considerable relaxations on the assumptions of the original Jackson network. The relaxations come in three categories :

i.Different queueing disciplines are allowed at different service centers in the same network.

ii.More general (Coxian) service time distributions are recognised for the non-FCFS service centers.

iii.The single chain framework of the Jackson networks was extended to multi-chain networks allowing different types of customers.

These extensions now are discussed in detail. The following disciplines can be handled :

Type 1 : All customers are served in order of their arrival to the service center and there is no restriction on the number of customers. This service discipline is referred to as FCFS.

Type 2 : Processor Sharing (PS) : All customers receive service simultaneously at a rate of 1/nth of the service rate when there are n customers at the service center.

Type 3 : Infinite Server (IS) or Server per Customer : Servers at a service center are identical and work in parallel. The number of servers at this type of service center is at least the maximum number of customers allowed at this center at any time. Thus, customers begin to receive full service as soon as they arrive at this type of service center and no queueing delay occurs.

Type 4 : Last Come First Served Preemptive-Resume (LCFS-PR) : The last arrival is served first. Upon last arrival, the customer receiving service (if there is any) is interrupted and the remaining part of service is resumed after the departure of the last customer (which also may be interrupted by another arrival during its service interval).

Coxian Service Time Distributions

Service times at service centers are represented by the response time (i.e. total time a job spends in the network) of a feedforward network consisting of only exponential subservers (stages) with different service rates as shown in Figure 2.2. Only one job is allowed in the network at a time.

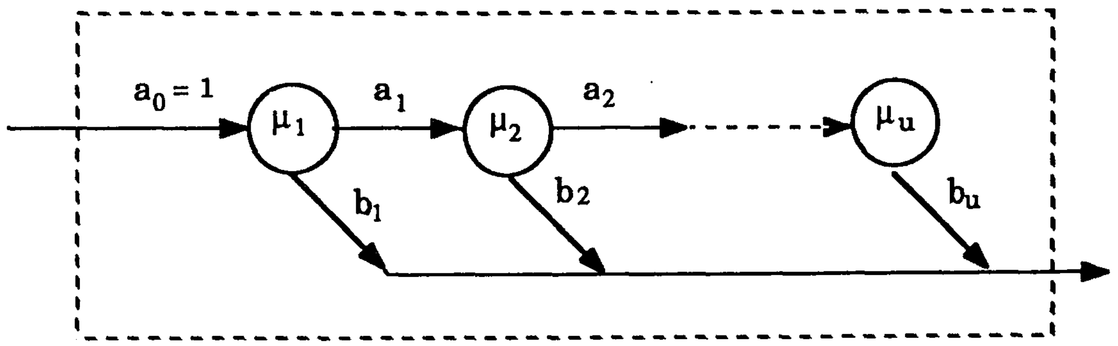


Figure 2.2. Representation of Coxian Service Time Distribution

Let τ denote the average response time. Its expected value is :

$$E[\tau] = \sum_{l=1}^{u} \left(A_l b_l \sum_{i=1}^{l} (1/\mu_i) \right) = \sum_{i=1}^{u} (A_i/\mu_i) \qquad (2.9)$$

where $A_i = a_0 a_1 .. a_{i-1}$

u is the number of stages (subservers)

a_i is the probability that a job proceeds to subserver i+1 given that it receives service at subserver i.

$b_i = 1 - a_i \qquad i=1,..,u.$

The Laplace transform of the p.d.f. of τ, $f^*(s)$ is given as :

$$f^*(s) = \sum_{l=1}^{u} A_l b_l \left[\prod_{i=1}^{l} (\mu_i/(\mu_i+s)) \right] \qquad (2.10)$$

which is a rational function of s and can be written as a ratio of two polynomials P(s) / Q(s). Thus, any distribution whose Laplace transform has a rational functional form can be represented by a Coxian distribution. Furthermore, any distribution function (without the rational Laplace transform) can be approximately represented by a Coxian distribution (Gelenbe and Mitrani, 1980).

BCMP Network Model

A BCMP network is represented by a graph with M nodes (representing service centers). There are R classes of customers and each customer belongs to a single class at a time but it is permitted for a customer to shift to another class after a service completion at a service center with some fixed probability namely, $p_{ir,sj}$, the probability that a customer of class r at center i moves to center j as a class s customer after a service completion at center i. This class switching feature of BCMP networks allows different routing parameters and/or service time requirements to a class r customer at different visits to the same service center, thus brings flexibility in modelling real life situations as queueing network models. A class r customer at node i leaves the network with probability $p_{ir,o} = 1 - \sum_{j=1}^{M} \sum_{s=1}^{R} p_{ir,js}$ $i=1,..,M.$ $r=1,..,R.$ The pair (i,r) can be considered as a customer state. Then, the set of customer states can be split into m ($m \geq 1$) non-communicating subsets of customer states, since it may be impossible for a customer at state (i,r) to visit other customer states and vice versa even though it is permitted to change classes. In this way, it is possible to form m separate subchains denoted as $E_1,..,E_m$. (For example, if no class switching is permitted then there will be at least R subchains ($m \geq R$)).

In a BCMP network some subchains may be closed having a fixed number of customers in the network (no external arrival or departure) whereas others are open allowing external arrivals and departures. The external arrival process may be generated in two ways:

(i) The external arrival process is Poisson depending on the total number of customers in the network N with mean rate $\lambda(N)$. The probability of a new arrival to node i as a class r customer is $p_{o,ir}$ with $\sum_{i=1}^{M} \sum_{r=1}^{R} p_{o,ir} = 1.$

(ii) There are m independent Poisson processes one for each subchain. The mean arrival rate of the kth process depends on the number of customers at subchain k, N_k, and denoted by $\lambda_k(N_k)$. The probability of a new arrival from the kth process to node i as a class r customer is $p_{o,ir}$ with $\sum_{(i,r)\epsilon E_k} p_{o,ir} = 1 \quad k=1,\ldots,m$.

When Coxian service time distributions are used, the process *N* is not Markovian. However, the Markov property can be restored by redefining the network state by specifying also the stage at which the service is. In this way, the network states form a Markov chain because all the servers are now exponential. These states are represented by a vector $\mathbf{x} = (\mathbf{x}_1, \mathbf{x}_2,\ldots,\mathbf{x}_M)$ whose component $\mathbf{x}_i$ is also a vector denoting the state at center i. Baskett et al. (1975) described these states for each service center (discipline) as follows:

If service center i is of type 1 (FCFS), then $\mathbf{x}_i = (x_{i1},x_{i2},\ldots,x_{in_i})$ where n_i is the number of customers at center i and x_{ij} is the class of customers who is in jth position in the queue.

If service center i is of type 2 or 3 (PS or IS), then $\mathbf{x}_i = (\mathbf{v}_{i1},\mathbf{v}_{i2},\ldots,\mathbf{v}_{iR})$ with $\mathbf{v}_{ir} = (m_{1r}, m_{2r},\ldots, m_{u_{ir}r})$ where u_{ir} is the number of stages for a class r customer at center i and m_{lr} is the number of class r customers at center i in the lth stage of service.

If service center i is of type 4 (LCFS-PR), then $\mathbf{x}_i = ((r_1,m_1), (r_2,m_2),\ldots,(r_{n_i},m_{n_i}))$ where a pair is written for each customer at center i denoting the class of the customer, r_j and the current stage of service, m_j.

The solution of the global balance equations written for the equilibrium state probabilities is a tedius work. Instead, they wrote the independent balance equations (or local balance equations) which equate the flow of customer between two adjacent states of the network. The solution of the independent balance equations are given as the BCMP theorem.

The customer flow balance equations can be written for each ergodic subchain E_k, k=1,..,m. as follows :

$$e_{ir} = p_{o,ir} + \sum_{(j,s)\varepsilon E_k} e_{js}\, p_{js,ir} = 1 \qquad (i,r)\varepsilon E_k \qquad (2.11)$$

If $p_{o,ir}$ is zero for all $(i,r)\varepsilon E_k$ then the subchain E_k is closed. In this case e_{ir} values are determined to within a multiplicative constant and can be interpreted as relative arrival rate of class r customers to center i. If not all $p_{o,ir}$'s are zero for all $(i,r)\varepsilon E_k$, then the subchain E_k is open and it is assumed that the flow balance equations have a unique solution and e_{ir} values can be interpreted as absolute arrival rate of class r customers to center i.

BCMP Theorem

The solution for a network which can be open, cclosed or mixed in which each service center can be one of the FCFS, PS, IS, or LCFS-PR disciplines is given by :

$$P(S = (\mathbf{x}_1,\mathbf{x}_2,....,\mathbf{x}_M)) = G \; d(N) \; f_1(\mathbf{x}_1) \; f_2(\mathbf{x}_2)....f_M(\mathbf{x}_M) \qquad (2.12)$$

where

G is the normalization constant, d(N) is a function of the number of customers in the system and each f_i is a function that depends on the service discipline of center i.

If center i is FCFS then

$$f_i(\mathbf{x}_i) = (1/\mu_i)^{n_i} \prod_{j=1}^{n_i} e_{ix_{ij}}$$

If center i is PS then

$$f_i(\mathbf{x}_i) = n_i! \prod_{r=1}^{R} \prod_{l=1}^{u_{ir}} \{ (e_{ir}\; A_{irl} / \mu_{irl})^{m_{irl}} \; (1/ m_{irl}!) \}$$

If center i is IS then,

$$f_i(\mathbf{x}_i) = \prod_{r=1}^{R} \prod_{l=1}^{u_{ir}} \{(e_{ir}\ A_{irl}/\mu_{irl})^{m_{irl}}\ (1/m_{irl}!)\}$$

If center is LCFS-PR then

$$f_i(\mathbf{x}_i) = \prod_{j=1}^{n_i} \{\ e_{ir_j}\ A_{ir_jm_j}\ (1/m_{ir_jm_j})\}$$

where $A_{irl} = \prod_{j=1}^{l} a_{irj}$, a_{irj} is the probability of going to stage j+1 after the service completion at stage j for a class r customer at center i and m_{irl} is the number of class r customers at center i in stage l.

If the first case of external arrival process is adopted, in which the arrival rate depends on the total number of customers in the network, N,

$$d(N) = \prod_{i=0}^{N-1} \lambda(i) \qquad (2.13)$$

If the arrivals to the system are according to the second case, in which the arrival rate depends on the number of customers in subchain k, N_k, then

$$d(N) = \prod_{k=1}^{m} \prod_{i=0}^{N_k-1} \lambda_k(i) \qquad (2.14)$$

d (N) = 1 if the network is closed.

The theorem is proved by substituting the solution into the independent balance equations.

2.5. Analysis of Product Form Networks

When usual Markovian assumptions are made, exact analysis of closed queueing networks is impractical and too costly due to the large size of the state space. Sauer and Chandy (1981) give an example of such a network solution by solving balance equations. This problem was partially overcome by Jackson (1963) who showed that for a certain class of queueing networks the joint queue length distribution appears in the form of a product of terms each corresponding to a service center in an open queueing network with exponential service time distributions. This special class of queueing networks is known as 'product form networks'. Gordon and Newell (1967) showed that closed queueing networks with load-dependent exponential service times, assume a product form solution, assuming as Jackson FCFS queueing discipline at every center of the network. Since then, a number of researchers (Ferdinand,1971; Posner and Bernholtz, 1968; Chandy, 1972) have attempted to relax these assumptions. Chandy (1972) showed that networks with PS and LCFSPR queueing discipline have product form solutions and satisfy local balance equations. Local balance equations state that the rate of transition out of state **n** due to a customer departure from center i is equal to the rate of transitions into state **n** due to a customer arrival into center i. Baskett et al.(1975) unified the above studies and extended them to multiple classes of customers for a queueing network consisting of FCFS, LCFSPR, PS, IS type service centers provided that service time distributions have a rational Laplace transform at non-FCFS centers. Muntz (1972) investigated the so called M --> M property which reads as Poisson arrivals imply Poisson departures. He showed that a network of queues with this property has a product form solution. Chandy et al.(1977) introduced a new property called ' station balance ' as an explanation of the product form solutions at non exponential centers. They showed that if a state probability density function satisfies station balance, then it satisfies both local balance and global balance and that a queueing discipline which satisfies station balance must begin to serve new customers immediately. They also note that FCFS and fixed priority disciplines do not satisfy the station balance since they fail to satisfy the immediate service criterion. The findings of Chandy et al.(1977) has made a significant contribution to the characterization of product form networks. In general, product form property of a service center depends on both the queueing discipline and the service time distribution employed at that center. However there exist some special cases which are identified in the literature:

i) If the queueing discipline at a center satisfies station balance then a product form solution is obtained independently of the service time distribution depending only on the mean service times. This independence is known as the ' insensitivity property ' of product form networks. It should be noted that this case is an extension of BCMP queueing networks with PS, IS, LCFSPR type (all of which satisfy station balance) service centers and allow arbitrary service time distributions. Due to this important property, exact algorithms devised for closed product form queueing networks assume service centers whose queueing discipline satisfy station balance (e.g. PS, IS, LCFSPR) and only the mean service times are needed for such centers. Noetzel (1979) described a generalized queueing discipline called Last Batch Processor Sharing (LBPS) covering all disciplines which yield station balance and therefore, product form solution for any service time distribution. In this discipline, the set of customers receiving service at once is called a batch. All customers in a batch are served according to the PS discipline and there is a maximum batch size. If a customer arrives when the batch is at its maximum size, the batch is pre-empted and the arriving customer is given full service. The batches are served according to the LCFSPR discipline. It can be noted that this discipline reduces to PS if maximum batch size is relatively large or infinite and to LCFSPR if maximum batch size is one.

ii) If the service times of customers at a service center are exponentially distributed with the same mean for all chains then again a product form solution is obtained for all work conserving queueing disciplines even if station balance is not satisfied. Therefore, station balance is sufficient but not necessary for product form solutions. Exponential service times for FCFS centers of BCMP queueing networks is included in this category as well as other disciplines which do not satisfy station balance (e.g. random selection service discipline (Spirn, 1979)). There are other implications of these cases: For example, any center satisfying product form with non-exponential service times (including exponential distributions with different mean values for different chains) implies that steady state product form probabilities must satisfy station balance.

Algorithms for Product Form Networks

To obtain a product form solution for closed queueing networks, it is necessary to calculate the normalization constant. This normalization requires the computation of the sum of the product terms over the state space. The Convolution Algorithm of Buzen (1973) was the first to compute the normalization constant efficiently. For practical purposes, however, the joint distribution contains far too much detail. Much simpler quantities such as mean queue sizes, mean waiting times, utilizations and throughputs are needed as performance measures. Within the framework of the convolution algorithm, it has also been shown that such quantities can be derived from the normalization constants in an efficient manner. However, such a procedure suffers from computational problems; the normalization constant may have a value outside of the floating point range of many computers even though the network parameters and performance measures have values well within the floating point range. This problem can usually, but not always, be eliminated by an appropriate choice of the relative throughput rates (Lavenberg, 1983). The convolution approach also suffers from numerical instability near the lower limit of the floating point range. Reiser and Kobayashi (1975) extended this algorithm to allow multiple chain queueing networks.

Reiser and Lavenberg (1980) proposed an alternative way of solving a product form queueing network called Mean Value Analysis (MVA). The recursion in MVA is in terms of mean performance measures and there is no need to compute a normalization constant. The MVA recursion relates mean performance measures of a network of population

N to those of a network with population N-1. In this respect, MVA considers the network population customer by customer.

The Local Balance Algorithm for Normalizing Constant (LBANC) is another exact algorithm developed by Chandy and Sauer (1980). This algorithm resembles both the Convolution and the MVA algorithms in that it has the same type of recursion as MVA with unnormalized performance measures. Meanwhile, normalization constants are computed in parallel and normalization of performance measures takes place at the end of the algorithm. In LBANC, it is possible to reduce the storage requirement by saving the normalization constants corresponding to intermediate population levels. This feature of LBANC constitutes its advantage over MVA but it still has problems associated with the normalization constant.

The Recursion by Chain Algorithm (RECAL) (Conway and Georganas, 1986) is the first of the three algorithms (others are MVAC and DAC) developed essentially for multiple chain product form queueing networks. The recursion in RECAL can be considered as a chain by chain approach ; it starts with no chain in the network and adds one chain at a time until all chains are introduced. The recursion in RECAL aims to compute the normalization constants of these intermediate networks (which are named as related networks).

The Mean Value Analysis by Chain algorithm (MVAC) (Conway et al.,1989) is similar to RECAL in terms of the chain by chain decomposition, but its recursion involves mean performance measures like that of MVA.

The Distribution Analysis by Chain algorithm (DAC) (Silva and Lavenberg,1989) is devised especially for the computation of the joint queue length distribution as the name implies. Its recursion is in terms of the joint queue length probabilities associated with the related networks and the mean performance measures can be obtained by a supplementary algorithm.

Besides the above general algorithms, there exist some special algorithms developed for solving large queueing networks with many routing chains and centers if the network has sparseness property. A queueing network where most of the routing chains visit only a small subset of the service centers, is said to have ' sparseness ' property. These algorithms include the Tree Convolution algorithm of Lam and Lien (1983), the Tree MVA algorithm of Tucci and Sauer (1985) and Hoyme et al. (1986), the Tree RECAL algorithm of Greenberg and McKenna (1989). These special algorithms are especially useful in solving communication network models; in such network models each origin-destination pair is often represented by a separate chain. This generates a large number of chains but they have usually the sparseness property.

2.6. Mean Value Analysis Algorithm

The Mean Value Analysis (MVA) algorithm introduced by Reiser and Lavenberg (1980) works directly with the desired statistics. It is mainly based on the Arrival Theorem (Sevcik & Mitrani,1981 and Lavenberg&Reiser,1980) and Little's formula (Little,1961).

2.6.1. Single Chain MVA

Consider initially a closed single chain queueing network with product-form solution. Let Q(N) denote such a network with N customers and M service centers. The Arrival Theorem for single chain networks can be stated verbally as : In a product form queueing network an arriving customer at center i in Q(N) observes the network Q(N-1) in equilibrium. The importance of this theorem comes from the fact that it allows to establish a relation between networks Q(N) and Q(N-1) which gives rise to the MVA algorithm. The form of the Arrival Theorem suggests that this relation has a recursive nature which is to be solved iteratively by applying the two results successively at each population level from one to N.

In this section the MVA analysis is described and developed. To start, we define the following parameters for Q(N) in steady-state :

$\theta_i = E[v_i]$: Expected value of v_i, the number of visits a customer makes to service center i between successive visits to i*, an arbitrarily chosen queue

$\tau_i = E[s_i]$: Expected value of s_i, the service demand brought into queue i at a given arrival epoch (measured in number of instructions to be executed, for example).

$\mu_i(k)$: The service rate when there are k customers present at queue i (measured in number of instructions executed per second, for example). This is also referred to as the ' capacity function ' and it will be set equal to one for all k if the service rate is fixed.

Assume the routing matrix P to be irreducible. From standard Markov chain analysis θ_i's satisfy the equation

$\theta = \theta P$ where $\theta = (\theta_1, \ldots ,\theta_M)$ and P is a (M by M) routing matrix.

Note that θ satisfying the above equation can be chosen in infinitely many different ways since P is stochastic (row sums equal 1). We can choose one θ setting $\theta_{i^*} = 1$ where i* is an arbitrarily selected queue (this queue is also named as the marked center). We know from Basket et al. (1975) that additional assumptions must be made to have a product-form solution : service times s_i are assumed i.i.d. random variables from a Coxian distribution if the queueing discipline is PS, IS, LCFS-PR and exponential for FCFS queues. To proceed further let us introduce the following notation :

k_i : Number of customers at queue i at a given instant in time (including the one being served)

$\mathbf{k} = (k_1, \ldots ,k_M)$: State vector of Q(N)

$n_i = E[k_i]$: Mean queue length at center i

w_i : Mean waiting time per visit of a customer at queue i (including service time)

$\lambda = \lambda_{i^*}$: Throughput rate of marked queue i*

$\lambda_i = \theta_i \lambda$: Throughput rate of queue i

$\rho_i = \theta_i \tau_i$: Mean service demand brought into queue i by a customer between successive visits to the marked center i*

$W_i = \theta_i w_i$: Mean waiting time in queue i between successive visits to the marked center i*

g(N) : Normalization constant of the network Q(N)

$g^{[i]}(N-k_i)$: Normalization constant of the network Q(N) with center i removed

$p_i(k,N) = P\{ k_i = k \mid N \}$: Steady state marginal probability of k customers at center i of Q(N)

Here the subscript i is used to denote any queue in the set of nodes {1,..,M}

Before proceeding further it is convenient to show the following relations written for the network Q(N) :

$$p_i(k,N) = \pi_i(k) \frac{g^{[i]}(N-k)}{g(N)} \qquad k = 0,1,\dots,N. \qquad (2.15)$$

$$\lambda(N) = \lambda_{i^*}(N) = \frac{g(N-1)}{g(N)} \qquad (2.16)$$

$$p_i(k,N) = \frac{\rho_i \ \lambda(N)}{\mu_i(k)} \ p_i(k-1,N-1) \quad k = 1,\dots,N. \qquad (2.17)$$

The key result here is (2.17) and the relations (2.15), (2.16) are used to derive this result. In the following we provide the derivations of the relations (2.15)-(2.17).

i) In a product form network, the probability of state **k** is given by :

$$P\{ \mathbf{k} = (k_1,\dots,k_M) \} = \pi_1(k_1)\, \pi_2(k_2) \dots \pi_M(k_M) / g(N)$$

where : $\pi_i(k) = \dfrac{\rho_i^{\,k}}{\mu_i(1)\dots\mu_i(k)}$ and $\mu_i(0) = 1$, i=1,...,M.

$$p_i(k,N) = \sum_{\mathbf{k}:\sum_{j\neq i} k_j=N-k} P\{\mathbf{k}=(k_1,\dots,k_{i-1},k,k_{i+1},\dots,k_M)\}$$

$$= \frac{\pi_i(k)}{g(N)} \sum_{\sum_{j\neq i} k_j=N-k} \prod_{l\neq i} \pi_l(k_l)$$

$$= \pi_i(k) \frac{g^{[i]}(N-k)}{g(N)}$$

ii) From the definition of the expected value of throughput :

$$\lambda_i(N) = \sum_{k=1}^{N} p_i(k,N)\, \mu_i(k) / \tau_i$$

using equation (2.15),

$$= \frac{1}{\tau_i} \sum_{k=1}^{N} \pi_i(k) \frac{g^{[i]}(N-k)}{g(N)} \mu_i(k)$$

$$= \frac{\theta_i}{g(N)} \sum_{k=1}^{N} \frac{\rho_i^{k-1}}{\mu_i(1)\dots\mu_i(k-1)} g^{[i]}(N-k)$$

$$= \frac{\theta_i}{g(N)} \sum_{k=1}^{N} \pi_i(k-1)\, g^{[i]}(N-k)$$

$$= \theta_i \frac{g(N-1)}{g(N)}$$

iii) Using equations (2.15) and (2.16),

$$p_i(k,N) = \pi_i(k) \frac{g^{[i]}(N-k)}{g(N)} = \frac{\rho_i^{k}}{\mu_i(1)\dots\mu_i(k)} \frac{g^{[i]}(N-k)}{g(N)}$$

$$= \frac{\rho_i}{\mu_i(k)}\ \pi_i(k-1)\ \frac{g^{[i]}(N-k)}{g(N)} = \frac{\rho_i}{\mu_i(k)}\ \pi_i(k-1)\ \frac{g^{[i]}(N-1-(k-1))}{g(N-1)}\ \frac{g(N-1)}{g(N)}$$

$$= \frac{\rho_i}{\mu_i(k)}\ p_i(k-1,N-1)\ \lambda(N)$$

$$= \frac{\tau_i}{\mu_i(k)}\ \lambda_i(N)\ p_i(k-1,N-1)$$

This last equation was first shown in Reiser and Lavenberg(1980) which forms a recurrence relation between the marginal probability distribution of queue i in Q(N) and marginal probability distribution of the same queue in Q(N-1) (i.e. same system as Q(N) with one customer less).If Q(N) consists of only single server fixed rate (SSFR) queues having $\mu_i=1$ then the mean waiting time of an arriving customer at queue i,w_i can be written as :

$$w_i(N) = \tau_i + \tau_i \text{ \{ mean queue size at arrival epochs \}} \quad (2.18)$$

The quantity within brackets can be replaced by $n_i(N-1)$ by Arrival Theorem. Thus, equation (2.18) becomes now :

$$w_i(N) = \tau_i + \tau_i\, n_i(N-1) = \tau_i\,(\,1+n_i(N-1)\,) \quad (2.19)$$

Let us look at the queueing network Q(N) from the marked center i*. The average number of customers in the network is N (fixed) and the mean time a customer spends in the network between successive visits to i* (i.e. mean cycle time) is $\sum_{i=1}^{M} \theta_i w_i(N)$. Thus, Little's equation when applied to the entire network viewed from i* gives :

$$\lambda(N) = \frac{N}{\sum_{i=1}^{M} \theta_i w_i(N)} \quad (2.20)$$

Little's equation applied to each service center separately yields :

$$n_i(N) = \lambda_i(N)\, w_i(N) \qquad i = 1,\dots,M \tag{2.21}$$

Infinite server (IS) type service centers can be considered as a special case of (2.19) since $n_i(N-1)$ always equals zero at such centers. Then, substituting the quantities $W_i(N) = \theta_i w_i(N)$, $\rho_i = \theta_i \tau_i$ into equations (2.15)-(2.17) we obtain the following set of equations which forms the heart of MVA recursion :

$$W_i(N) = \rho_i\,(1 + \delta_i\, n_i(N-1)) \tag{2.22}$$

$$\lambda(N) = \frac{N}{\sum_{i=1}^{M} W_i(N)} \tag{2.23}$$

$$n_i(N) = \lambda(N)\, W_i(N) \tag{2.24}$$

where $\delta_i = \begin{cases} 0 \text{ if i is a IS type service center} \\ 1 \text{ if i is a FCFS type service center} \end{cases}$

The equations (2.22)-(2.24) provide the basis for an iterative evaluation of the performance measures. The iteration starts for each queue, with the initial conditions $n_i(0) = 0$ and proceeds until the desired population level N is reached by adding a single customer into the network at a time. Note that all three performance measures (mean queue sizes,mean waiting times, throughputs) are calculated in parallel. The utilization can be computed easily at the end of the iteration using

$$u_i(N) = \lambda_i(N)\, \tau_i \tag{2.25}$$

Note that all the above equations deal with mean performance values and there is no use of normalization constants or marginal distributions.

When load-dependent service centers are allowed, the mean waiting time is no longer determined by what is encountered by an arriving customer since later arrivals may change the service rate. It is therefore necessary to calculate marginal probabilities at these centers in addition to mean values. Using Little's formula, we may write for the expected waiting time of a customer at a load-dependent service center i :

$$w_i(N) = \frac{n_i(N)}{\lambda_i(N)} = \sum_{j=1}^{N} \frac{j\, p_i(j,N)}{\lambda_i(N)}$$

$$= \sum_{j=1}^{N} \frac{j\, \tau_i\, p_i(j-1,N-1)}{\mu_i(j)}$$

using (2.17) the mean value equations now become :

$$W_i(N) = \begin{cases} \rho_i\,(\,1 + \delta_i\, n_i(N-1)\,) & \text{if center i is IS or SSFR} \\ \rho_i \sum_{j=1}^{N} \frac{j\, p_i(j-1,N-1)}{\mu_i(j)} & \text{if center i is load-dependent} \end{cases} \qquad (2.26)$$

$$\lambda(N) = \frac{N}{\sum_{i=1}^{M} W_i(N)}$$

$$p_i(j,N) = \lambda(N)\, \frac{\rho_i}{\mu_i(j)}\, p_i(j-1,N-1) \quad j=1,\ldots,N.$$

$$p_i(0,N) = 1 - \sum_{j=1}^{N} p_i(j,N) \qquad (2.27)$$

$$n_i(N) = \lambda(N)\, W_i(N)$$

Here, the iteration starts with the initial conditions $p_i(0,0) = 1$ and $n_i(0)=0$ $i=1,\ldots,M$.

We illustrate the recursion using a small example :

Example 2.1.

Consider a closed product form queueing network with three service centers (M=3) and three customers (N=3). Assume that the first center is of type IS, the second one is a SSFR and the third one is a load dependent center. The service rate at center i when there are k customers, $\mu_i(k)$ is as given in Table 2.1.

Table 2.1. Service rates at center i with k customers

$\mu_i(k)$	k = 1	2	3
i=1	1	2	3
2	1	1	1
3	1	3	5

The probability transition matrix is given as :

$$P = \begin{Bmatrix} 1/5 & 1/5 & 3/5 \\ 0 & 3/5 & 2/5 \\ 3/5 & 1/5 & 1/5 \end{Bmatrix}$$

Choosing arbitrarily the first center as a marked i.e. $i^*=1$ and solving $\theta=\theta P$ by setting $\theta_1=1$; we get :

$\theta = (1, 7/6, 4/3)$

Assume that customers have the following service requirements at center i :

$\tau = (2, 3, 6)$

Calculations proceed through the following steps :

Initialization :

$n_i(0) = 0 \qquad i=1,2,3.$

$p_3(0,0) = 1$

Main Loop :

Population Level : N = 1

$w_1(1) = 2$ $\qquad$ $n_1(1) = 4/27$

$w_2(1) = 3$ $\qquad$ $\lambda(1) = 2/27$ $\qquad$ $n_2(1) = 7/27$

$w_3(1) = 6$ $\qquad$ $n_3(1) = 16/27$

$p_3(1,1) = 16/27$

$p_3(0,1) = 1 - p_3(1,1) = 11/27$

Population Level : N = 2

$w_1(2) = 2$ $\qquad$ $n_1(2) = 324/1039$

$w_2(2) = 34/9$ $\qquad$ $\lambda(2) = 162/1039$ $\qquad$ $n_2(2) = 714/1039$

$w_3(2) = 130/27$ $\qquad$ $n_3(2) = 1040/1039$

$p_3(1,2) = 528/1039$

$p_3(2,2) = 256/1039$

$p_3(0,2) = 1 - p_3(1,2) - p_3(2,2) = 255/1039$

Population Level : N = 3

$w_1(3) = 2$ $\qquad$ $n_1(3) = 0.4360$

$w_2(3) = 5.0616$ $\qquad$ $\lambda(3) = 0.2180$ $\qquad$ $n_2(3) = 1.2873$

$w_3(3) = 4.3923$ $\qquad$ $n_3(3) = 1.2767$

$p_3(1,3) = 0.4280$

$p_3(2,3) = 0.2954$

$p_3(3,3) = 0.0859$

$p_3(0,3) = 1 - p_3(1,2) - p_3(2,2) - p_3(3,3) = 0.1907$

The equation (2.27) is proposed by Reiser and Lavenberg (1980) in the original MVA algorithm but it fails numerically as $p_i(0,N)$ tends to zero (Chandy&Sauer(1980)). In such cases, the progress of the algorithm can be described as follows: During the first iterations the algorithm works correctly numerically but after some point the values of $p_i(0,N)$ decreases and eventually becomes negative due to the insufficiency in the number of significant digits kept by the computer. Subsequent iterations are bound to generate large numerical errors. This numerical instability was first reported in Chandy and Sauer(1980). However, the difference in (2.27) can be avoided if the [i] complement system $Q^{[i]}(N)$ is also evaluated. Reiser (1981) proposed a modified approach using the following relation :

$$p_i(0,N) = p_i(0,N-1)\ \frac{\lambda(N)}{\lambda^{[i]}(N)} \tag{2.28}$$

which can be obtained from equations (2.15), (2.16) and (2.17) as :

$$\begin{aligned} p_i(0,N) &= \frac{g^{[i]}(N)}{g(N)} \\ &= \frac{g^{[i]}(N-1)}{g(N-1)}\ \frac{g(N-1)}{g(N)}\ \frac{g^{[i]}(N)}{g^{[i]}(N-1)} \\ &= p_i(0,N-1)\ \frac{\lambda(N)}{\lambda^{[i]}(N)} \end{aligned}$$

where $\lambda^{[i]}(N)$ is the throughput of the network Q(N) with center i removed.

The relation (2.28) is numerically stable but potentially expensive: Determining $\lambda^{[i]}(N)$ is nearly as expensive as determining $\lambda(N)$, so for a given number of service centers it is nearly twice as expensive to solve a network with one load-dependent center as it is to solve a network with only SSFR and IS centers. If a network has two load-dependent centers, i and j, then it will be necessary to obtain $\lambda^{[i,j]}(N)$, $\lambda^{[i]}(N)$, $\lambda^{[j]}(N)$ so the solution will be nearly four times as expensive as if the network had only SSFR and IS centers. In general, if there are n load-dependent centers, the number of additional networks to be considered is $\sum_{i=0}^{n} \binom{n}{i} - 1$ or equivalently $2^n - 1$. This exponential increase in complexity may be prohibitive if there are more then a few load-dependent centers.

2.6.2. Multiple Chain MVA

The single chain MVA algorithm can be extended to the multiple chain case. The multiple chain MVA equations are straightforward extensions of their single chain versions. Assume that there are R closed chains each having a fixed number of customers, N_r r=1,...,R so that $\mathbf{N} = (N_1,...,N_R)$ is the population vector. The following additional notation is adopted for the multiple chain case :

$i^*(r)$: an arbitrarily selected center among the centers visited by chain r customers (marked center of chain r)

τ_{ir} : mean service demand per visit of a chain r customer at center i

θ_{ir} : mean number of visits a chain r customer makes to center i between successive arrivals at the marked center $i^*(r)$ (visit ratio)

$\rho_{ir} = \tau_{ir}\,\theta_{ir}$: Mean service demand brought into queue i by a chain r customer between successive visits to marked center $i^*(r)$

$\lambda_r(\mathbf{N}) = \lambda_{\,i^*(r)r}(\mathbf{N})$: throughput rate at the marked center $i^*(r)$ given that the network population vector **N.**

$\lambda_{ir}(\mathbf{N}) = \theta_{ir}\,\lambda_r(\mathbf{N})$: throughput rate of chain r customers through center i

P_r : routing matrix of chain r customers

$n_{ir}(\mathbf{N})$: mean number of chain r customers at center i

$n_i(\mathbf{N}) = \sum_{r=1}^{R} n_{ir}(\mathbf{N})$: mean queue size at center i

$w_{ir}(\mathbf{N})$: mean waiting time per visit of a chain r customer at center i (including service time)

$W_{ir}(\mathbf{N}) = \theta_{ir}\, w_{ir}(\mathbf{N})$: mean waiting time of a chain r customer in queue i between successive visits to marked center $i^*(r)$

$p_i(\mathbf{k},\mathbf{N}) = \Pr\{\, \mathbf{k}_i = \mathbf{k} \mid \mathbf{N} \,\}$ where $\mathbf{k}_1+\mathbf{k}_2+...+\mathbf{k}_M=\mathbf{N}$ and $\mathbf{k}_i = (k_{i1},...,k_{iR})$

$p_i(k,\mathbf{N}) = \Pr\{\, |\mathbf{k}_i| = k \mid \mathbf{N} \,\}$ where $|\mathbf{k}_i| = k_{i1}+k_{i2}+ .. +k_{iR}$

$\mathbf{k}_i$: R-dimensional state vector of center i

$\mathbf{n}$: M-dimensional state vector of the network

The same capacity function $\mu_i(k)$ applies depending on the number of customers at center i, independently of chain membership with $\mu_i(1)=1$.

θ_{ir}'s are computed from the equation $\theta_r = \theta_r P_r$ where $\theta_r=(\theta_{1r},\theta_{2r},..,\theta_{Mr})$. Since $\theta_{i*(r)r} = 1$ by definition, and P_r is assumed to be an irreducible stochastic matrix, the solution to the above equation is unique.

The equilibrium probability of being in state $(\mathbf{k}_1,\mathbf{k}_2,...,\mathbf{k}_M)$ has a product-form solution given by Basket et al. (1975) assuming that the BCMP conditions in section 2.4. are satisfied :

$$\Pr\{\mathbf{n}=(\mathbf{k}_1,\mathbf{k}_2,...,\mathbf{k}_M) \mid \mathbf{N}\} = \pi_1(\mathbf{k}_1)\,\pi_2(\mathbf{k}_2)...\pi_M(\mathbf{k}_M)\,/\,g(\mathbf{N}) \quad (2.29)$$

where

$$\pi_i(\mathbf{k}_i) = \frac{1}{\mu_i(1)...\mu_i(k_i)}\;\frac{k_i!}{k_{i1}!...k_{iR}!}\;\rho_{i1}^{k_{i1}}\,\rho_{i2}^{k_{i2}}\;...\;\rho_{iR}^{k_{iR}}$$

$k_i = |\mathbf{k}_i| = k_{i1}+k_{i2}+ ... +k_{iR}$

and $g(\mathbf{N})$ is the normalization constant.

The multiple chain counterparts of equations (2.15)-(2.17) are :

i) $$p_i(\mathbf{k},\mathbf{N}) = \pi_i(\mathbf{k})\,\frac{g^{[i]}(\mathbf{N}-\mathbf{k})}{g(\mathbf{N})} \qquad 0 \le \mathbf{k} \le \mathbf{N} \qquad (2.30)$$

ii) $$\lambda_{ir}(\mathbf{N}) = \theta_{ir}\,\frac{g(\mathbf{N}-\mathbf{1}_r)}{g(\mathbf{N})} \qquad (2.31)$$

iii) $$p_i(k,\mathbf{N}) = \frac{1}{\mu_i(k)}\sum_{r=1}^{R}\tau_{ir}\,\lambda_{ir}(\mathbf{N})\,p_i(k-1,\mathbf{N}-\mathbf{1}_r) \quad k=1,...,|\mathbf{N}| \quad (2.32)$$

which are derived as follows :

i) Using equation (2.29),

$$p_i(\mathbf{k},\mathbf{N}) = \sum_{\sum_{j\neq i} \mathbf{k}_j=\mathbf{N}-\mathbf{k}} \Pr\{\mathbf{n}=(\mathbf{k}_1,\ldots,\mathbf{k}_{i-1},\mathbf{k},\mathbf{k}_{i+1},\ldots,\mathbf{k}_M) \mid \mathbf{N}\}$$

$$= \frac{\pi_i(\mathbf{k})}{g(\mathbf{N})} \sum_{\sum_{j\neq i} \mathbf{k}_j=\mathbf{N}-\mathbf{k}} \prod_{l\neq i} \pi_l(\mathbf{k}_l)$$

$$= \pi_i(\mathbf{k}) \; \frac{g^{[i]}(\mathbf{N}-\mathbf{k})}{g(\mathbf{N})}$$

ii) From the definition of throughput,

$$\lambda_{ir}(\mathbf{N}) = \sum_{\mathbf{k}=\mathbf{1}_r}^{\mathbf{N}} p_i(\mathbf{k},\mathbf{N}) \frac{\mu_i(k)\, \mathbf{k}_r}{k} \frac{1}{\tau_{ir}}$$

$$= \sum_{\mathbf{k}=\mathbf{1}_r}^{\mathbf{N}} \pi_i(\mathbf{k}) \frac{g^{[i]}(\mathbf{N}-\mathbf{k})}{g(\mathbf{N})} \frac{\mu_i(k)\, \mathbf{k}_r}{k} \frac{1}{\tau_{ir}}$$

where $\frac{\mu_i(k)\, \mathbf{k}_r}{k}$ is the portion of mean service rate consumed by chain r customers and the summation is over all feasible population vectors and $\mathbf{1}_r$ denotes the R-dimensional unit vector in direction r. After necessary simplifications :

$$= \frac{\theta_{ir}}{g(\mathbf{N})} \sum_{\mathbf{k}=\mathbf{1}_r}^{\mathbf{N}} \frac{\rho_{ir}^{k-1}}{\mu_i(1)\ldots\mu_i(k-1)} \frac{(k-1)!}{k_1!..(k_r-1)!..k_R!} g^{[i]}(\mathbf{N}-\mathbf{k})$$

$$= \frac{\theta_{ir}}{g(\mathbf{N})} \sum_{\mathbf{k}=\mathbf{1}_r}^{\mathbf{N}} \pi_i(\mathbf{k}-\mathbf{1}_r)\, g^{[i]}(\mathbf{N}-\mathbf{k})$$

$$= \theta_{ir} \frac{g(\mathbf{N}-\mathbf{1}_r)}{g(\mathbf{N})}$$

iii) The proof (Kant,1992) is in two parts.

In part one we show : $p_i(\mathbf{k},\mathbf{N}) = \dfrac{1}{\mu_i(k)} \sum_{r=1}^{R} \tau_{ir}\, \lambda_{ir}(\mathbf{N}) p_i(\mathbf{k}-\mathbf{1}_r,\mathbf{N}-\mathbf{1}_r)$

$$p_i(\mathbf{k},\mathbf{N}) = \pi_i(\mathbf{k}) \frac{g^{[i]}(\mathbf{N}-\mathbf{k})}{g(\mathbf{N})}$$

$$= \frac{1}{\mu_i(1)\dots\mu_i(k)} \frac{k!}{k_1!\dots k_R!} \rho_{i1}^{k_1} \rho_{i2}^{k_2} \dots \rho_{iR}^{k_R} \frac{g^{[i]}(\mathbf{N}-\mathbf{k})}{g(\mathbf{N})}$$

$$= \frac{1}{\mu_i(1)\dots\mu_i(k)} \sum_{r=1}^{R} C(\mathbf{k}-\mathbf{1}_r)\, \rho_{i1}^{k_1} \rho_{i2}^{k_2} \dots \rho_{iR}^{k_R} \frac{g^{[i]}(\mathbf{N}-\mathbf{k})}{g(\mathbf{N})}$$

$$= \sum_{r=1}^{R} \frac{1}{\mu_i(1)\dots\mu_i(k-1)} C(\mathbf{k}-\mathbf{1}_r)\, \rho_{i1}^{k_1} \dots \rho_{ir}^{k_r-1} \dots \rho_{iR}^{k_R} \frac{\rho_{ir}}{\mu_i(k)} \frac{g^{[i]}(\mathbf{N}-\mathbf{k})}{g(\mathbf{N})}$$

$$= \frac{1}{\mu_i(k)} \sum_{r=1}^{R} \rho_{ir}\; \pi_i(\mathbf{k}-\mathbf{1}_r) \frac{g^{[i]}(\mathbf{N}-\mathbf{1}_r-(\mathbf{k}-\mathbf{1}_r))}{g(\mathbf{N}-\mathbf{1}_r)} \frac{g(\mathbf{N}-\mathbf{1}_r)}{g(\mathbf{N})}$$

$$= \frac{1}{\mu_i(k)} \sum_{r=1}^{R} \rho_{ir}\; p_i(\mathbf{k}-\mathbf{1}_r,\mathbf{N}-\mathbf{1}_r)\; \lambda_r(\mathbf{N})$$

$$= \frac{1}{\mu_i(k)} \sum_{r=1}^{R} \tau_{ir}\, \theta_{ir}\, \lambda_r(\mathbf{N})\; p_i(\mathbf{k}-\mathbf{1}_r,\mathbf{N}-\mathbf{1}_r)$$

$$= \frac{1}{\mu_i(k)} \sum_{r=1}^{R} \tau_{ir}\, \lambda_{ir}(\mathbf{N})\; p_i(\mathbf{k}-\mathbf{1}_r,\mathbf{N}-\mathbf{1}_r)$$

where $C(\mathbf{k}) = \dfrac{k!}{k_1!\dots k_R!}$ and we used the identity $C(\mathbf{k}) = \sum_{r=1}^{R} C(\mathbf{k}-\mathbf{1}_r)$

Part two :

By definition $p_i(k-1,\mathbf{N}-\mathbf{1}_r) = \sum_{|\mathbf{k}_i|=k-1} p_i(\mathbf{k},\mathbf{N}-\mathbf{1}_r)$

Note that for all $j=1,\dots,R$ $k_{ij} \geq 0$, $k_{ir} \leq N_r-1$, and for all $j \neq r$ $k_{ij} \leq N_j$ by substituting $\mathbf{k}_i-\mathbf{1}_r$ for $\mathbf{k}_i$ on the right hand side, we get

$$p_i(k-1,\mathbf{N}-\mathbf{1}_r) = \sum_{k_{ir}>0\ |\mathbf{k}_i|=k} p_i(\mathbf{k}_i-\mathbf{1}_r,\mathbf{N}-\mathbf{1}_r)$$

first multiplying both sides by $\dfrac{\tau_{ir}\,\lambda_{ir}(\mathbf{N})}{\mu_i(k)}$

$$\frac{\tau_{ir}\,\lambda_{ir}(\mathbf{N})}{\mu_i(k)}\,p_i(k-1,\mathbf{N}-\mathbf{1}_r) = \frac{1}{\mu_i(k)} \sum_{k_{ir}>0\ |\mathbf{k}_i|=k} \tau_{ir}\,\lambda_{ir}(\mathbf{N})\,p_i(\mathbf{k}_i-\mathbf{1}_r,\mathbf{N}-\mathbf{1}_r)$$

then summing over all chains r=1,..,R and changing the order of summation on the RHS,

$$\sum_{r=1}^{R} \frac{\tau_{ir}\,\lambda_{ir}(\mathbf{N})}{\mu_i(k)}\,p_i(k-1,\mathbf{N}-\mathbf{1}_r) = \sum_{k_{ir}>0\ |\mathbf{k}_i|=k} \frac{1}{\mu_i(k)} \sum_{r=1}^{R} \tau_{ir}\lambda_{ir}(\mathbf{N})p_i(\mathbf{k}_i-\mathbf{1}_r,\mathbf{N}-\mathbf{1}_r)$$

from part one :

$$= \sum_{k_{ir}>0\ |\mathbf{k}_i|=k} p_i(\mathbf{k}_i,\mathbf{N})$$

now we can remove the condition $k_{ir} > 0$ since the conditions are not over $\mathbf{N}-\mathbf{1}_r$ any more.

$$= p_i(k,\mathbf{N})$$

The Arrival Theorem holds also for multiple chain product-form networks and can be restated as (Zahorjan et al.,1988) :

"In a product-form network, the average number of customers a chain r customer finds already at center i when it arrives there given network population **N**, is identical to the equilibrium mean queue length at center i when one chain r customer is removed from the network."

At this stage, equations for the SSFR and IS type service centers of the single chain MVA (2.22)-(2.24) can be rewritten for the multiple chain case :

$$w_{ir}(\mathbf{N}) = \tau_{ir}\,(\,1 + \delta_i\, n_i(\mathbf{N}-\mathbf{1}_r)\,) \tag{2.33}$$

$$\lambda_r(\mathbf{N}) = \frac{N_r}{\sum_{i=1}^{M} \theta_{ir}\, w_{ir}(\mathbf{N})} \tag{2.34}$$

$$n_{ir}(\mathbf{N}) = \theta_{ir}\, \lambda_r(\mathbf{N})\, w_{ir}(\mathbf{N}) \tag{2.35}$$

respectively, where δ_i is one if center i is SSFR and zero if center i is IS type.

For load dependent service centers, the mean waiting time equation for single chain (2.26) becomes :

$$w_{ir}(\mathbf{N}) = \tau_{ir} \sum_{j=1}^{|\mathbf{N}|} \frac{j\ p_i(j-1,\mathbf{N}-\mathbf{1}_r)}{\mu_i(j)} \tag{2.36}$$

Proof (Kant, 1992) :

From equation (2.30) we get,

$$p_i(\mathbf{k},\mathbf{N}) = \frac{1}{\mu_i(1)\ldots\mu_i(k)}\ \frac{k!}{k_1!\ldots k_R!}\ \rho_{i1}^{k_1}\, \rho_{i2}^{k_2} \cdots \rho_{iR}^{k_R}\ \frac{g^{[i]}(\mathbf{N}-\mathbf{k})}{g(\mathbf{N})}$$

$$= \frac{1}{\mu_i(1)\ldots\mu_i(k-1)}\ C(\mathbf{k}-\mathbf{1}_r)\ \rho_{i1}^{k_1}\ldots\rho_{ir}^{k_r-1}\ldots\rho_{iR}^{k_R}\ \frac{k}{\mu_i(k)\,k_r}\ \rho_{ir}\ \frac{g^{[i]}(\mathbf{N}-\mathbf{k})}{g(\mathbf{N})}$$

$$= \rho_{ir}\ \pi_i(\mathbf{k}-\mathbf{1}_r)\ \frac{k}{\mu_i(k)\,k_r}\ \frac{g^{[i]}(\mathbf{N}-\mathbf{1}_r-(\mathbf{k}-\mathbf{1}_r))}{g(\mathbf{N}-\mathbf{1}_r)}\ \frac{g(\mathbf{N}-\mathbf{1}_r)}{g(\mathbf{N})}$$

$$= \rho_{ir}\ p_i(\mathbf{k}-\mathbf{1}_r,\mathbf{N}-\mathbf{1}_r)\ \frac{k}{\mu_i(k)\,k_r}\ \lambda_r(\mathbf{N})$$

By definition, $n_{ir}(\mathbf{N}) = \sum_{j=1}^{N_r} j\,\Pr\{\,k_{ir}=j \mid \mathbf{N}\,\}$

$$= \sum_{j=1}^{N_r}\ \sum_{\mathbf{k}_i : k_{ir}=j} j\ p_i(\mathbf{k}_i,\mathbf{N})$$

$$= \sum_{j=1}^{N_r} \sum_{\mathbf{k}_i : k_{ir} = j} \rho_{ir} \; p_i(\mathbf{k}_i - \mathbf{1}_r, \mathbf{N} - \mathbf{1}_r) \frac{j}{\mu_i(|\mathbf{k}_i|)} \; \lambda_r(\mathbf{N})$$

Note that in the last equation, the two summations cover all population vectors at center i where $k_{ir}>0$. Therefore, by grouping together terms of each value of $|\mathbf{k}_i|$ we get,

$$= \rho_{ir} \; \lambda_r(\mathbf{N}) \sum_{j=1}^{|\mathbf{N}|} \sum_{|\mathbf{k}_i|=j \; k_{ir}>0} \frac{j}{\mu_i(|\mathbf{k}_i|)} \; p_i(\mathbf{k}_i - \mathbf{1}_r, \mathbf{N} - \mathbf{1}_r)$$

$$= \rho_{ir} \; \lambda_r(\mathbf{N}) \sum_{j=1}^{|\mathbf{N}|} \frac{j}{\mu_i(j)} \sum_{|\mathbf{k}_i|=j \; k_{ir}>0} p_i(\mathbf{k}_i - \mathbf{1}_r, \mathbf{N} - \mathbf{1}_r)$$

$$= \tau_{ir} \; \lambda_{ir}(\mathbf{N}) \sum_{j=1}^{|\mathbf{N}|} \frac{j}{\mu_i(j)} \; p_i(j-1, \mathbf{N} - \mathbf{1}_r)$$

the proof is completed by applying Little's formula to the last equation.

Similarly, equation (2.28) becomes :

$$p_i(0,\mathbf{N}) = p_i(0,\mathbf{N}-\mathbf{1}_r) \; \frac{\lambda_{jr}(\mathbf{N})}{\lambda_{jr}^{[i]}(\mathbf{N})} \qquad (2.37)$$

where j is any service center other than i and any r such that $N_r>0$ may be used.
The proof is similar to that of equation (2.28). From equation (2.30) we get

$$p_i(0,\mathbf{N}) \;=\; \frac{g^{[i]}(\mathbf{N}\text{-}\mathbf{0})}{g(\mathbf{N})} \;=\; \frac{g(\mathbf{N}-\mathbf{1}_r)}{g(\mathbf{N})} \; \frac{g^{[i]}(\mathbf{N})}{g^{[i]}(\mathbf{N}-\mathbf{1}_r)} \; \frac{g^{[i]}(\mathbf{N}-\mathbf{1}_r)}{g(\mathbf{N}-\mathbf{1}_r)}$$

using equations (2.31) and (2.30) appropriately,

$$= \frac{\theta_{jr}\lambda_r(\mathbf{N})}{\theta_{jr}\lambda_r^{[i]}(\mathbf{N})} \; p_i(0,\mathbf{N}-\mathbf{1}_r) \quad j \neq i$$

$$= \frac{\lambda_{jr}(\mathbf{N})}{\lambda_{jr}^{[i]}(\mathbf{N})} \; p_i(\mathbf{0},\mathbf{N}\text{-}\mathbf{1}_r)$$

If we examine the MVA equations (2.33)-(2.36), it can be seen that to calculate the mean performance measures of a network with population vector **N**, we have to know the mean queue lengths of the networks with population $\mathbf{N}\text{-}\mathbf{1}_r$ r=1,...,R. These networks can be called reduced networks. This relation implies that all the reduced networks are to be solved recursively from the empty state **0** up to full network state **N**.

The easiest way to organize such a recursive solution procedure is to use nested loops; one loop for each chain. The procedure is illustrated by the following example :

Example 2.2 :

Consider a queueing network with two chains (R=2) and a total of four customers ($|\mathbf{N}|=4$) in it. Assume the single chain queueing network of example 2.1 constitutes the first chain with all three customers ($N_1=3$) and a second chain with a single customer ($N_2=1$) is added to the network. Assume also that the probability matrix of the second chain is :

$$P_2 = \left\{ \begin{matrix} 0.7 & 0.2 & 0.1 \\ 0.2 & 0.6 & 0.2 \\ 0.1 & 0.4 & 0.5 \end{matrix} \right\}$$

If we choose the second center as the marked center ($i^*(2)=2$) this time for chain 2, we obtain the following visit ratios :

$\theta_{12} = 6/7$ $\theta_{22} = 1$ $\theta_{32} = 4/7$

Let the service time requirements for chain 2 be :

$\tau_{12} = 3$ $\tau_{22} = 2$ $\tau_{32} = 4$

Table 2.2. Visit Ratios and Service Times

		r = 1	r = 2
θ_{ir}	i = 1	1	6/7
	2	7/6	1
	3	4/3	4/7
τ_{ir}	i = 1	2	3
	2	3	2
	3	6	4

The mean service rates when there are k customers in center i are given in Table 2.3. below :

Table 2.3. Mean Service Rates

$\mu_i(k)$	k = 1	2	3	4
i = 1	1	2	3	4
2	1	1	1	1
3	1	3	5	6

In the following, we scan all reduced networks **n**, $\mathbf{n}=(n_1,n_2,..,n_R)$ with $\mathbf{0} \leq \mathbf{n} \leq \mathbf{N}$ in an order generated by the nested loops. Figure 2.3. illustrates the order of evaluation of all reduced networks when the first chain's index is in the inner loop (or runs faster).

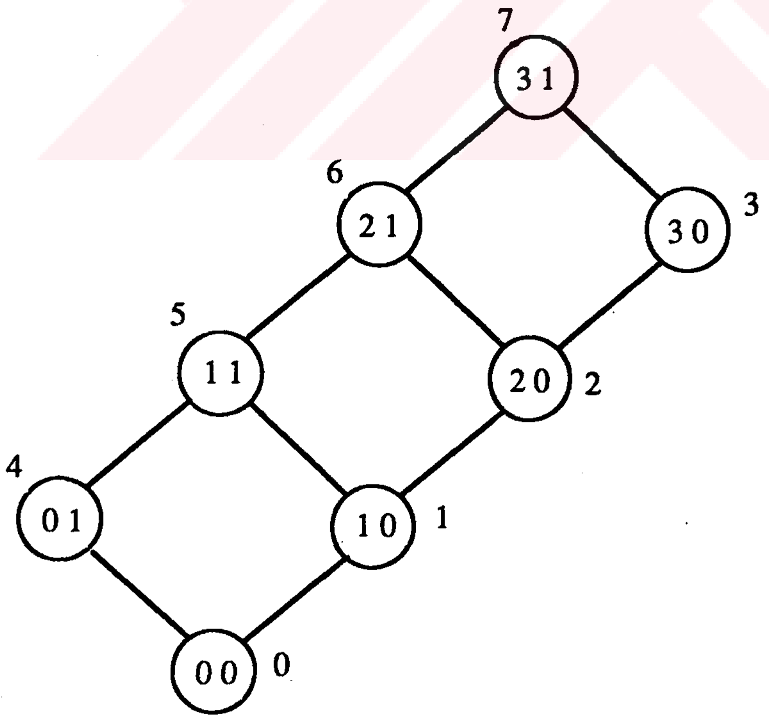


Figure 2.3. Network diagram showing the order of evaluation

n=(0,0)

initially we set $n_{ir}(\mathbf{0}) = 0$ $i=1,2,3.$ $r=1,2.$

and $p_3(0,(0,0)) = 1$ for the load dependent center no.3

The mean performance measures of the reduced networks (1,0), (2,0), (3,0) and (0,1) are computed in the same way as single chain networks. Here, we present a summary of related results in Tables 2.4. and 2.5.

Table 2.4. Mean queue lengths at reduced network **n**

$n_{ir}(\mathbf{n})$		r = 1	r = 2
n=(1,0)	i=1	0.1481	0.0000
	2	0.2593	0.0000
	3	0.5926	0.0000
n=(2,0)	i=1	0.3118	0.0000
	2	0.6872	0.0000
	3	1.0010	0.0000
n=(3,0)	i=1	0.4360	0.0000
	2	1.2873	0.0000
	3	1.2767	0.0000
n=(0,1)	i=1	0.0000	0.3750
	2	0.0000	0.2917
	3	0.0000	0.3333

Table 2.5. Steady-state marginal probabilities, $p_3(j,\mathbf{n})$

	$p_3(j,\mathbf{n})$ j = 0	1	2	3
n = (1,0)	0.4074	0.5956	-	-
n = (2,0)	0.2454	0.5082	0.2464	-
n = (3,0)	0.1907	0.4280	0.2954	0.0859
n = (0,1)	0.6667	0.3333	-	-

n=(1,1)

We can use the MVA equations (2.33) to compute $w_{ir}(\mathbf{n})$ values for all load independent centers and the equation (2.36) for load dependent centers as follows :

$$w_{11}(1,1) = \tau_{11}\ (1+\delta_1\ n_1(0,1)) = 2\ (1+0^*(3/8)) = 2.0$$
$$w_{21}(1,1) = \tau_{21}\ (1+\delta_2\ n_2(0,1)) = 3\ (1+1^*(7/24)) = 31/8$$

$$w_{31}(1,1) = \tau_{31} \left[\frac{1}{\mu_3(1)} \, p_3(0,(0,1)) + \frac{2}{\mu_3(2)} \, p_3(1,(0,1)) \right]$$
$$= 6*(1*(2/3) + (2/3)*(1/3)) = 16/3$$

From (2.34)

$$\lambda_1(1,1) = \frac{n_1}{\theta_{11}w_{11}(1,1)+\theta_{21}w_{21}(1,1) + \theta_{31}w_{31}(1,1)}$$

$$= \frac{1}{1*2 + (7/6)*(31/8) + (4/3)*(16/3)} = 144/1963$$

Using equation (2.35)

$$n_{11}(1,1) = \lambda_{11}(1,1)\, w_{11}(1,1) = \theta_{11}\, \lambda_1(1,1)\, w_{11}(1,1)$$
$$= 1*(144/1963)*2 = 288/1963$$

$$n_{21}(1,1) = \lambda_{21}(1,1)\, w_{21}(1,1) = \theta_{21}\, \lambda_1(1,1)\, w_{21}(1,1)$$
$$= (7/6)*(144/1963)*(31/8) = 651/1963$$

$$n_{31}(1,1) = \lambda_{31}(1,1)\, w_{31}(1,1) = \theta_{31}\, \lambda_1(1,1)\, w_{31}(1,1)$$
$$= (4/3)*(144/1963)*(16/3) = 1024/1963$$

now, we repeat the same procedure for chain 2

$$w_{12}(1,1) = \tau_{12}\, (1+\delta_1\, n_1(1,0)) = 3*(1+0*(4/27)) = 3.0$$
$$w_{22}(1,1) = \tau_{22}\, (1+\delta_2\, n_2(1,0)) = 2*(1+1*(7/27)) = 68/27$$
$$w_{32}(1,1) = \tau_{32} \left[\frac{1}{\mu_3(1)} \, p_3(0,(1,0)) + \frac{2}{\mu_3(2)} \, p_3(1,(1,0)) \right]$$
$$= 4*(1*(11/27) + (2/3)*(16/27)) = 260/81$$

$$\lambda_2(1,1) = \frac{n_2}{\theta_{12}w_{12}(1,1) + \theta_{22}w_{22}(1,1) + \theta_{32}w_{32}(1,1)}$$

$$= \frac{1}{(6/7)*3 + 1*(68/27) + (4/7)*(260/81)} = 567/3926$$

$$n_{12}(1,1) = \lambda_{12}(1,1)\, w_{12}(1,1) = \theta_{12}\, \lambda_2(1,1)\, w_{12}(1,1)$$
$$= (6/7)*(567/3926)*3 = 729/1963$$

$$n_{22}(1,1) = \lambda_{22}(1,1)\, w_{22}(1,1) = \theta_{22}\, \lambda_2(1,1)\, w_{22}(1,1)$$
$$= 1*(567/3926)*(68/27) = 714/1963$$

$$n_{32}(1,1) = \lambda_{32}(1,1)\, w_{32}(1,1) = \theta_{32}\, \lambda_2(1,1)\, w_{32}(1,1)$$
$$= (4/7)*(567/1926)*(260/81) = 520/1963$$

In addition, we need to calculate the marginal probability of j customers at a load dependent center i, namely, $p_3(j,(1,1))$ j=1,2. by using equation (2.32):

$$p_3(1,(1,1)) = \frac{1}{\mu_3(1)}(\theta_{31}\lambda_1(1,1)\tau_{31}p_3(0,(0,1))+\theta_{32}\lambda_2(1,1)\tau_{32}p_3(0,(1,0)))$$
$$= (1/1)*[\ (4/3)*(144/1963)*6*(2/3) + (4/7)*(567/3926)*4*(11/27)\ ]$$
$$= 1032/1963$$

$$p_3(2,(1,1))=\frac{1}{\mu_3(2)}(\theta_{31}\lambda_1(1,1)\tau_{31}p_3(1,(0,1))+\theta_{32}\lambda_2(1,1)\tau_{32}p_3(1,(1,0)))$$
$$=(1/3)*[\ (4/3)*(144/1963)*6*(1/3) + (4/7)*(567/3926)*4*(16/27)]$$
$$= 256/1963$$

$$p_3(0,(1,1)) = 1 - p_3(1,(1,1)) - p_3(2,(1,1)) = 675/1963$$

Same procedure as **n**=(1,1) is applied to networks (2,1) and (3,1) and the mean performance measures associated with these networks are given in Tables 2.6. - 2.9.

Table 2.6. Mean performance measures at **n**=(1,1)

n = (1,1)		r = 1	r = 2
$w_{ir}(\mathbf{n})$	i = 1	2.0000	3.0000
	2	3.8750	2.5185
	3	5.3333	3.2099
$\lambda_r(\mathbf{n})$		0.0734	0.1444
$n_{ir}(\mathbf{n})$	i = 1	0.1467	0.3714
	2	0.3316	0.3637
	3	0.5217	0.2649

Table 2.7. Mean performance measures at **n**=(2,1)

n = (2,1)		r = 1	r = 2
$w_{ir}(\mathbf{n})$	i = 1	2.0000	3.0000
	2	5.0860	3.3758
	3	4.6355	2.9291
$\lambda_r(\mathbf{n})$		0.1417	0.1312
$n_{ir}(\mathbf{n})$	i = 1	0.2834	0.3375
	2	0.8408	0.4429
	3	0.8758	0.2196

Table 2.8. Mean performance measures at **n**=(3,1)

n = (3,1)		r = 1	r = 2
$w_{ir}(\mathbf{n})$	i = 1	2.0000	3.0000
	2	6.8508	4.5754
	3	4.3856	2.8430
$\lambda_r(\mathbf{n})$		0.1894	0.1140
$n_{ir}(\mathbf{n})$	i = 1	0.3788	0.2932
	2	1.5138	0.5216
	3	1.1075	0.1852

Table 2.9. Steady-state marginal probabilities, $p_3(j,\mathbf{n})$

	$p_3(j,\mathbf{n})$				
	j = 0	1	2	3	4
n = (1,1)	0.3439	0.5257	0.1304	-	-
n = (2,1)	0.2428	0.4634	0.2495	0.0444	-
n = (3,1)	0.2054	0.4175	0.2712	0.0910	0.0149

The MVA algorithm for multiple chain networks can now be stated formally :

MVA Algorithm

Initialization :
$n_{ir}(\mathbf{0}) = 0$ for all i=1,...,M. and r=1,...,R.
$p_i(0,\mathbf{0}) = 1$ for all load dependent centers i

```
Main Loop :
For all networks $\mathbf{n}=0$ to $\mathbf{N}$ do
        For all chains $r=1$ to $R$ do
        For all centers $i=1$ to $M$ do

            $w_{ir} = \tau_{ir}$   if center i is of type IS

            $w_{ir} = \tau_{ir} \left( 1+ \sum_{k=1}^{R} n_{ik}(\mathbf{n}-\mathbf{1}_r) \right)$ if center i is load independent

            $w_{ir} = \tau_{ir} \sum_{j=1}^{|\mathbf{n}|} \frac{j}{\mu_i(j)} p_i(j-1,\mathbf{n}-\mathbf{1}_r)$ if center i is load dependent

        $\lambda_r(\mathbf{n}) = \frac{n_r}{\sum_{i=1}^{M} \theta_{ir} w_{ir}}$

        For all centers $i=1$ to $M$ do
                $n_{ir}(\mathbf{n}) = \theta_{ir} \lambda_r(\mathbf{n}) w_{ir}$

        For load dependent centers i do
                For each customer $j=1$ to $|\mathbf{n}|$ do
                $p_i(j,\mathbf{n}) = \sum_{r=1}^{R} \frac{\theta_{ir} \lambda_r(\mathbf{n}) \tau_{ir}}{\mu_i(j)} p_i(j-1,\mathbf{n}-\mathbf{1}_r)$
        $p_i(0,\mathbf{n}) = 1 - \sum_{j=1}^{|\mathbf{n}|} p_i(j,\mathbf{n})$
End Main Loop
```

2.7. Mean Value Analysis by Chain Algorithm

MVAC is a recursive algorithm like all of the exact algorithms for product form queueing networks. The most important feature of MVAC is the fact that it combines the relative advantages of MVA and RECAL into a single computational algorithm: The recursion of MVAC is similar in structure to the recursion that is used in RECAL; so are the storage and computational complexities. The complexity of MVAC grows polynomially as the number of chains increases as in RECAL. The recursion used in MVAC involves only mean performance measures of some related networks and as in MVA, does not require the computation of the normalization constant and consequently avoids numerical instabilities.

MVAC was first reported in Conway et al. (1989) which is a combination of two earlier works by Silva and Lavenberg (1986) and Conway (1986), which were carried out independently.

We first derive the MVAC recursion for multiple chain product form networks having only SSFR and IS service centers and later extend the algorithm to cover centers with queue length dependent service rates. The notation used in the remainder is the same notation used so far, where additionally L_{ir} is the mean number of chain r customers at center i.

First we consider networks with only SSFR (where $\mu_i(n)=1$ for all n) and IS (where $\mu_i(n)=n$ for all n) service centers. In order to establish the recursion, the network is converted to a single customer per chain network by replacing each chain r with $N_r > 1$ by N_r identical single customer chains. This conversion has no physical effect on the network and the mean performance measures remain unchanged. Therefore, from now on we assume that $N_r=1$, $r=1,\ldots,R$. where R is now the total number of chains and total number of customers as well. This network will be called the ' original network '. Now, we consider a ' related network ' which is obtained by keeping the first r chains of the original network and replacing the remaining chains by some number of single customer self-looping (SCSL) chains. A ' self-looping chain ' loops continuously through a single service center and has a relative utilization equal to one. Let ν_i denote the number of SCSL chains at center i and let $\boldsymbol{\nu} = (\nu_1,\ldots,\nu_M)$ be the vector denoting the distribution of R-r SCSL chains of the related network over service centers. Let $G_r(\boldsymbol{\nu})$ be the normalization constant of this related network. Let

$$I_r = \{ \boldsymbol{\nu} : \sum_{i=1}^{M} \nu_i = R\text{-}r \;, \nu_i \geq 0 \,, i=1,\ldots,M. \} \qquad (2.38)$$

i.e., I_r is the set of all $\boldsymbol{\nu}$'s associated with the above related network. Note that $G_R(\mathbf{0})$ is the normalization constant of the original network where **0** is the vector of zeros.

In RECAL, Conway and Georganas(1986) derive the following recursive equation which expresses the normalization constant of a related network, $G_r(\boldsymbol{\nu})$ as a weighted sum of $G_{r-1}(\boldsymbol{\nu}+\mathbf{1}_i)$, the normalization constant of a related network obtained by replacing the r^{th} chain by a SCSL chain at center i.

$$G_r(\boldsymbol{\nu}) = \sum_{i=1}^{M} (1+\nu_i\delta_i)\, \rho_{ir}\, G_{r-1}(\boldsymbol{\nu}+\mathbf{1}_i) \quad \boldsymbol{\nu} \in I_r \quad r=1,\ldots,R. \quad (2.39)$$

where $\mathbf{1}_i$ is a vector of zeros except a one in the ith position, and

$$\delta_i = \begin{cases} 0 \text{ if center i is IS} \\ 1 \text{ if center i is SSFR} \end{cases}$$

The above recursion was used by Conway et al. (1989) as a starting point to derive the MVAC recursion involving only mean performance measures. Let $L_{il}^{r}(\boldsymbol{\nu})$ denote the mean number of chain l customers at center i for the related network that has normalization constant $G_r(\boldsymbol{\nu})$.

Silva and Muntz(1988) have shown that for product form networks the mean number of chain l customers at center i is related to the partial derivative of the normalization constant of the network with respect to the relative utilization ρ_{il}. Applying this result to the related network with normalization constant $G_r(\boldsymbol{\nu})$ gives :

$$\frac{\partial G_r(\boldsymbol{\nu})}{\partial \rho_{il}} = \frac{G_r(\boldsymbol{\nu})}{\rho_{il}} L_{il}^{r}(\boldsymbol{\nu}) \tag{2.40}$$

Differentiating the recursive equation in (2.39) with respect to ρ_{jr} and using (2.40) we get :

$$L_{jr}^{r}(\boldsymbol{\nu}) = (1+\nu_j\delta_j)\,\rho_{jr}\,\frac{G_{r-1}(\boldsymbol{\nu}+\mathbf{1}_j)}{G_r(\boldsymbol{\nu})} \tag{2.41}$$

Note that $G_{r-1}(\boldsymbol{\nu}+\mathbf{1}_j)$ is not a function of ρ_{jr} since it does not contain the rth chain of the original network. We can continue differentiating (2.39) with respect to ρ_{jl}, l=1,...,r-1. This can be done by first rewriting the recursion in (2.39) until the terms ρ_{jl} and $G_{l-1}(.)$ appear in the expression explicitly and then, differentiating as above since $G_{l-1}(.)$ is not a function of ρ_{jl}. Equating this partial derivative to the RHS of (2.40) we get a version of equation (2.41) for chain l, l=1,...,r-1. :

$$L_{jl}^{r}(\boldsymbol{\nu}) = \sum_{i=1}^{M} (1+\nu_i\delta_i)\,\rho_{ir}\,\frac{G_{r-1}(\boldsymbol{\nu}+\mathbf{1}_i)}{G_r(\boldsymbol{\nu})}\,L_{jl}^{r-1}(\boldsymbol{\nu}+\mathbf{1}_i) \tag{2.42}$$

substituting equation (2.41) into (2.42) yields :

$$L_{jl}^{r}(\boldsymbol{\nu}) = \sum_{i=1}^{M} L_{ir}^{r}(\boldsymbol{\nu})\,L_{jl}^{r-1}(\boldsymbol{\nu}+\mathbf{1}_i) \qquad l=1,\ldots,r-1. \tag{2.43}$$

summing (2.43) over l=1,...,r-1.

$$\sum_{l=1}^{r-1} L_{jl}^{r}(\boldsymbol{\nu}) = \sum_{l=1}^{r-1}\sum_{i=1}^{M} L_{ir}^{r}(\boldsymbol{\nu})\,L_{jl}^{r-1}(\boldsymbol{\nu}+\mathbf{1}_i)$$

$$L_j^r(\boldsymbol{\nu}) - L_{jr}^r(\boldsymbol{\nu}) = \sum_{i=1}^{M} L_{ir}^r(\boldsymbol{\nu}) \sum_{l=1}^{r-1} L_{jl}^{r-1}(\boldsymbol{\nu}+\mathbf{1}_i)$$

$$= \sum_{i=1}^{M} L_{ir}^r(\boldsymbol{\nu})\, L_j^{r-1}(\boldsymbol{\nu}+\mathbf{1}_i)$$

$$L_j^r(\boldsymbol{\nu}) = \sum_{i=1}^{M} L_{ir}^r(\boldsymbol{\nu})\, L_j^{r-1}(\boldsymbol{\nu}+\mathbf{1}_i) + L_{jr}^r(\boldsymbol{\nu}) \qquad (2.44)$$

where:
$L_j^r(\boldsymbol{\nu})$ is the mean number of customers at center j on the first r chains of the related network (i.e. not counting SCSL chain customers) with normalization constant $G_r(\boldsymbol{\nu})$.

Now, we can make use of the standard MVA equation for the mean waiting time to our related network with normalization constant $G_r(\boldsymbol{\nu})$.

$$w_{ir}^r(\boldsymbol{\nu}) = \tau_{ir}\,(1 + L_i^{r-1}(\boldsymbol{\nu}) + \nu_i) \qquad (2.45)$$

where:
$w_{il}^r(\boldsymbol{\nu})$ denotes the mean waiting time (including service) of the chain l customer at center i for the network with normalization constant $G_r(\boldsymbol{\nu})$ and $L_i^{r-1}(\boldsymbol{\nu}) + \nu_i$ is the mean number of customers at center i when the chain r customer is removed from the network.

Applying Little's formula to the left side of equation (2.45) we obtain :

$$\begin{aligned} L_{ir}^r(\boldsymbol{\nu}) &= \lambda_{ir}^r(\boldsymbol{\nu})\ \tau_{ir}\,(1 + L_i^{r-1}(\boldsymbol{\nu}) + \nu_i) \\ &= \lambda_r^r(\boldsymbol{\nu})\ \theta_{ir}\ \tau_{ir}\,(1 + L_i^{r-1}(\boldsymbol{\nu}) + \nu_i) \\ &= \lambda_r^r(\boldsymbol{\nu})\ \rho_{ir}\,(1 + L_i^{r-1}(\boldsymbol{\nu}) + \nu_i) \end{aligned} \qquad (2.46)$$

for SSFR service centers and

$$L_{ir}^{r}(\boldsymbol{\nu}) = \lambda_r^r(\boldsymbol{\nu})\, \rho_{ir} \tag{2.47}$$

for IS service centers.

where $\lambda_1^r(\boldsymbol{\nu})$ is the throughput rate at the marked center of chain 1 of the related network with normalization constant $G_r(\boldsymbol{\nu})$.

We can rearrange the centers by numbering the SSFR service centers from 1 to M_1 and IS service centers from M_1+1 to M. We assume there exists at least one SSFR service center since otherwise the network has a trivial solution. $\sum_{i=1}^{M} L_{ir}^{r}(\boldsymbol{\nu}) = 1$ since chain r has a single customer and if we substitute equations (2.46) and (2.47) into the LHS of the above equation appropriately, it becomes :

$$\lambda_r^r(\boldsymbol{\nu}) = \frac{1}{\rho_r + \sum_{i=1}^{M_1} \rho_{ir}\,(L_i^{r-1}(\boldsymbol{\nu}) + \nu_i)} \tag{2.48}$$

where:

$$\rho_r = \sum_{i=1}^{M} \rho_{ir}$$

Equations (2.44), (2.46), (2.47), (2.48) constitute a set of recursive equations that may be used to obtain the mean performance measures for chain R in the original network which consists of only SSFR and IS service centers. The algorithm is summarized below:

```
Initialization :
    $L_i^0(\boldsymbol{\nu}) = 0$   $i=1,\dots,M.$          $\boldsymbol{\nu} \varepsilon I_0 \cup I_1 \cup \dots \cup I_R$
Main Recursion :
    For $r = 1,\dots,R$
        For all $\boldsymbol{\nu} \varepsilon I_r \cup I_{r+1} \cup \dots \cup I_R$
            Compute $\lambda_r^r(\boldsymbol{\nu})$ using (2.48)
            Compute $L_{ir}^r(\boldsymbol{\nu})$ using (2.46) and (2.47) $i=1,\dots,M.$
            Compute $L_i^r(\boldsymbol{\nu})$ using (2.44)          $i=1,\dots,M.$
```

As can be noted from the algorithm at the end of the main recursion, $\lambda_R^R(0)$ and $L_{iR}^R(0)$ values which corresponds to only chain R are computed. In order to compute the performance measures of other chains, the chains of the original network are rearranged so that a different chain occupies the chain in the R^{th} position (i.e., last chain) and main recursion part of the algorithm is repeated for each chain (i.e., for each different arrangement).

The performance measures obtained in this way, correspond to the single customer chains of the original network. If there are actually N_r customers in each chain r before conversion, each performance measure obtained above has to be multiplied by N_r to get the actual values of the performance measures. The other performance measures such as w_{ir}, λ_{ir} can be found at the end of the algorithm since $\lambda_{ir} = \theta_{ir}\,\lambda_r$ and $w_{ir} = L_{ir}/\lambda_{ir}$. This completes the description of basic MVAC.

Now we can consider queue length dependent service rates at one or more centers of the network. The service rates are of the form $\mu_i(n_i)$ as a function of the total number of customers present at center i. This extension causes computation and storage difficulties for queue-length distributions as can be expected. Let $P_i^r(n,\boldsymbol{\nu})$ be the probability that there are n customers in service center i in the related network with normalization constant $G_r(\boldsymbol{\nu})$ when only the first r chains are considered. In order to find a relation between networks with normalization constants $G_r(\boldsymbol{\nu})$ and $G_{r-1}(\boldsymbol{\nu}+\mathbf{1}_i)$ we remove chain r from the network and replace it by a SCSL chain at center i with mean service time τ_{ir}. Let $\lambda_r(\boldsymbol{\nu},i)$ denote the throughput of this SCSL chain at center i. The queueing discipline can be assumed to be processor sharing (PS). This assumption does not affect the mean performance measures since the other BCMP queueing disciplines (LCFSPR, FCFS) have identical marginal state distributions. Now, we can write the mean service rate at center i for the SCSL chain as $\mu_i(n+\nu_i+1) / (n+\nu_i+1)$ where n is the number of customers in the first r-1 chains. Note that $(n+\nu_i+1)$ gives the total number of customers at center i.

From the definition of expectation, the overall mean service rate at center i of SCSL chain can be written as :

$$\sum_{n=0}^{r-1} P_i^{r-1}(n,\boldsymbol{\nu}+\mathbf{1}_i)\,\frac{\mu_i(n+\nu_i+1)}{n+\nu_i+1}$$

This overall mean service rate also equals $\lambda_r(\boldsymbol{\nu},i)\,\tau_{ir}$ for queue-length dependent FCFS, LCFSPR or PS type queues and 1 for IS type queues since $\mu_i(n+\nu_i+1)=(n+\nu_i+1)$ and $P_i^{r-1}(n,\boldsymbol{\nu}+\mathbf{1}_i)$ is a proper probability distribution. Therefore, equating these two expressions for overall mean service rates we get:

$$\lambda_r(\boldsymbol{\nu},i) = \begin{cases} \dfrac{1}{\tau_{ir}} \displaystyle\sum_{n=0}^{r-1} P_i^{r-1}(n,\boldsymbol{\nu}+\mathbf{1}_i)\frac{\mu_i(n+\nu_i+1)}{n+\nu_i+1} & \text{if i is PS LCFSPR or FCFS} \\ \dfrac{1}{\tau_{ir}} \text{if i is IS} & \end{cases} \tag{2.49}$$

Additionally, the following relations are established by Conway et al. (1989) using probabilistic arguments.

$$L_{ir}^{r}(\boldsymbol{\nu}) = \theta_{ir}\;\lambda_r(\boldsymbol{\nu},i)^{-1} \Big/ \sum_{j=1}^{M} \theta_{jr}\;\lambda_r(\boldsymbol{\nu},j)^{-1} \tag{2.50}$$

$$\lambda_r^{r}(\boldsymbol{\nu}) = \lambda_r(\boldsymbol{\nu},i^*(r))\;\; L_{i^*(r)r}^{r}(\boldsymbol{\nu}) \tag{2.51}$$

$$P_i^{r}(n,\boldsymbol{\nu}) = L_{ir}^{r}(\boldsymbol{\nu})\,P_i^{r-1}(n-1,\boldsymbol{\nu}+\mathbf{1}_i) + \sum_{j\neq i\; j=1}^{M} L_{jr}^{r}(\boldsymbol{\nu})\,P_i^{r-1}(n,\boldsymbol{\nu}+\mathbf{1}_j) \tag{2.52}$$

where;

$$0 \le n \le r,\;\; P_j^{r-1}(n,\boldsymbol{\nu}+\mathbf{1}_j) = 0 \text{ if } n > r\text{-}1 \text{ (or } n < 0\text{), and } P_j^{0}(0,\boldsymbol{\nu}) = 1.$$

Equations (2.49)-(2.52) form a set of recursive equations which requires the computation of $\lambda_r^{r}(\boldsymbol{\nu})$, $L_{ir}^{r}(\boldsymbol{\nu})$ and $P_i^{r}(n,\boldsymbol{\nu})$ at each recursion. If we carefully look at the above equation, we can see that the computation of $\lambda_r^r(\boldsymbol{\nu})$, $L_{ir}^r(\boldsymbol{\nu})$ and $P_i^r(n,\boldsymbol{\nu})$ for n=0,...,r only requires the queue-length distributions associated with the networks having normalization constant $G_{r-1}(\boldsymbol{\nu}+1_i)$ i=1,...,M. Therefore, in MVAC recursion, it is now sufficient to compute $\boldsymbol{\nu}$'s over only the set I_r instead of the set $I_r \cup I_{r+1} \cup ... \cup I_R$. Note that this simplification was not possible with SSFR and IS service centers having fixed service rates even for a reduced set like $I_r \cup I_{r+1} \cup I_{r+2}$. For example, suppose that when r=1 the mean performance measure of the related network is computed for $\boldsymbol{\nu} \varepsilon I_1 \cup I_2 \cup I_3$ using equations (2.44), (2.46), (2.47), (2.48). In the next iteration, r becomes 2 and the mean performance measures of this network are to be computed for $\boldsymbol{\nu} \varepsilon I_2 \cup I_3 \cup I_4$. However, when $\boldsymbol{\nu} \varepsilon I_4$ equation (2.48) necessitates that the values for $L_i^1(\boldsymbol{\nu})$ should already be computed in the previous iteration when r=1 but they were not. Choosing the set $I_1 \cup I_2 \cup I_3 \cup I_4$ initially does not help because the same problem arises when $\boldsymbol{\nu} \varepsilon I_5$ in the second iteration.

A summary of the basic step of MVAC algorithm is given below for queue-length dependent servers.

```
Initialization:
        For all $\boldsymbol{\nu} \varepsilon I_0$
            $P_i^0(n,\boldsymbol{\nu}) = 0$        i=1,...,M.    $n > 0$
            $P_i^0(0,\boldsymbol{\nu}) = 1$        i=1,...,M.

Main recursion:
        For r=1,...,R.
            For all $\boldsymbol{\nu} \varepsilon I_r$
                Compute $\lambda_r(\boldsymbol{\nu},i)$ using (2.49) i=1,...,M.
                Compute $L_{ir}^r(\boldsymbol{\nu})$ using (2.50) i=1,...,M.
                Compute $\lambda_r^r(\boldsymbol{\nu})$ using (2.51).
                For i=1,...,M.
                    For n=0,...,r.
                        Compute $P_i^r(n,\boldsymbol{\nu})$ using (2.52).
```

2.8. Approximate MVA Algorithms for Product Form Networks

In this section we consider MVA based heuristics for queueing networks for which either exact analytical results are not available or if they are available the computation of the performance measures is prohibitively expensive. Thus, one is interested in faster solution algorithms for product-form queueing networks with a large number of closed chains such as models of computer communication networks where each closed chain represents a window flow control on virtual channels (e.g., Reiser,1979). Hundreds of such virtual channels between source-destination node pairs may be active at a time. Exact solution algorithms such as the convolution and MVA algorithms are obviously not applicable since their computation time and memory space requirements grow very fast (exponentially) with the number of chains and/or with the number of customers in the chains. In practice, exact algorithms can not be used efficiently for most networks with more than six or seven chains unless the network has additional special route sparsity structure (Lam and Lien,1983). Therefore it is important to develop reasonably accurate and cost effective methods for the approximate analysis of closed queueing networks which handle primarily those queueing networks with multiple chains. The common strategy of MVA based heuristics is to eliminate the need for recursive evaluation of all networks with population vector **n**, $\mathbf{n} \leq \mathbf{N}$ and thus, saving from the memory space as well as computation time.

One of the first approximations is given by Reiser and Lavenberg (1980) who introduce the correction terms $\varepsilon^{j}_{ir}(N)$ as follows :

$$L_{ir}(N-1_j) = L_{ir}(N) - \varepsilon^{j}_{ir}(N) \qquad (2.53)$$

where $L_{ir}(N)$ is the mean queue length of chain r customers at center i of a network with population vector **N**. They estimate the values for $\varepsilon^{j}_{ir}(N)$ assuming :

(i) $\varepsilon^{j}_{ir}(N) = 0 \quad \text{for } j \neq r$ (2.54) i.e. removing one customer from a chain affects only that chain and the performance measures of other chains are unaffected.

(ii) In order to estimate the correction terms at the affected chain (j=r), the affected chain is solved separately as a single chain problem with N_r customers and with adjusted parameters $\bar{\rho}_i = \rho_{ir} \frac{L_i}{L_{ir}}$ where ρ_{ir} is the traffic intensity (or load) of chain r customers through center i and the adjustment assumes that the traffic intensities are directly proportional to the mean number of customers. The correction term becomes :

$$\varepsilon_{ir}^{r}(N) = \bar{L}_i(N_r) - \bar{L}_i(N_r-1) \qquad (2.55)$$

where $\bar{L}_i$ is the mean queue length at center i in the single chain problem. Then the MVA equations are converted into a set of non-linear equations :

$$\varepsilon_{ir}^{j} = f(L_{ir} : i=1,..,M.\ r=1,..,R.) \qquad (2.56)$$

$$W_{ir} = \rho_{ir}\left(1 + L_i - \sum_{j=1}^{R} \varepsilon_{ij}^{r}\right) \qquad (2.57)$$

$$\lambda_r = \frac{N_r}{\sum_{i=1}^{M} W_{ir}} \qquad (2.58)$$

$$L_i = \sum_{i=1}^{M} \lambda_r W_{ir} \qquad (2.59)$$

The above equations can be solved by cyclic iteration through (2.56) to (2.59). Reiser and Lavenberg(1980) have shown that this heuristic algorithm is asymptotically exact (i.e. the approximation error tends to zero as the chain populations go to infinity) and hence, it is expected to work better the larger the problem.

Bard(1979) used an approximation which breaks the recursive nature of the exact MVA equations :

$$L_{ir}(N-1_k) = L_{ir}(N) \quad \text{for all } i,r,k \tag{2.60}$$

Using this approximation in the mean waiting time equation of MVA gives a set of non-linear equations in which all variables are now at the same population level. But then, we lose the intuitive interpretation of MVA equations therefore we can not be confident that the existence, uniqueness, convergence properties of the exact MVA are preserved. Chow(1983) reduced these equations to R non-linear equations in throughputs. He then showed that these equations have a unique solution in the feasible region where all utilizations are between zero and one. However, the above approximation is not very accurate for small populations.

A widely used and more accurate approximation is proposed by Schweitzer(1979) which improves Bard's approximation by using a separate equation for the chain from which one customer is removed :

$$L_{ir}(N-1_k) = \begin{cases} L_{ir}(N) & \text{if } r \neq k \\ \dfrac{N_k-1}{N_k} L_{ik}(N) & \text{if } r=k \end{cases} \tag{2.61}$$

This approximation, while keeping the assumption of "the removal of a single customer from chain k does not affect the performance metrics of other chains" introduces the chain k population level N_k into approximation equations. Hence, this approximation becomes sensitive to N_k and it is expected to behave better for chains with small populations with respect to Bard's approximation. Note that this approximation approaches Bard's approximation as chain k population N_k tends to infinity.

There exist two approaches of solving this approximation :

i. In the first approach, Schweitzer's approximation is combined with the MVA equations to obtain the following simultaneous non-linear equations in the unknowns $w_{ir}(N)$ and $L_{ir}(N)$:

$$w_{ir}(N) = \tau_{ir} \left(1 + L_i(N) - \frac{L_{ik}(N)}{N_k} \right) \tag{2.62}$$

$$L_{ir}(N) = N_r \frac{\theta_{ir} w_{ir}(N)}{\sum_{i=1}^{M} \theta_{ir} w_{ir}(N)} \tag{2.63}$$

where $L_i(N) = \sum_{r=1}^{R} L_{ir}(N)$ and equation (2.62) is obtained by substituting Schweitzer's approximation (2.61) into the MVA equation for mean waiting times. Equation (2.63) is the combination of the two other MVA equations eliminating the throughput. Equations (2.62) and (2.63) are then reduced to R non-linear equations in R unknowns in throughputs (Lavenberg,1983:179) and can be solved using an existing well tested program. Lavenberg (1983) also expressed approximate values of mean queue sizes and mean waiting times directly in terms of the chain r throughputs r=1,...,R. Silva et al. (1984) have proved that these non-linear equations have at least one solution in the feasible region but they have not been able to show the uniqueness of solution. Eager and Sevcik (1984) analyzed Schweitzer's approximation and showed that the equations defining the approximation have a unique solution when there is only a single chain, and the algorithm converges monotonically when the specified initialization is adopted. It is also proved that the solution is pessimistic relative to the exact queueing network solution.

ii. In the second approach, equations (2.62) and (2.63) are numerically solved by the method of successive approximations: Estimates for w_{ir}'s are computed from (2.62) given the initial estimates of L_{ir}'s and they are substituted in equation (2.63) to obtain new estimates for L_{ir}'s. These new estimates are used to calculate new estimates for

w_{ir}'s and this iterative procedure continues until two consecutive estimates of L_{ir}'s are sufficiently close to each other. Note that equation (2.63) guarantees that at each iteration the sum of L_{ir}'s over i is equal to N_r. When the convergence issue is examined, there is no guarantee for convergence in using this successive substitutions method in multiple chain networks. However, a number of experiments made in literature reveals that the algorithm converges practically. Zahorjan et al. (1988) also have given examples of both single and multiple chain networks in which convergence occurs only slowly. The single chain network they considered has two centers connected in cyclic form. The service demands τ_i are chosen arbitrarily close to one another (e.g. $\tau_2 = \tau_1 + \varepsilon$, $\varepsilon>0$, small). For small number of customers, the approximate and exact solutions to networks are almost the same. In theory, the bottleneck queue length is unbounded while the queue length at the non-bottleneck center approaches its finite asymptotic limit of $\rho/(1-\rho)$ where $\rho=\tau_2/(\tau_1+\varepsilon)$. However, for very large network populations, the approximation algorithm have difficulty in determining such a large difference between the queue lengths whereas the two centers are almost identical.

Bard (1980) has applied the approximation method to hundreds of networks of different sizes and populations and reported that the results are generally within 5 percent of the exact solution without any failure to convergence. Furthermore, this approximation can be applied directly to chains with non-integral populations: Such cases arises in network models where chain populations are average values derived from a higher level model.

Chandy and Neuse (1982) have applied this method to both randomly generated networks and stress networks. Stress networks are obtained by choosing networks which they expect to violate Schweitzer's approximation: They considered a network in which one chain consists of a single customer competing significantly with the customers of other chains at a center. The removal of that customer, will affect significantly the mean queue lengths of other chains at that center, contrary to the assumption of Schweitzer's approximation. Then, they varied the parameters of these networks one parameter at a time and shifted the parameters in the direction of increasing error. However, they state that the Schweitzer's algorithm performs better on most networks tested. In the following, we present the Schweitzer's algorithm for multiple chain networks :

Initialization :
Estimate mean queue lengths at population **N** by equally distributing each chain's population over centers.

$$L_{ir}^{0}(N) = N_r / M \qquad i=1,..,M \;\; r=1,..,R.$$

Set $I = 1$

Step 1 :
Compute approximate queue lengths at population **N-1**$_k$, $k=1,..,R$ by using Schweitzer's approximation:

$$L_{ir}(N\text{-}1_k) = \begin{cases} L_{ir}(N) & \text{if } r \neq k \\ \dfrac{N_k - 1}{N_k} L_{ik}(N) & \text{if } r = k \end{cases}$$

Step 2 :
Compute new estimates of $L_{ir}(N)$ by making a single exact MVA iteration, i.e.,

$$w_{ir}(N) = \tau_{ir} \left(1 + \delta_i \sum_{r=1}^{R} L_{ir}(N\text{-}1_k) \right)$$

$$L_{ir}(N) = N_r \frac{\theta_{ir} w_{ir}(N)}{\sum_{i=1}^{M} \theta_{ir} w_{ir}(N)}$$

Step 3 : (Termination Test)

$$\text{error} = \text{Max} \{ \mid L_{ir}^{I}(N) - L_{ir}^{I-1}(N) \mid / N_r \}$$

over all i,r

If error $> \varepsilon$ set $I = I + 1$ go to Step 1
else Stop.

where the superscript I denotes iteration I statistics and ε is set equal to $1/(4000 + 16\ |\mathbf{N}|)$

Note that Schweitzer's approximation assumes that the mean fractions of a chain's customers in each center do not change if there is one less customer in the network. In reality, these fractions do change. Chandy and Neuse (1982) took into consideration these changes in fractions and developed an improved approximation algorithm known as Linearizer.

Linearizer Algorithm

Let $F_{ir}(\mathbf{N})$ denote the fraction of chain r customers at center i when the network population vector is $\mathbf{N}$. i.e.,

$$F_{ir}(\mathbf{N}) = L_{ir}(\mathbf{N}) \ / \ N_r \tag{2.64}$$

and let $D_{irk}(\mathbf{N})$ be the change in F_{ir} when a single chain k customer is removed from the network with population vector $\mathbf{N}$.

$$D_{irk}(\mathbf{N}) = F_{ir}(\mathbf{N}-\mathbf{1}_k) - F_{ir}(\mathbf{N}) \tag{2.65}$$

then, $L_{ir}(\mathbf{N}-\mathbf{1}_k)$ can be written as follows :

$$L_{ir}(\mathbf{N}-\mathbf{1}_k) = (\mathbf{N}-\mathbf{1}_k)_r \ (F_{ir}(\mathbf{N}) + D_{irk}(\mathbf{N})) \tag{2.66}$$

where $(\mathbf{N}-\mathbf{1}_k)_r$ is the rth component of the vector $\mathbf{N}-\mathbf{1}_k$

Note that equation (2.66) reduces to (2.61) when D_{irk} 's are assumed to be zero. Thus, we can iteratively calculate the estimates of $L_{ir}(\mathbf{N})$ as in Schweitzer's heuristic provided that $D_{irk}(\mathbf{N})$ values are given. This procedure is known as the Core algorithm. It requires the estimates of $D_{irk}(\mathbf{N})$ and $L_{ir}(\mathbf{N})$ values as inputs to compute the estimates of mean performance measures at population level $\mathbf{N}$ (i.e., $L_{ir}(\mathbf{N})$, $w_{ir}(\mathbf{N})$, $\lambda_r(\mathbf{N})$). In the following the Core algorithm is given:

Inputs :
Estimates of $L_{ir}(N)$ and $D_{irk}(N)$

Step 1 :
Compute approximate mean queue lengths at population $\mathbf{N}-\mathbf{1}_k$ by using equation (2.66).

Step 2 :
Compute new estimates of $L_{ir}(N)$ exactly as in Step 2 of Schweitzer's algorithm.

Step 3 :
Same as Step 3 of Schweitzer's algorithm.

Chandy and Neuse (1982) refer to the Core algorithm with inputs $D_{irk}(N) = 0$ as the Schweitzer-Core algorithm. Note that the Core algorithm reduces to Schweitzer's algorithm in such a situation. Thus, the Core algorithm is more general than Schweitzer's algorithm and Linearizer calls the Core algorithm as a subroutine each time with different input parameters $L_{ir}(N)$ and $D_{irk}(N)$.

Now, we can estimate the values for $D_{irk}(N)$'s from its definition (2.65). The Core algorithm is invoked R+1 times; once to the network $\mathbf{N}$ to calculate the estimates of $L_{ir}(N)$ and once to each $\mathbf{N}-\mathbf{1}_k$ $k=1,..,R$ independently to calculate the estimates of $L_{ir}(\mathbf{N}-\mathbf{1}_k)$. But, each application of the Core algorithm requires the estimates of D_{irk}'s. As Linearizer is an iterative algorithm we can use the most recent estimates available for D_{irk}'s. At this point, another problem arises: Applying the Core algorithm at population levels $\mathbf{N}$ and $\mathbf{N}-\mathbf{1}_k$ implies that we already know the D_{irk} values at levels $\mathbf{N}$ and $\mathbf{N}-\mathbf{1}_k$ whereas, the equation (2.65) provides only the estimates for $D_{irk}(N)$. Thus we have to make an additional assumption: Linearizer assumes that $D_{irk}(N) = D_{irk}(\mathbf{N}-\mathbf{1}_j)$ $j=1,...,R$. independently of the population level. This assumption means that the change in the F_{ir} caused by the removal of a chain k customer is a constant and independent of the population level $\mathbf{N}$. This is to say, F_{ir} is a linear function of chain populations, hence the name Linearizer.

Schweitzer's heuristic assumes the F_{ir}'s are constant. Linearizer assumes that the change in F_{ir}'s is constant. Thus, if we consider Schweitzer's heuristic as first order, we can say that Linearizer is of second order. Similarly, a third order algorithm can be designed by assuming that the change in D_{irk}'s is constant (e.g. $S_{irkj}(N)= D_{irk}(N\text{-}1_j) - D_{irk}(N)$). However, each higher order algorithm requires an additional subscript in the notation increasing the space and time complexities of the algorithm by a factor of R.

After the estimation of D_{irk} values the Core algorithm is applied again R+1 times as in the first iteration of Linearizer. Chandy and Neuse (1982) reported that more than three Linearizer iterations produce negligible improvement in accuracy and they used this fixed three iteration rule as the stopping condition of the Linearizer algorithm. The Linearizer algorithm is given below :

Initialization :
Estimate mean queue lengths at population **N** and $\mathbf{N\text{-}1_k}$ by equally distributing each chain's population over centers:
Set

$L_{ir}(N) = N_r / M \qquad i=1,..,M \;\; r=1,..,R.$

$L_{ir}(N\text{-}1_k) = (N\text{-}1_k)_r / M$

$D_{irk}(N) = 0$

$L = 1$

Step 1 :
Apply the Core algorithm at population **N**. Use the most recent estimates of $L_{ir}(N)$ and $D_{irk}(N)$. The only output is an improved estimate of $L_{ir}(N)$.

Termination Test :

If $L = 3$ then Stop, otherwise continue with Step 2.

Step 2 :
Apply the Core algorithm at each of the R populations $L_{ir}(N-1_j)$ j=1,..,R. independently. Use the most recent estimates of $L_{ir}(N-1_j)$ and $D_{irk}(N-1_j)$ as inputs. Use the Linearizer assumption $D_{irk}(N-1_j)=D_{irk}(N)$.

Step 3 :
Compute new estimates of $D_{irk}(N)$ from (2.65) for all i,r and k.

Set $L = L + 1$ and go to Step 1.

Note that the Linearizer assumption $D_{irk}(N-1_j) = D_{irk}(N)$ allows us to use a single three-dimensional array to store the values of D_{irk} 's throughout the Linearizer algorithm. A FORTRAN code for the Linearizer algorithm is given in Appendix G.

In the Linearizer algorithm, the computational complexity of a single Core iteration is $O(MR^2)$. Chandy and Neuse (1982) reported according to the results of their experiments that excluding the cost of computing the D values, Linearizer costs approximately 1.54 * (R+1) times as much as the Schweitzer-Core algorithm. The D values can be computed in $O(MR^2)$ operations in step 3 of Linearizer. Therefore, the computational complexity of the Linearizer becomes $O(MR^3)$.

Silva and Muntz (1990) showed that with some algebraic manipulations, the computational cost of Linearizer can be reduced to $O(MR^2)$. This implementation of Linearizer algorithm is known as the Improved Linearizer and the results of these two algorithms are exactly the same.

Zahorjan et al. (1988) proposed another approximate MVA algorithm, the Aggregate Queue Length (AQL) algorithm. The idea behind this algorithm involves computing the changes in total queue length caused by the removal of a single customer from the network instead of dealing with the effect on each individual chain. AQL algorithm uses a modified version of Linearizer equations: Equations (2.65) and (2.66) become

$$\gamma_{ik}(N) = \frac{L_i(N-1_k)}{N-1} - \frac{L_i(N)}{N} \qquad (2.67)$$

and $$L_i(N-1_k) = (N-1)\left(\frac{L_i(N)}{N} + \gamma_{ik}(N)\right) \qquad (2.68)$$

respectively.
Similarly, the Linearizer assumption takes the following form :

$$\gamma_{ik}(N-1_j) = \gamma_{ik}(N) \qquad j=1,..,R. \qquad (2.69)$$

Notice that AQL algorithm aims to compute $L_i(N-1_k)$ values which are exactly needed by the single step MVA iteration of the Core algorithm. This aggregation is quite similar to the implementation of Silva and Muntz (1990) and allows a reduction in both time and space requirements by a factor of R with respect to the original Linearizer algorithm. The iterative structure of the AQL algorithm is exactly the same as that of Linearizer except that they used a relative change criteria on the queue lengths as a stopping rule to prevent possible premature terminations. They performed a number of experiments on randomly generated networks to compare the AQL algorithm to the Linearizer and the Schweitzer-Core algorithm with respect to accuracy and execution time. The space and time complexities of the approximate MVA algorithms as well as the exact MVA algorithm are given in Table 2.10.

Table 2.10. Space and Time Requirements of Approximate and Exact MVA Algorithms

Algorithms	Space Requirement	Time Requirement
Schweitzer-Core	$O(MR)$	$O(MR)$
Linearizer	$O(MR^2)$	$O(MR^3)$
Improved Lin.	$O(MR^2)$	$O(MR^2)$
AQL	$O(MR)$	$O(MR^2)$
Exact MVA	$O(M\prod_{r=1}^{R}(N_r+1))$	$O(MR\prod_{r=1}^{R}(N_r+1))$

This table reveals the large difference between approximate and exact algorithms in terms of storage and time requirements very clearly.

All the approximations discussed up to now, involve iterative computations before the approximate solutions are obtained. Hsieh and Lam (1989) proposed three non-iterative approximation algorithms. They reported that these algorithms are suitable for networks with many chains and sufficiently accurate for the analysis and design of communication networks. The method is based on the distribution of a chain' s population over centers proportional to the loads of the centers. The accuracy of the method is tested against the exact solutions obtained by the Tree Convolution Algorithm.

Extensions to Load-Dependent Centers :

MVA based approximations for closed product-form networks with load-dependent centers were first introduced by Neuse and Chandy (1981). SCAT (Self Correcting Approximation Technique) is an early version of Linearizer allowing load-dependent centers. However, the numerical examples have shown that large errors can occur in estimating the performance measures at load-dependent centers using this method. The main reason of this inaccuracy is the inadequate estimation of the marginal queue size probabilities at population levels $\mathbf{N}\text{-}\mathbf{1}_k$.

SCAT assigns probabilities to only two neighboring values of the mean queue length at center i, $L_i(\mathbf{N}\text{-}\mathbf{1}_k)$ and the probabilities are estimated as follows :

Step 1 :
Compute $(\text{floor})_{ik}$ and $(\text{ceiling})_{ik}$, the two integers surrounding $L_i(\mathbf{N}\text{-}\mathbf{1}_k)$:

$$(\text{floor})_{ik} = L_i(\mathbf{N}\text{-}\mathbf{1}_k) \qquad (2.70)$$

$$(\text{ceiling})_{ik} = (\text{floor})_{ik} + 1 \qquad (2.71)$$

Step 2 :
Compute the marginal probabilities as follows :

$$p_i((\text{floor})_{ik}, \mathbf{N}\text{-}\mathbf{1}_k) = (\text{ceiling})_{ik} - L_i(\mathbf{N}\text{-}\mathbf{1}_k) \qquad (2.72)$$

$$p_i((\text{ceiling})_{ik}, \mathbf{N}\text{-}\mathbf{1}_k) = 1 - p_i((\text{floor})_{ik}, \mathbf{N}\text{-}\mathbf{1}_k) \qquad (2.73)$$

$$p_i(j, \mathbf{N}\text{-}\mathbf{1}_k) = 0 \quad \text{for all other } j. \qquad (2.74)$$

For example, if $L_i(\mathbf{N}\text{-}\mathbf{1}_k) = 2.8$ then,

$(\text{floor})_{ik} = 2$ and $(\text{ceiling})_{ik} = 3$

$p_i(2, \mathbf{N}\text{-}\mathbf{1}_k) = 3 - 2.8 = 0.2$

$p_i(3, \mathbf{N}\text{-}\mathbf{1}_k) = 1 - 0.2 = 0.8$

$p_i(j, \mathbf{N}\text{-}\mathbf{1}_k) = 0 \quad j=0,1,4,\ldots,|\mathbf{N}|-1.$

This probability assignment procedure is very rough and unrealistic; it is the main reason for the large approximation errors at load-dependent centers.

Akyıldız and Bolch (1988) improved this probability assignment procedure by considering a wider range of numbers centered at the mean queue length $L_i(\mathbf{N}\text{-}\mathbf{1}_k)$. The probabilities are distributed over this range according to a weight function which provides a normal distribution of the probability values.

Another approach (Krzesinski and Greyling, 1984) to approximate the marginal probability distribution is to assume that the removal of a single customer from a chain does not change the marginal probability distribution significantly i.e.,

$$p_i(j, \mathbf{N}\text{-}\mathbf{1}_k) \approx p_i(j, \mathbf{N}) \qquad j=0,1,\ldots,|\mathbf{N}|-1. \qquad (2.75)$$

Substituting this on the RHS of (2.32) we get

$$\begin{aligned} p_i(j,\mathbf{N}) &= \frac{1}{\mu_i(j)} \sum_{r=1}^{R} \tau_{ir}\, \lambda_{ir}(\mathbf{N})\, p_i(j-1,\mathbf{N}) \\ &= \frac{1}{\mu_i(j)}\, U_i(\mathbf{N})\, p_i(j-1,\mathbf{N}) \qquad j=1,..,|\mathbf{N}|. \qquad (2.76) \end{aligned}$$

where

$$U_i(\mathbf{N}) = \sum_{r=1}^{R} \tau_{ir} \lambda_{ir}(\mathbf{N})$$

Given an estimate of $U_i(\mathbf{N})$, $p_i(j,\mathbf{N})$ values can be written in terms of $p_i(0,\mathbf{N})$ by using the equation (2.76) recursively. Then, $p_i(0,\mathbf{N})$ can be calculated by summing all $p_i(j,\mathbf{N})$' s to 1.

CHAPTER III
IMPLEMENTATIONAL ISSUES

In this chapter, different implementations of the MVA and the MVAC algorithms are presented. Each implementation aims to overcome the difficulties faced during the previous implementation by using different strategies.

3.1. Implementation Issues of the MVA Algorithm

The original MVA algorithm (Reiser and Lavenberg,1980) represents the mean queue length at center i as $n_i(\mathbf{n})$ which can be stored in an R+1 dimensional array; with one dimension for the centers i and R dimensions for the population vector **n** when only SSFR and IS centers are allowed. The storage requirement for the marginal probability of j customers at center i when the population vector is **n**, $p_i(j,\mathbf{n})$, is an R+2 dimensional array; one additional dimension for the number of customers j at center i.

The MVA algorithm is implemented and coded in FORTRAN which allows at most seven dimensions. This limitation restricts the implementation to networks with six types of customers when only SSFR and IS centers are allowed. This number is five for networks which allow also load dependent centers. We will call this regular implementation of the MVA algorithm MVA-1 and its code is given in Appendix A.

A partial solution to this problem may be to divide for example an 8-dimensional array into a number of 7-dimensional arrays with respect to one of the dimensions. This division or partition requires additional condition statements in the code at each time that array is referenced. For this reason, the choice of the dimension for partition becomes important and the dimension with the lowest size is a good choice for partition. This method can be applied more than once to handle higher dimensional arrays: for example, twice for a 9-dimensional array and three times for a 10-dimensional array and so on. However, it may effect significantly the time complexity of the algorithm due to the large number of comparisons involved in the IF statements. An implementation of this method will be called MVA-2 and its FORTRAN code is given in Appendix B.

The limitations discussed above can be relaxed if another programming language permitting several dimensions is used. In this case, networks with more than six types of customers can be handled. However, since the total memory used in array storage is constant in a computer environment there will necessarily be a reduction in the sizes of dimensions. In MVA terms, this means that a network with seven types of customers for example, can be analyzed at the expense of giving up some of the customers. Additionally, this array has no sparsity property since most of the dimensions are used to denote the state of the corresponding network.

Another way to overcome this limitation on the array dimensions is to change the order of evaluation of networks **n** ($\mathbf{0}\leq\mathbf{n}\leq\mathbf{N}$) : The MVA algorithm for closed multichain networks consists of generation and evaluation of all networks **n** in some suitable order. The simplest way to obtain all **n**' s is to use R nested loops one for each chain with loop index ranging from zero to the maximum chain population. This procedure is suggested by (Reiser and Lavenberg, 1980). In using this nested loop structure, it is necessary to store all queue lengths. If we consider the total population of a network **n** as its level, the MVA equation for the mean waiting time (2.33) requires only the mean queue lengths of networks with one level less (not all of the previous queue lengths). In order to make use of this feature of the MVA recursion, we can rearrange the order of evaluation of lower level networks such that the evaluation occurs level by level from level 1 up to level $|\mathbf{N}|$. That is, first all the networks of level 1 are generated, then all networks of level 2 and then level 3 up to level $|\mathbf{N}|$. In this way, the evaluation of a level k network requires only the mean queue lengths of networks at level k-1. Thus, the mean queue lengths of networks at previous levels (i.e. k-2, k-3,..,1) are not required and the memory space allocated for these values can be reused during the evaluation of higher level networks. The procedure which generates all lower level networks to **N** is provided by Lavenberg (1983) :

```
For n=1 to $|\mathbf{N}|$
          For $n_1 = \max(0, n-(N_2+..+N_R))$ to $\min(n,N_1)$
              For $n_2 = \max(0, n-(n_1+N_3+..+N_R))$ to $\min(n-n_1,N_2)$
                                    .
                                    .
                                    .
              For $n_r = \max(0, n-(n_1+..+n_{r-1}+N_{r+1}+..+N_R))$ to
                                              $\min(n-n_1-..-n_{r-1}, N_r)$
              For $n_{R-1} = \max(0, n-(n_1+..+n_{R-2}+N_R))$ to
                                              $\min(n-n_1-..-n_{R-2}, N_{R-1})$
              $n_R = n - (n_1+..+n_{R-1})$
```

Figure 3.1. illustrates the sequences of evaluation on an example network **N** , **N**=(2,1,1). The numbers on the left indicate the sequence used by Reiser and Lavenberg (1980) and the numbers on the right show the sequence proposed by Lavenberg (1983).

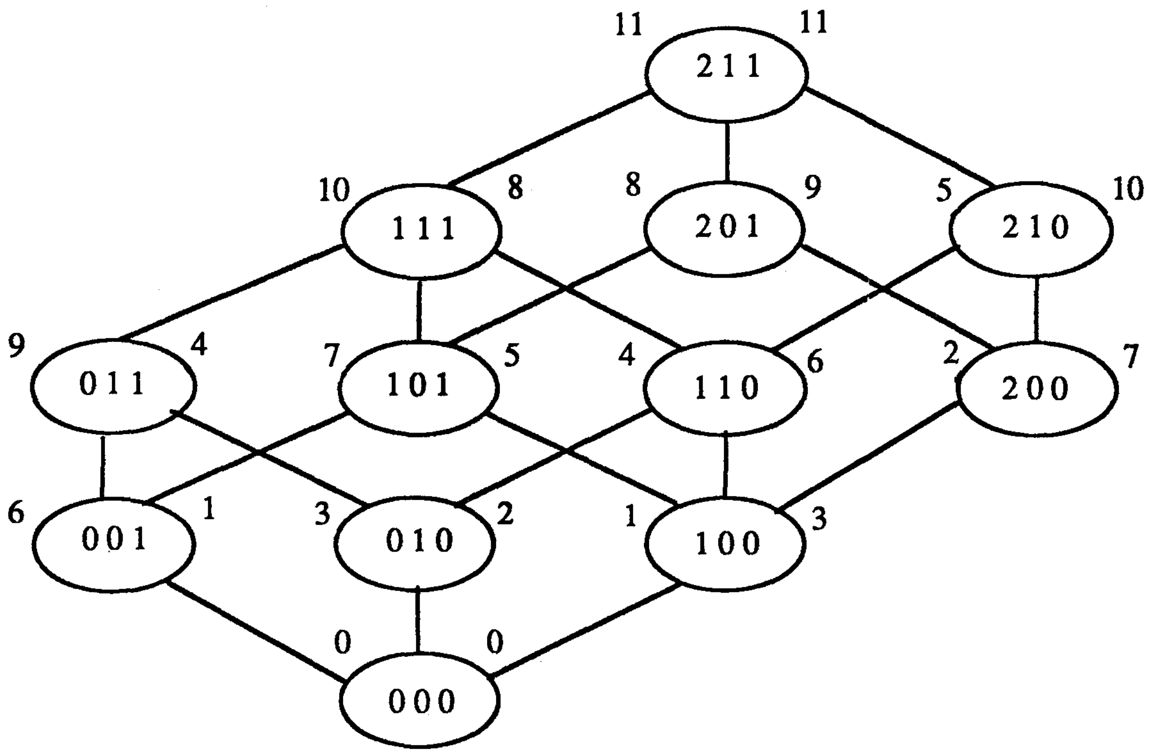


Figure. 3.1. Sequence of evaluation on an example network

Notice that the sequence proposed by Lavenberg (1983) generates the networks within the same level in the lexicographically increasing order. Although a saving in storage is possible, this does not solve the problem of dimensionality mentioned in the beginning of this section as long as the data structure is kept the same. A way to overcome this problem, is to generate and store all possible lower level networks in a two dimensional array VV. This array has $\prod_{r=1}^{R} (N_r+1)$ rows and R columns. A particular lower level network is referenced by its position (i.e. row number) in that array. In this way, only one dimension can be used to reference a network. Note that this procedure is equivalent to numbering all lower level networks in the order of evaluation. Evaluating the MVA equations in this fashion requires less storage and permits to implement an arbitrary number of chains. However, this data structure gives rise to another problem: Most of the networks at a level k do not require the queue lengths of all the networks at level k-1. We can observe this from Figure 3.1. in which all networks at levels 2 and 3 are of this type. It is also hard to find a functional relation between the position of a level k network and the positions of the related level k-1 networks (i.e., whose queue lengths are required). We could handle this problem only in a costly and inefficient way: Given a network at level k (or its position) all of its related networks at level k-1 can be generated easily (by removing a customer from a non-empty chain at a time) and the positions of these related level k-1 networks are found by a sequential search over the part of the array VV occupied by the level k-1 networks. If we consider the mean number of comparisons to find the position of one related level k-1 network to be roughly :

(total number of level k-1 networks / 2) * R

and the total number of comparisons required for the evaluation of a single level k network becomes approximately :

(number of related level k-1 networks) * (total number of level k-1 networks) * (R / 2)

Noting that the first term is at most R then,

$\leq$ (total number of level k-1 networks) * (R^2 / 2)

The large number of comparison required for this implementation makes it undesirable for the analysis of networks with a large number of chains. This implementation is coded in FORTRAN and given in Appendix C under the title MVA-3.

3.2. Implementation Issues of the MVAC Algorithm

The major source of difficulty in the implementation of the MVAC algorithm lies in the representation of the vector ν. This vector can be named as the customer distribution vector of the complement chain or shortly ' the distribution vector ' .

A first implementation makes use of multi dimensional arrays to store the variables of the algorithm. The variable with the largest number of arguments (also requiring the largest memory space) is $L_{il}^{r}(\boldsymbol{\nu})$ which can be stored in a M+3 dimensional array where M is the number of centers. Since FORTRAN allows at most 7-dimensions for a variable, this implementation is limited by the number of centers and allows at most four centers. It should be remembered that the MVAC algorithm transforms the network into a single customer per chain network equating the number of chains to the total number of customers in the network. So, the memory space required for the variable $L_{il}^{r}(\boldsymbol{\nu})$ alone is $M * R^{M+2}$. This large storage requirement usually restricts the networks to be analyzed to small values of R with $M \leq 4$. We will call this implementation of the MVAC algorithm as MVAC-1 and its FORTRAN code is given in Appendix D.

A second implementation of the MVAC algorithm is devised in this study, to allow more than four centers with acceptable levels of storage requirements. In this implementation all distribution vectors forming the sets $I_1, I_2, .., I_R$ are generated and stored in a two-dimensional array. This array has $\sum_{r=1}^{R} \binom{M+(R-r)-1}{M-1} = \binom{M+R-1}{M}$ rows and M columns. Call this matrix V. Each row in this matrix corresponds to a single distribution vector $\boldsymbol{\nu}$. This data structure makes possible to represent a specified distribution vector $\boldsymbol{\nu}$ by referring to only its position (i.e. row number) in the matrix V rather than specifying each component of $\boldsymbol{\nu}$ in a separate dimension. Here, we present the generation procedure of all $\boldsymbol{\nu}$'s :

```
Step 1. (Initialization)
Assign the $\boldsymbol{\nu}$'s in $I_R$ and $I_{R-1}$ to the first R+1 positions of matrix V such
that an identity matrix is formed between the positions 2 and R+1.

Step 2.
For each $r = R$ down to 3
    For each $\boldsymbol{\nu} \varepsilon I_{r-1}$
```

Generate new $\boldsymbol{\nu}$'s in I_{r-2} (i.e., $\boldsymbol{\nu}+\mathbf{1}_i$'s) where i ranges from k to M and k is the position (column) of the rightmost non-zero component of $\boldsymbol{\nu}$.
Append newly generated $\boldsymbol{\nu}$'s to the matrix V.

This procedure generates all $\boldsymbol{\nu}$'s in the set I_r from the previous $\boldsymbol{\nu}$'s in I_{r+1}. Moreover, every vector $\boldsymbol{\nu}$ in I_{r+1} is generated from a unique $\boldsymbol{\nu} \in I_r$ and note that within each set I_r r=1,..,R. $\boldsymbol{\nu}$'s are located in a lexicographically decreasing order in the matrix V.

During the implementation of this data structure to MVAC algorithm (allowing only SSFR and IS centers) equation (7) requires $L_i^{r-1}(\boldsymbol{\nu}+\mathbf{1}_j)$ values to compute $L_i^r(\boldsymbol{\nu})$, where $\boldsymbol{\nu} \in I_r$. This means that we need to know the position of the vector $\boldsymbol{\nu}+\mathbf{1}_j$ given the position of $\boldsymbol{\nu}$ in the matrix V. However, it is quite difficult to find a relation between the positions of vectors $\boldsymbol{\nu}$ and $\boldsymbol{\nu}+\mathbf{1}_j$ for all j=1,...,M. and this difficulty increases if M is large. Therefore, it is necessary to find these positions of $\boldsymbol{\nu}+\mathbf{1}_j$ through a search over the set I_{r-1} making the implementation inefficient and costly. The total number of comparisons required for only the first iteration of MVAC algorithm (i.e. when r=1) is about

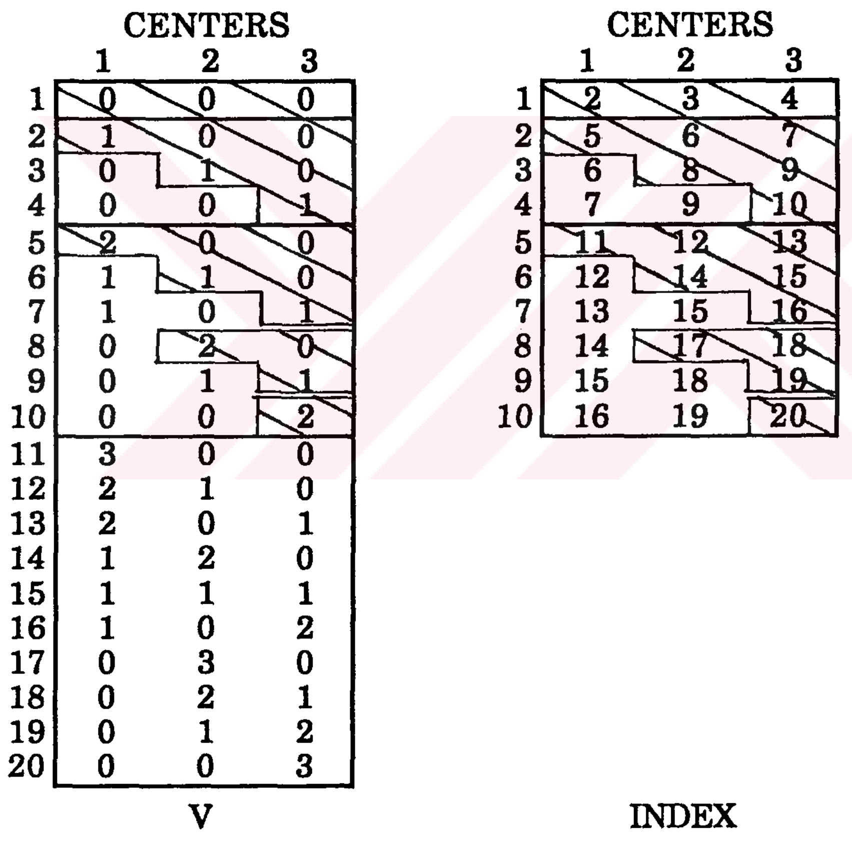
$$M^2 / 2 * \sum_{k=1}^{R} \binom{M+(R-k)-1}{M-1} \binom{M+R-k}{M-1} \qquad (3.1)$$

and this computational cost may become very high even if the network has moderate sizes in R and in M. This constitutes the main drawback of this implementation. The FORTRAN code of this implementation is given in Appendix E under the title MVAC-2.

In yet another implementation, we attempted to eliminate the need for a search which makes the implementation impractical. To do this, an auxiliary two-dimensional array is generated which contains the positions of the vectors $\boldsymbol{\nu}+\mathbf{1}_j$ given the position of vector $\boldsymbol{\nu}$ in V and j. We call this matrix as the INDEX matrix. Appearently, it has the same sizes as the matrix V. However, we will not need the part of INDEX corresponding to $\boldsymbol{\nu}$'s in I_1 since all $\boldsymbol{\nu}+1_j$'s will fall in I_0 and $L_i^0(\boldsymbol{\nu}) = 0$ for all $\boldsymbol{\nu}$ and i. If we can obtain this index matrix before the MVAC iterations without any search, this will provide for an efficient implementation of the MVAC algorithm in terms of computational cost relative to the previous implementation. Now at this point the problem reduces to obtaining the index matrix as efficiently as possible.

We partition the matrix V into two regions :

Region I covers the entries of V from which new $\boldsymbol{\nu}$'s are generated by the generation procedure explained previously. The shaded area in Figure 3.2. shows the region I. Region II consists of entries of V which are not covered by region I (unshaded region).



V

	CENTERS 1	2	3
1	0	0	0
2	1	0	0
3	0	1	0
4	0	0	1
5	2	0	0
6	1	1	0
7	1	0	1
8	0	2	0
9	0	1	1
10	0	0	2
11	3	0	0
12	2	1	0
13	2	0	1
14	1	2	0
15	1	1	1
16	1	0	2
17	0	3	0
18	0	2	1
19	0	1	2
20	0	0	3

INDEX

	CENTERS 1	2	3
1	2	3	4
2	5	6	7
3	6	8	9
4	7	9	10
5	11	12	13
6	12	14	15
7	13	15	16
8	14	17	18
9	15	18	19
10	16	19	20

Figure 3.2. Partition of matrices according to the generation procedure

If we examine the index matrix closely, we can observe a pattern which covers region I : The entries in region I of the index matrix are all consecutive numbers from 2 up to $\binom{M+R-1}{M}$. This is something that can be expected because actually these numbers show the order of generation and appendix of new $\boldsymbol{\nu}$'s during the generation procedure. This allows us to fill the region I of the index matrix during the generation procedure of new $\boldsymbol{\nu}$'s.

The entries in region II of the index matrix show some possibly combinatorial behaviour that we could not identify for arbitrary values of M. Therefore, we restrict the networks in question to only three centers (M=3) to simplify the problem.

Let k be the column number from which the generation has started (i.e., the rightmost non-zero entry on the current row in matrix V). k can take values of 1, 2 and 3 with the above assumption.

We can modify Step 2 of the generation procedure as follows:

```
Step 2.
For each $r = R$ down to 3
    For each $\boldsymbol{\nu} \varepsilon I_{r-1}$
        Generate new $\boldsymbol{\nu}$'s in $I_{r-2}$ (i.e., $\boldsymbol{\nu}+1_i$'s) where i ranges from
        k to M and k is the position (column) of the rightmost
        non-zero component of $\boldsymbol{\nu}$.
        If k= 1 then
            Append newly generated $\boldsymbol{\nu}$'s to the matrix V.
            Keep the position of the appended vector in index
            matrix.
        If k = 2 then
            Assign INDEX(.,1) = (Position of the last appended
            vector to V) - (number of times new $\boldsymbol{\nu}$'s are generated
            with k=2 in $I_{r-1}$).
            Append newly generated $\boldsymbol{\nu}$'s to the matrix V.
            Keep the position of the appended vector in index
            matrix.
        If k = 3 then
            Append newly generated $\boldsymbol{\nu}$ (i.e. $\boldsymbol{\nu}+1_3$) to the matrix V.
```

```
Keep the position of the appended vector in index
matrix.
Assign INDEX(.,1) = (Position of the last appended
vector to V) - (number of times new $\boldsymbol{\nu}$'s are generated
with k=2 in $I_{r-1}$) - 2.
Assign INDEX(.,2) = Position of the last appended
vector to V) - 1.
```

The above procedure is based on a pattern observation between matrices V and INDEX while no proof of it could be found. However, all the runs made with this implementation produced the same output obtained using the previous implementations. This implementation is also coded in FORTRAN and the code is given in Appendix F. This implementation will be referred to as MVAC-3 in the next chapter.

CHAPTER IV
EXPERIMENTAL RESULTS AND CONCLUSIONS

This chapter reports on the experiments and their results performed on the MVA and the MVAC algorithms and on a widely used heuristic called the Linearizer.

4.1. Experiments on MVA Based Exact Algorithms

In the following the first implementation of the MVAC algorithm will be referred to as MVAC-1, the second implementation as MVAC-2, and the third implementation as MVAC-3. The same nomenclature is used for MVA implementations as stated in Chapter III. The experiments in this section are performed on the same networks so that a comparison between algorithms is made possible.

4.1.1. Description of Experiments

All the experiments on the MVAC and the MVA algorithms involve product form networks with only SSFR service centers. Other center types are not considered in the experiments since the presence of such centers would necessarily require additional variables such as probability values and therefore extra memory. This would probably bring about storage problems since these algorithms, especially MVAC, requires a large amount of computer memory.

All implementations of the algorithms are tested on the same networks and the same mean performance measures are obtained. Furthermore, some of the works on approximations (e.g., Neuse and Chandy,1981; Chandy and Neuse,1982; Akyildiz and Bolch,1988) report also the exact solutions of example networks. The results obtained from our implementations coincide with the reported solutions which makes us confident on the correctness of the implementations.

A total of 134 experimental runs are made. Algorithms are coded using the FORTRAN programming language in IBM 3090/180S mainfraim. Each implementation is run with the same network parameters. The number of chains (R) ranges from two to eight for the MVAC algorithm and to six for the MVA algorithm and the number of customers per chain (N) ranges from 1 to 9 for the MVAC algorithm and 11 for the MVA algorithm.

During the experimental runs, total CPU times are observed and tabulated in the following :

Table 4.1. CPU Times of MVAC-1 Implementation in secs.

M=3	N=1	N=2	N=3	N=4	N=5	N=6	N=7	N=8	N=9
R=2	0.033	0.033	0.042	0.064	0.111	0.207	0.375	0.655	*
R=3	0.034	0.044	0.087	0.208	0.491	*	-	-	-
R=4	0.037	0.073	0.225	*	-	-	-	-	-
R=5	0.045	0.144	0.568	*	-	-	-	-	-
R=6	0.063	0.291	*	-	-	-	-	-	-
R=7	0.094	0.573	*	-	-	-	-	-	-
R=8	0.150	*	-	-	-	-	-	-	-

Table 4.2. CPU Times of MVAC-2 Implementation in secs.

M=3	N=1	N=2	N=3	N=4	N=5	N=6	N=7	N=8	N=9
R=2	0.032	0.036	0.062	0.177	0.556	1.586	4.011	9.149	19.86
R=3	0.034	0.065	0.327	1.644	6.358	19.88	53.0	-	-
R=4	0.042	0.188	1.650	9.49	38.78	-	-	-	-
R=5	0.063	0.598	6.38	38.79	-	-	-	-	-
R=6	0.116	1.7	19.9	-	-	-	-	-	-
R=7	0.233	4.3	53.2	-	-	-	-	-	-
R=8	0.470	9.8	125.6	-	-	-	-	-	-

Table 4.3. CPU Times of MVAC-3 implementation in secs.

M=3	N=1	N=2	N=3	N=4	N=5	N=6	N=7	N=8	N=9
R=2	0.032	0.032	0.033	0.034	0.039	0.047	0.059	0.077	0.103
R=3	0.032	0.033	0.037	0.047	0.067	0.103	0.159	0.243	*
R=4	0.032	0.035	0.048	0.079	0.141	0.246	*	-	-
R=5	0.034	0.041	0.068	0.141	0.285	*	-	-	-
R=6	0.035	0.050	0.106	0.249	*	-	-	-	-
R=7	0.038	0.064	0.167	*	-	-	-	-	-
R=8	0.043	0.086	0.256	-	-	-	-	-	-

Table 4.4. CPU Times of MVA -1 Implementation (in secs.)

M=3	N=1	N=3	N=5	N=7	N=9	N=11
R=2	0.030	0.031	0.031	0.032	0.033	0.033
R=3	0.031	0.033	0.036	0.043	0.054	0.072
R=4	0.032	0.038	0.068	0.151	0.329	0.655
R=5	0.033	0.067	0.300	0.622	1.196	2.154
R=6	0.033	0.172	1.987	11.41	*	*
r	-	0.91	0.94	0.95	0.98	0.98

Table 4.5. CPU Times of MVA-2 Implementation

M=3	N=1	N=2	N=3
R=7	0.040	0.7173	-
R=8	0.055	6.14	*

Table 4.6. CPU Times of MVA-3 Implementation

M=3	N=1	N=2	N=3
R=7	0.038	0.1525	1.023
R=8	0.044	0.4532	4.702

4.1.2 Comparison of Algorithms and Implementations

As seen from the above Tables 4.1. - 4.6. not all of the cases could be run successfully. The *'s in the tables indicate that the network in question is beyond the memory capacity of the computer. All these cases, require more than 4 MB. of memory space which is the largest memory size that the FORTRAN compiler can address.

MVAC implementations suffer from this a great deal because the actual number of chains that the MVAC algorithm considers is not R but N*R. This effect is seen clearly in tables 4.1.-4.3. Therefore, the available memory space is the most important requirement for the MVAC algorithm. We observe a gradual enlargement in the problem sizes that can be solved as we go from implementation 1 to 3.

The results of the experimental runs justify that the computational cost of the MVA algorithm increases exponentially with the number of chains in the network when the number of centers (M) and the chain populations (N) are held constant. The last row of Table 4.4. shows the correlation coefficient, r, between the CPU times required by the MVA algorithm and the number of chains when N is kept constant. The CPU times observed are the total times and include I/O times. We can have r values nearer to 1 when the I/O times are disregarded. In addition, this exponential increase becomes clearer as N gets larger. Unfortunately, the MVA algorithm can not be implemented for R>6 efficiently and restricts the observation range over R for fixed N values. On the other hand, the computational cost of the MVAC algorithm grows only as a power of R when M and N are held constant implying a polynomial time complexity. But there still exist storage problems in reaching the region in which MVAC becomes advantageous to MVA. However, this can be handled with a professional coding which makes use of the recursive nature of algorithms and a more suitable programming language allowing several dimensions or recursive procedures.

MVAC-1 is the implementation which requires the most memory space. This can be expected since it uses a separate dimension for each entry of the vector $\boldsymbol{\nu}$. It can be applied only to networks with small N values. However, the CPU times are much shorter than MVAC-2 and comparable with MVAC-3 implementation since the access to individual $\boldsymbol{\nu}$' s is directly performed.

MVAC-2 can be applied across a larger region than MVAC-1 but it requires excessive CPU times to evaluate the networks that can not be evaluated by MVAC-1. The main reason for the large CPU times is the search of $\boldsymbol{\nu}+\mathbf{1}_j$'s j=1,..,M. during the MVAC iterations.

MVAC-3 which is introduced in chapter 3, has the smallest CPU times among MVAC implementations and it is the only implementation that can compete with MVA implementations. However, this last implementation can be applied to networks consisting of only three service centers. It is possible to extend this implementation to cover more than three service centers.

4.2. Experiments on the Linearizer Algorithm

The Linearizer Algorithm is one of the approximation methods which performs well on most networks tested in the literature. This section deals with the original Linearizer algorithm evaluating its accuracy based on a set of randomly generated test networks. The purpose of the experiments is to analyze the accuracy of the Linearizer algorithm.

The experiments consist of applying the Linearizer algorithm to randomly generated test networks and comparing the results with the exact values obtained by the MVA algorithm. All the centers are assumed to be of load-independent type. The network generation parameters are as given in Table 4.7. where U denotes the uniform distribution.

Table 4.7. Network Generation Parameters

Population Size (N_r) :	U(1,10) for each chain
Loading (ρ_{ir}) :	U(0.1,50.0)
No. of Centers (M) :	U(2,7)

Since the Linearizer algorithm works with reduced networks, we analyzed the networks into five groups according to the number of chains from one to five. 100 random networks are generated from each group. Each randomly generated network is solved first exactly by using the MVA algorithm and then approximately the Linearizer algorithm.

The queue length tolerance error (L-tolerance) is considered as the main measure of comparison. L-tolerance is defined as follows (Chandy et al.,1975) :

$$\text{L-tolerance} = \underset{\text{over all } i,r}{\text{Max}} \; \{ \mid L_{ir}^{ex} - L_{ir}^{app} \mid / N_r \}$$

L-tolerance measures how well the heuristic determines the fraction of each chain' s population at each center. Some statistics on the L-tolerance errors are given in Table 4.8.

Table 4.8. Statistics on L-Tolerance Values (X 100)

	R=1	R=2	R=3	R=4	R=5*
Ave. L-tolerance	0.1	1.7	3.1	3.6	3.7
St. Dev.	0.12	2.9	3.5	3.2	3.0
Max L-tolerance	0.50	35.15	40.23	36.51	48.68

* In six of the experiments in this group exact solutions could not be obtained due to the memory/time requirements.

As seen from Table 4.8. average values of L-tolerance errors are quite low although the maximum values are considerably high. In fact the Linearizer may give even larger errors in the case of stress networks. We can conclude from the results of the runs that a large portion of networks tested are approximated quite well and a very small part causes large errors. This remark is valid for each group of network tested.

Although there still exists studies to give upper bounds for the error made by the Linearizer there isn't any result reported in literature. Similarly, no study was made to identify a stress network from its parameters without solving it. Another point is that the Linearizer algorithm can not be tested for networks with many chains and therefore its performance in these cases is unknown. These issues need to be further investigated in the future.

4.3. Conclusions

In general, from the users' point of view the answer to the question which algorithm is the algorithm of choice given the parameters of a network gains importance. When a choice has to be made between the MVA and the MVAC algorithms the following points may help in making this choice: For networks having less than or equal to six chains, the MVA algorithm is very practical and provide efficient solutions when chain populations are not large. However, this is not possible when there are more than six chains and in these cases the MVAC algorithm becomes relatively efficient. Another point is that when the chain populations are large the MVAC algorithm becomes more efficient.

In this study, a comprehensive survey of MVA based exact algorithms for product-form queueing networks is presented together with the heuristic extensions. The difficulties of finding exact solutions are discussed by means of different implementations. The Linearizer algorithm derived from MVA seems to perform well on most networks and can be used effectively to analyse such networks only after the investigation of the issues mentioned in section 4.2.

Usually, the essential problem in the implementation of exact algorithms turns out to be the high storage requirements. This problem prevented us from experimenting on networks which exploit the relative advantages of the MVA and MVAC algorithms. However, the storage problem can be reduced by making use of the recursive nature of exact algorithms and a more suitable programming language allowing several dimensions or recursive procedures.

REFERENCES

Akyıldız, I.F., and Bolch, G.,1988. "Mean Value Analysis Approximation for multiple server queueing networks", Performance Evaluation, Vol.8, pp. 77-91.

Bard, Y., 1979. "Some extensions to multiclass queueing network analysis", Performance of Computer Systems, M. Arato, A. Butrimenko and E. Gelenbe (Eds.), North-Holland, pp.51-61.

Bard, Y., 1980. "A model of shared DASD and multipathing", Communications of Association for Computing Machinary, Vol.23, No.10, pp.564-572.

Baskett, F., Chandy, K.M., Muntz, R.R., and Palacios, F.G., 1975. "Open, closed, and mixed networks of queues with different classes of customers", Journal of the Association for Computing Machinary, Vol.22, No.2, pp. 248-260.

Bruell, S.C., Balbo, G., and Afshari, P.V., 1984. "Mean value analysis of mixed, multiple class BCMP networks with load dependent service stations", Performance Evaluation, Vol.4, No.4, pp. 241-260.

Burke, P.J., 1956. "The output of a queuing system", Operations Research, Vol.4, pp. 699-704.

Buzen, J.P., 1973. "Computational algorithms for closed queueing networks with exponential servers", Communications of Association for Computing Machinary, Vol.16, No.9, pp. 527-531.

Chandy, K.M., 1972. "The analysis and solutions for general queueing networks", Proc. Sixth Annual Princeton Conference on Information Sciences and Systems, Princeton U., Princeton, N.J., U.S.A.,pp.224-228.

Chandy, K.M., Herzog, U., and Woo, L.S., 1975. "Approximate analysis of general queueing networks", IBM Journal of Research and Development, Vol.19, No.1, pp.43-49.

Chandy, K.M., Howard, J.H., and Towsley, D.F., 1977. "Product form and local balance in queueing networks", Journal of the Association for Computing Machinary, Vol.24, No.2, pp. 250-263.

Chandy, K.M., and Sauer, C.H., 1980. "Computational algorithms for product form queueing networks", Communications of Association for Computing Machinary, Vol.23, No.10, pp.573-583.

Chandy, K.M., and Neuse, D., 1982. "Linearizer: A heuristic algorithm for queueing network models of computer systems", Communications of Association for Computing Machinary, Vol.25, No.2, pp. 126-133.

Chow, W-M., 1983. "Approximations for large scale closed queueing networks", Performance Evaluation, Vol.3, pp.1-12.

Conway, A.E., 1986. "A polynomial complexity MVA algorithm for multiple chain closed queueing networks", IEEE Int. Symp. Inform. Theory Dig. Papers, Ann Arbor, M.I., U.S.A.

Conway, A.E., and Georganas, N.D., 1986. "Decomposition and Aggregation by class in closed queueing networks", IEEE Transactions on Software Engineering, Vol.12, No.10, pp. 1025-1040.

Conway, A.E., and Georganas, N.D., 1986. "RECAL-A New Efficient Algorithm for the Exact Analysis of Multiple-Chain Closed Queuing

Networks", Journal of the Association for Computing Machinary, Vol.33, No.4, pp. 768-791.

Conway, A.E., Silva, Edmundo de Souza e , Lavenberg, S.S., 1989. "Mean Value Analysis by Chain of Product Form Queueing Networks", IEEE Transactions on Computers, Vol.38, No.3, pp. 432-442.

Eager, D.L., and Sevcik, K.C., 1984. "An analysis of an approximation algorithm for queueing networks", Performance Evaluation, Vol.4, pp. 275-284.

Ferdinand, A. E., 1971. "An analysis of machine interference model", IBM Sys. J., Vol.10, pp.129-142.

Gelenbe, E. and Mitrani, I., 1980. Analysis and Synthesis of Computer Systems, Academic Press, New York.

Gordon, W.J., and Newell, G.F., 1967. "Closed queueing networks with exponential servers", Operations Research, Vol.15, No.2, pp.254-265.

Greenberg, A.G., and McKenna J., 1989. "Solution of Product-Form Closed Queueing Networks Via the RECAL and Tree-RECAL Methods on a Shared Memory Multiprocessor", ACM Sigmetrics, pp. 1178-1186.

Hoyme, K.P., Bruell, S.C., Afshari, P.V. and Kain, R.Y., 1986. "A Tree Structured Mean Value Analysis Algorithm", ACM Transactions on Computer Systems, Vol.4, pp. 178-185.

Hsieh, C.T., and Lam, S.S., 1989. "Pam-A noniterative approximate solution method for closed multichain queueing networks", Performance Evaluation, Vol.9, pp.119-133.

Jackson, R.R.P., 1954. "Queueing Systems with Phase Type Service", Operations Research Quarterly, Vol.5, pp. 109-120.

Jackson, J.R., 1957. "Networks of waiting lines", Operations Research, Vol.5, No.4, pp.518-521.

Jackson, J.R., 1963. "Jobshop-like queueing systems", Management Science, Vol.10, No.1, pp.131-142.

Kant, K., 1992. Introduction to Computer System Performance Evaluation, Mc Graw-Hill Inc., U.S.A.

Kaylan, A.R., 1990. "Queueing Networks: A Survey of Analytical Results", Queueing Theory and Applications, S. Özekici(Ed.), Hemisphere Publishing Cooperation, U.S.A., pp. 80-98.

Koenigsberg,E., 1958. "Cyclic queues", Operational Reaserch Quarterly, Vol,9, pp.22-35.

Krzesinski, A., and Greyling, J., 1984. "Improved Linearizer methods for queueing networks with queue dependent centres", Journal of Association for Computing Machinary, pp. 41-51.

Lam, S.S., and Lien, Y.L., 1983. "A Tree Convolution Algorithm for the Solution of Queueing Networks", Communications of Association for Computing Machinary, Vol.26, No.10, pp. 203-215.

Lavenberg, S.S., and Reiser, M., 1980. " Stationary State Probabilities at Arrival Instants for Closed Queueing Networks with Multiple Type of Customers", Journal of Applied Probability, Vol. 17, pp 1048-1061.

Lavenberg, S.S., 1983. Computer Performance Modelling Handbook, Academic Press, New York.

Little, J.D.C., 1961. "A proof of the queueing formula $L=\lambda W$", Operations Research, pp.383-387.

Muntz, R.R., 1972. "Poisson departure processes and queueing networks", IBM Res. Rep. RC-4145, Yorktown Heights, N.Y.

Neuse, D., and Chandy, K.M., 1981. "SCAT: A heuristic algorithm for queueing network models of computing systems", ACM SIGMETRICS Conference Proceedings, Vol.10, No.3, pp.59-79.

Noetzel, A.S., 1979. "A Generalized Queueing Discipline for Product-Form Network Solutions", Journal of ACM, Vol.26, No.4, pp. 779-793.

Örmeci, E.L.,1993. "Normalization Constant Based Exact Algorithms for Queueing Networks", M.S. Thesis in Industrial Engineering, Middle East Technical University, Ankara.

Posner, M., and Bernholtz, B., 1968. "Closed Finite Queueing Networks with Time Lags and with Several Classes of Units", Operations Research, Vol. 16, pp. 977-985.

Reiser, M., 1979. "A queueing network analysis of computer communication networks with window flow control", IEEE Transactions on Communications, Vol.27, No.8, pp. 1199-1209.

Reiser, M., 1981. "Mean value analysis and convolution method for queue-dependent servers in closed queueing networks", Performance Evaluation, Vol.1, No.1, pp.7-18.

Reiser, M., and Kobayashi, H., 1975. "Queueing networks with multiple closed chains: Theory and computational algorithms", IBM Journal of Research and Development , Vol.19, No.3, pp.283-293.

Reiser, M., and Lavenberg, S.S., 1980. "Mean-value analysis of closed multichain queueing networks", Journal of the Association for Computing Machinary, Vol.27, No.2, pp. 313-322.

Sauer, C.H., and Chandy, K.M., 1981. Computer Systems Performance Modeling, Prentice-Hall, New Jersey.

Schweitzer, P., 1979. "Approximate analysis of multiclass closed networks of queues", Proceedings International Conference on Stochastic Control and Optimization, Amsterdam.

Sevcik, C.H., and Mitrani, I., 1981. "The Distribution of Queueing Network States at Input and Output Instants", Journal of ACM , Vol.28, pp. 172-184.

Silva, Edmundo de Souza e, Lavenberg, S.S., Muntz, R.R., 1984. "A Perspective on Iterative Methods for the Approximate Analysis of Closed Queueing Networks", Mathematical Computer Performance and Reliability, Hordijk, A. (Ed.) pp. 225-244. North-Holland.

Silva, Edmundo de Souza e, and Lavenberg, S.S., 1986. "A MVA by Chain algorithm for product form queueing networks", IBM Res. Rep. RC 11641, Yorktown Heights, N.Y., U.S.A.

Silva, Edmundo de Souza e, and Muntz, R.R., 1988. "Simple relationship among moments of queue lengths in product form queueing networks", IEEE Transactions on Computers, Vol.37, No.9, pp.1125-1129.

Silva, Edmundo de Souza e, and Lavenberg, S.S., 1989. "Calculating Joint Queue-Length Distributions in Product-Form Queueing Networks", Journal of the Association for Computing Machinary, Vol.36, No.1, pp. 194-207.

Silva, Edmundo de Souza e, and Muntz, R.R., 1990. "A note on the computational cost of the Linearizer algorithm for queueing networks", IEEE Transactions on Computers, Vol.39, No.6, pp.840-842.

Spirn, J.R., 1979. "Queueing networks with random selection for service", IEEE Transactions on Software Engineering, Vol.5, No.3, pp.287-289.

Tucci, S., and MacNair, E.A., 1982. "Implementation of mean-value analysis for open, closed and mixed queueing networks", Computer Performance, Vol.3, No.4, pp. 223-239.

Tucci S., and Sauer, C.H., 1985. "The Tree MVA Algorithm", Performance Evaluation, Vol.5, pp. 187-196.

Zahorjan, J., Eager, D.L., and Sweillam, H.M., 1988. "Accuracy, speed, and convergence of approximate Mean Value Analysis", Performance Evaluation, Vol.8, pp. 255-270.

APPENDICES

APPENDIX A

FORTRAN CODE FOR MVA-1

```
      PROGRAM GENERAL_EXACT_MVA_ALGORITHM
C     FILENAME : CH6A.FOR   ( MVA-1 )
      PARAMETER ( MAXCEN = 10 , MPOP = 15 )
      INTEGER  I(6),II(6),CENTER,CHAIN,Q(MAXCEN),N(6)
      REAL  RO(MAXCEN,6),LAMBDA(6),W(MAXCEN,6),LL(MAXCEN,6),
     $   L(0:MPOP,0:MPOP,0:MPOP,0:MPOP,0:MPOP,0:MPOP,MAXCEN),T1,T2
C     CALL CPUTIME(T1)
      CALL DATA ( K,M,N,Q,RO,MAXCEN )
      DATA I / 6 * 0 /
C ****** MAIN  LOOP  *******
      DO  60   I6 = 0,N(6)
        I(6) = I6
      DO  70   I5 = 0,N(5)
        I(5) = I5
      DO  80   I4 = 0,N(4)
        I(4) = I4
      DO  90   I3 = 0,N(3)
        I(3) = I3
      DO  100  I2 = 0,N(2)
           I(2) = I2
      DO  200  I1 = 0,N(1)
        I(1) = I1
        IF ( I1+I2+I3+I4+I5+I6 .EQ. 0 )  GOTO  200
        DO  300  CHAIN = 1,K
          IF ( I(CHAIN).EQ.0 )   THEN
            LAMBDA(CHAIN) = 0.
          ELSE
            CALL  XX (I,CHAIN,II)  ! FIND REDUCED NETWORK
            SUMW = 0.
            DO  400  CENTER=1,M
       W(CENTER,CHAIN) = RO(CENTER,CHAIN)
     $* ( 1.+Q(CENTER) * L(II(1),II(2),II(3),II(4),II(5),II(6),CENTER))
               SUMW = SUMW + W(CENTER,CHAIN)
 400          CONTINUE
            LAMBDA(CHAIN) = I(CHAIN) / SUMW
          ENDIF
 300    CONTINUE

        DO  500  CENTER = 1,M
          SUM = 0.
          DO  600  CHAIN = 1,K
            LL(CENTER,CHAIN) = LAMBDA(CHAIN)*W(CENTER,CHAIN)
            SUM = SUM + LAMBDA(CHAIN)*W(CENTER,CHAIN)
 600        CONTINUE
          L(I(1),I(2),I(3),I(4),I(5),I(6),CENTER) = SUM
 500    CONTINUE

 200 CONTINUE
 100 CONTINUE
  90 CONTINUE
  80 CONTINUE
  70 CONTINUE
  60 CONTINUE
C *****  END OF MAIN LOOP  ******

     DO 150 CENTER = 1,M
        WRITE (*,2) ( LL(CENTER,CHAIN), CHAIN = 1,K ) !QUEUE LENGTHS
 150 CONTINUE
     WRITE (*,2) ( LAMBDA(CHAIN) , CHAIN = 1,K ) ! CHAIN THROUGHPUTS
   2 FORMAT (3X,6F12.6)
C    CALL CPUTIME(T2)
C    WRITE(*,*) 'CPU TIME = ',(T2-T1)/1000000.
     STOP
     END

     SUBROUTINE DATA ( K,M,N,Q,S,MAXCEN )
     INTEGER N(6),Q(MAXCEN),K,M,MAXCEN
     REAL S(MAXCEN,6)
     OPEN ( 1,FILE='/NET2 DATA A' )
     READ (1,*) K                  ! NO OF CHAINS
     READ (1,*) M                  ! NO OF CENTERS
     READ (1,*) ( Q(I) , I = 1,M )    ! CENTER TYPES
     READ (1,*) ( N(I) , I = 1,K )    ! CHAIN POPULATIONS
     DO 10 I=1,M
        READ (1,*) ( S(I,J) , J=1,K ) ! MEAN SERVICE DEMANDS =
 10  CONTINUE      !  VISIT RATIO * MEAN SERVICE TIME PER VISIT
     RETURN
     END

     SUBROUTINE XX(A,R,B)
     INTEGER A(6),B(6),R
     DO 10 J = 1,6
        IF (J .EQ. R) THEN
          B(R) = A(R) - 1
        ELSE
          B(J) = A(J)
        ENDIF
 10  CONTINUE
     RETURN
     END
```

APPENDIX B

FORTRAN CODE FOR MVA-2

```
C    PROGRAM EXACT_MVA_ALGORITHM
C    FILENAME : MVAB.FOR ( MVA-2 )
     PARAMETER ( MAXCEN =  5 )
     INTEGER  I(8),II(8),CENTER,CHAIN,Q(MAXCEN),N(8),V(0: 10000,8)
    $      , START(0:100),LEND(0:100),C,Z,TOTN
     REAL  RO(MAXCEN,8),LAMBDA(8),W(MAXCEN,8),LL(MAXCEN,8),
    $  L(10000,MAXCEN),T1,T2
C
     CALL CPUTIME(T1)
     CALL DATA ( K,M,N,Q,RO,MAXCEN )
     TOTN = 0
     DO  5  J=1,K
        TOTN = TOTN + N(J)
   5  CONTINUE
     START(0)=0
     LEND(0) =0
     DO 7 J=1,K
        V(0,J)=0
  7   CONTINUE
     COUNT = 0
C ****** MAIN  LOOP  ******
     DO 10 C = 1,TOTN
         START(C) = COUNT+1
         DO  20 I1 = MAX(0,C-(N(2)+N(3)+N(4)+N(5)+N(6)+N(7)+N(8))),
    $                 MIN(C,N(1))
           I(1) = I1
         DO  30 I2 = MAX( 0,C-(I1+N(3)+N(4)+N(5)+N(6)+N(7)+N(8))),
    $                 MIN(C-I1,N(2))
           I(2) = I2
         DO  40 I3 = MAX( 0,C-(I1+I2+N(4)+N(5)+N(6)+N(7)+N(8))) ,
    $                 MIN(C-I1-I2,N(3))
           I(3) = I3
         DO  50 I4 = MAX( 0,C-(I1+I2+I3+N(5)+N(6)+N(7)+N(8))) ,
    $                 MIN(C-I1-I2-I3,N(4))
           I(4) = I4
         DO  60 I5 = MAX( 0,C-(I1+I2+I3+I4+N(6)+N(7)+N(8))) ,
    $                 MIN(C-I1-I2-I3-I4,N(5))
           I(5) = I5
         DO  70 I6 = MAX( 0,C-(I1+I2+I3+I4+I5+N(7)+N(8))) ,
    $                 MIN(C-I1-I2-I3-I4-I5,N(6))
           I(6) = I6
         DO  80 I7 = MAX( 0,C-(I1+I2+I3+I4+I5+I6+N(8))) ,
    $                 MIN(C-I1-I2-I3-I4-I5-I6,N(7))
           I(7) = I7
```

```
      I(8) = C-(I1+I2+I3+I4+I5+I6+I7)
      COUNT = COUNT + 1
      DO  82 J=1,K
         V(COUNT,J) = I(J)
 82     CONTINUE

     DO  300  CHAIN = 1,K
        IF ( I(CHAIN).EQ.0 )   THEN
          LAMBDA(CHAIN) = 0.
        ELSE
          DO 90 J=1,K
            IF ( J.EQ.CHAIN ) THEN
              II(CHAIN) = I(CHAIN)-1
            ELSE
              II(J) = I(J)
            ENDIF
 90         CONTINUE
          DO  100 J=START(C-1),LEND(C-1)
             DO  110  L1=1,K-1
               IF ( V(J,L1).NE.II(L1) )  GO TO 100
110            CONTINUE
             Z = J
             GO TO 120
100         CONTINUE
120         CONTINUE
          SUMW = 0.
          DO  400  CENTER=1,M
    W(CENTER,CHAIN) = RO(CENTER,CHAIN)
  $                 * ( 1.+Q(CENTER) * L(Z,CENTER))
             SUMW = SUMW + W(CENTER,CHAIN)
400         CONTINUE
          LAMBDA(CHAIN) = I(CHAIN) / SUMW
       ENDIF
300    CONTINUE

     DO  500  CENTER = 1,M
        SUM = 0.
        DO  600  CHAIN = 1,K
           LL(CENTER,CHAIN) = LAMBDA(CHAIN)*W(CENTER,CHAIN)
           SUM = SUM + LAMBDA(CHAIN)*W(CENTER,CHAIN)
600       CONTINUE
        L(COUNT,CENTER) = SUM
500    CONTINUE

 80 CONTINUE
 70 CONTINUE
 60 CONTINUE
 50 CONTINUE
 40 CONTINUE
 30 CONTINUE
 20 CONTINUE
   LEND(C) = COUNT

 10 CONTINUE

   DO  150  CENTER = 1,M
     WRITE (*,2) ( LL(CENTER,CHAIN), CHAIN = 1,K )! MEAN QUEUE LENGTHS
150 CONTINUE
   WRITE (*,2) ( LAMBDA(CHAIN) , CHAIN = 1,K ) ! CHAIN THROUGHPUTS
 2 FORMAT (3X,8F 9.6)
```

```
   CALL CPUTIME(T2)
   WRITE (*,*) 'CPU TIME = ',(T2-T1)/1000000.
   STOP
   END

   SUBROUTINE DATA ( K,M,N,Q,S,MAXCEN )
   INTEGER N(8),Q(MAXCEN),K,M,MAXCEN
   REAL S(MAXCEN,8)
   OPEN ( 1,FILE='/NET2 DATA A')
   READ (1,*) K                    ! NO. OF CHAINS
   READ (1,*) M                    ! NO. OF CENTERS
   READ (1,*) ( Q(I) , I = 1,M )     ! CENTER TYPES
   READ (1,*) ( N(I) , I = 1,K )     ! CHAIN POPULATIONS
   DO  10  I=1,M
     READ (1,*) ( S(I,J) , J=1,K )  ! MEAN SERVICE DEMANDS =
10 CONTINUE        ! VISIT RATIO * MEAN SERVICE TIME PER VISIT
   RETURN
   END
```

APPENDIX C

FORTRAN CODE FOR MVA-3

```
      PROGRAM EIGHT_CHAIN_EXACT_MVA_ALGORITHM
C     FILENAME : CH8C.FOR ( MVA-3 )
      PARAMETER ( MAXCEN = 3 )
C     MAXIMUM NUMBER OF CENTERS ALLOWED = 3
C     MAXIMUM NO. OF CUSTOMERS IN FIRST CHAIN, N(1) = 3
C     MAXIMUM NO. OF CUSTOMERS IN OTHER CHAINS = 6
      INTEGER  I(8),II(8),CENTER,CHAIN,Q(3),N(8)
      REAL  RO(MAXCEN,8),LAMBDA(8),W(MAXCEN,8),LL(MAXCEN,8),
     $      L10(0: 6,0: 6,0: 6,0: 6,0: 6,0: 6,0: 6),
     $      L11(0: 6,0: 6,0: 6,0: 6,0: 6,0: 6,0: 6),
     $      L12(0: 6,0: 6,0: 6,0: 6,0: 6,0: 6,0: 6),
     $      L13(0: 6,0: 6,0: 6,0: 6,0: 6,0: 6,0: 6),
     $      L20(0: 6,0: 6,0: 6,0: 6,0: 6,0: 6,0: 6),
     $      L21(0: 6,0: 6,0: 6,0: 6,0: 6,0: 6,0: 6),
     $      L22(0: 6,0: 6,0: 6,0: 6,0: 6,0: 6,0: 6),
     $      L23(0: 6,0: 6,0: 6,0: 6,0: 6,0: 6,0: 6),
     $      L30(0: 6,0: 6,0: 6,0: 6,0: 6,0: 6,0: 6),
     $      L31(0: 6,0: 6,0: 6,0: 6,0: 6,0: 6,0: 6),
     $      L32(0: 6,0: 6,0: 6,0: 6,0: 6,0: 6,0: 6),
     $      L33(0: 6,0: 6,0: 6,0: 6,0: 6,0: 6,0: 6)
      REAL T1,T2
      CALL CPUTIME(T1)
      DATA I / 8 * 0 /
      CALL DATA ( K,M,N,Q,RO,MAXCEN)
C *****  MAIN  LOOP  ******
      DO  40   I8 = 0,N(8)
        I(8) = I8
      DO  50   I7 = 0,N(7)
        I(7) = I7
      DO  60   I6 = 0,N(6)
        I(6) = I6
      DO  70   I5 = 0,N(5)
        I(5) = I5
      DO  80   I4 = 0,N(4)
        I(4) = I4
      DO  90   I3 = 0,N(3)
        I(3) = I3
      DO  100  I2 = 0,N(2)
        I(2) = I2
      DO  200  I1 = 0,N(1)
        I(1) = I1
        IF ( I1+I2+I3+I4+I5+I6+I7+I8 .EQ. 0 )  GOTO  200
        DO  300  CHAIN = 1,K
          IF ( I(CHAIN).EQ.0 )  THEN
            LAMBDA(CHAIN) = 0.
      ELSE
        CALL  XX (I,CHAIN,II) ! FIND A REDUCED NETWORK
        SUMW = 0.
        DO  400  CENTER=1,M
          IF ( CENTER.EQ.1 ) THEN
           IF ( II(1).EQ.0 ) THEN
      W(CENTER,CHAIN) = RO(CENTER,CHAIN)
     $* ( 1.+Q(CENTER) * L10(II(2),II(3),II(4),II(5),II(6),II(7),II(8)))
           ELSE IF ( II(1).EQ.1 ) THEN
      W(CENTER,CHAIN) = RO(CENTER,CHAIN)
     $* ( 1.+Q(CENTER) * L11(II(2),II(3),II(4),II(5),II(6),II(7),II(8)))
           ELSE IF ( II(1).EQ.2 ) THEN
      W(CENTER,CHAIN) = RO(CENTER,CHAIN)
     $* ( 1.+Q(CENTER) * L12(II(2),II(3),II(4),II(5),II(6),II(7),II(8)))
           ELSE IF ( II(1).EQ.3 ) THEN
      W(CENTER,CHAIN) = RO(CENTER,CHAIN)
     $* ( 1.+Q(CENTER) * L13(II(2),II(3),II(4),II(5),II(6),II(7),II(8)))
           ENDIF
          ELSE IF ( CENTER.EQ.2 ) THEN
           IF ( II(1).EQ.0 ) THEN
      W(CENTER,CHAIN) = RO(CENTER,CHAIN)
     $* ( 1.+Q(CENTER) * L20(II(2),II(3),II(4),II(5),II(6),II(7),II(8)))
           ELSE IF ( II(1).EQ.1 ) THEN
      W(CENTER,CHAIN) = RO(CENTER,CHAIN)
     $* ( 1.+Q(CENTER) * L21(II(2),II(3),II(4),II(5),II(6),II(7),II(8)))
           ELSE IF ( II(1).EQ.2 ) THEN
      W(CENTER,CHAIN) = RO(CENTER,CHAIN)
     $* ( 1.+Q(CENTER) * L22(II(2),II(3),II(4),II(5),II(6),II(7),II(8)))
           ELSE IF ( II(1).EQ.3 ) THEN
      W(CENTER,CHAIN) = RO(CENTER,CHAIN)
     $* ( 1.+Q(CENTER) * L23(II(2),II(3),II(4),II(5),II(6),II(7),II(8)))
           ENDIF
          ELSE IF ( CENTER.EQ.3 ) THEN
           IF ( II(1).EQ.0 ) THEN
      W(CENTER,CHAIN) = RO(CENTER,CHAIN)
     $* ( 1.+Q(CENTER) * L30(II(2),II(3),II(4),II(5),II(6),II(7),II(8)))
           ELSE IF ( II(1).EQ.1 ) THEN
      W(CENTER,CHAIN) = RO(CENTER,CHAIN)
     $* ( 1.+Q(CENTER) * L31(II(2),II(3),II(4),II(5),II(6),II(7),II(8)))
           ELSE IF ( II(1).EQ.2 ) THEN
      W(CENTER,CHAIN) = RO(CENTER,CHAIN)
     $* ( 1.+Q(CENTER) * L32(II(2),II(3),II(4),II(5),II(6),II(7),II(8)))
           ELSE IF ( II(1).EQ.3 ) THEN
      W(CENTER,CHAIN) = RO(CENTER,CHAIN)
     $* ( 1.+Q(CENTER) * L33(II(2),II(3),II(4),II(5),II(6),II(7),II(8)))
           ENDIF
          ENDIF
          SUMW = SUMW + W(CENTER,CHAIN)
400       CONTINUE
        LAMBDA(CHAIN) = I(CHAIN) / SUMW
      ENDIF
300     CONTINUE

      DO  500  CENTER = 1,M
        SUM = 0.
        DO  600  CHAIN = 1,K
          LL(CENTER,CHAIN) = LAMBDA(CHAIN)*W(CENTER,CHAIN)
          SUM = SUM + LAMBDA(CHAIN)*W(CENTER,CHAIN)
600       CONTINUE
        IF ( CENTER.EQ.1 ) THEN
```

```
               IF ( I(1).EQ.0 ) THEN
              L10(I(2),I(3),I(4),I(5),I(6),I(7),I(8)) = SUM
               ELSE IF ( I(1).EQ.1 ) THEN
              L11(I(2),I(3),I(4),I(5),I(6),I(7),I(8)) = SUM
               ELSE IF ( I(1).EQ.2 ) THEN
              L12(I(2),I(3),I(4),I(5),I(6),I(7),I(8)) = SUM
               ELSE IF ( I(1).EQ.3 ) THEN
              L13(I(2),I(3),I(4),I(5),I(6),I(7),I(8)) = SUM
               ENDIF
            ELSE IF ( CENTER.EQ.2 ) THEN
               IF ( I(1).EQ.0 ) THEN
              L20(I(2),I(3),I(4),I(5),I(6),I(7),I(8)) = SUM
               ELSE IF ( I(1).EQ.1 ) THEN
              L21(I(2),I(3),I(4),I(5),I(6),I(7),I(8)) = SUM
               ELSE IF ( I(1).EQ.2 ) THEN
              L22(I(2),I(3),I(4),I(5),I(6),I(7),I(8)) = SUM
               ELSE IF ( I(1).EQ.3 ) THEN
              L23(I(2),I(3),I(4),I(5),I(6),I(7),I(8)) = SUM
               ENDIF
            ELSE IF ( CENTER.EQ.3 ) THEN
               IF ( I(1).EQ.0 ) THEN
              L30(I(2),I(3),I(4),I(5),I(6),I(7),I(8)) = SUM
               ELSE IF ( I(1).EQ.1 ) THEN
              L31(I(2),I(3),I(4),I(5),I(6),I(7),I(8)) = SUM
               ELSE IF ( I(1).EQ.2 ) THEN
              L32(I(2),I(3),I(4),I(5),I(6),I(7),I(8)) = SUM
               ELSE IF ( I(1).EQ.3 ) THEN
              L33(I(2),I(3),I(4),I(5),I(6),I(7),I(8)) = SUM
               ENDIF
            ENDIF
 500       CONTINUE

 200    CONTINUE
 100 CONTINUE
  90 CONTINUE
  80 CONTINUE
  70 CONTINUE
  60 CONTINUE
  50 CONTINUE
  40 CONTINUE
C  ****  END OF MAIN LOOP  ****
   DO  150  CENTER = 1,M
      WRITE (*,2) ( LL(CENTER,CHAIN), CHAIN = 1,K )! MEAN QUEUE LENGTHS
 150 CONTINUE
   WRITE (*,2) ( LAMBDA(CHAIN) , CHAIN = 1,K ) ! CHAIN THROUGHPUTS
  2 FORMAT (2X,8F8.5)
   CALL CPUTIME(T2)
   WRITE (*,*) 'CPU TIME = ',(T2-T1)/1000000.
   STOP
   END

   SUBROUTINE XX(A,R,B)
   INTEGER A(8),B(8),R
   DO  10  J = 1,8
      IF (J  .EQ. R) THEN
        B(R) = A(R) - 1
      ELSE
        B(J) = A(J)
      ENDIF
 10  CONTINUE
```

```
  RETURN
  END

  SUBROUTINE DATA (K,M,N,Q,S,MAXCEN)
  INTEGER N(8),Q(MAXCEN),K,M,MAXCEN
  REAL S(MAXCEN,8)
  OPEN(1,FILE='/NET2 DATA A')
  READ(1,*) K                ! NO OF CHAINS
  READ(1,*) M                ! NO OF CENTERS
  READ(1,*) ( Q(I),I=1,M )      ! CENTER TYPES
  READ(1,*) ( N(I),I=1,K )      ! CHAIN POPULATIONS
  DO 10 I=1,M
    READ(1,*) ( S(I,J) , J=1,K ) ! MEAN SERVICE DEMANDS =
10 CONTINUE       ! VISIT RATIO * MEAN SERVICE TIME PER VISIT
  RETURN
  END
```

APPENDIX D

FORTRAN CODE FOR MVAC-1

```
      PROGRAM MVAC_ALGORITHM
C      REGULAR IMPLEMENTATION  ( MVAC-1 )
C      FILENAME : MVAC83.FOR   ( 3 CENTERS )
C
      INTEGER VV(3),N(8),QQ(3),B(3,3)
      INTEGER CHAIN,CENTER,S,D
      REAL L(0:10,3,0:10,0:10,0:10),LL(3,10,0:10,0:10,0:10),A(3,10),
     $    Q(3,5),LAMBDA1(5),LAMBDA(10,0:10,0:10,0:10),
     $    AA(3,5),SUMA(10)
      CALL DATA (D,CENTER,N,QQ,AA)
      DO 15 I=1,CENTER
        B(I,I) = 1
 15   CONTINUE
      S = D
      J1 = 3
C  **  GENERATION OF MEAN SERVICE TIMES FOR THE ORIGINAL NETWORK **
      LK = 0
      DO  13 J=1,D
      DO  23 K=1,N(J)-1
         LK = LK + 1
         DO 33 I = 1,CENTER
           A(I,LK) = AA(I,J)
 33        CONTINUE
 23   CONTINUE
 13   CONTINUE
      DO 43 J=LK+1,LK+D
      DO 53 I=1,CENTER
        A(I,J) = AA(I,J-LK)
 53   CONTINUE
 43   CONTINUE

      ISUM = 0
      DO  2  K = 1,D
         ISUM = ISUM + N(K)
  2   CONTINUE
      CHAIN = ISUM       !
      DO  6  K = 1,CHAIN
         SUMA(K) = 0.0
         DO  7  J = 1,CENTER
            SUMA(K) = SUMA(K) + A(J,K)
 7        CONTINUE
 6    CONTINUE

C     ** BEGINNING OF THE BASIC STEP  **
      K0 = 1
```

```
    DO 10  K = K0,CHAIN

C   ******* BASIC STEP *******

      DO 20  KI = CHAIN,K,-1
        TOTAL = CHAIN - KI
        DO  30  V1 = 0,CHAIN
           VV(1) = V1
        DO  30  V2 = 0,CHAIN
           VV(2) = V2
        DO  30  V3 = 0,CHAIN
           VV(3) = V3
        IF ( V1+V2+V3 .EQ.TOTAL ) THEN
          SUM = 0.0
          DO  40  J = 1,J1
             SUM = SUM + A(J,K)*( L(K-1,J,V1,V2,V3)+VV(J) )
 40        CONTINUE
          LAMBDA(K,V1,V2,V3) = 1. / ( SUMA(K) + SUM )
          DO 50  J = 1,CENTER
            LL(J,K,V1,V2,V3) = LAMBDA(K,V1,V2,V3) * A(J,K)
  $        * ( 1. + QQ(J) * ( L(K-1,J,V1,V2,V3)+VV(J)))
 50        CONTINUE

          DO 60 I = 1,CENTER
            SUM = 0.0
            DO 70  J = 1,CENTER
              SUM = SUM + LL(J,K,V1,V2,V3) *
  $            L( K-1 , I , V1+B(J,1) , V2+B(J,2) , V3+B(J,3) )
 70          CONTINUE
            L(K,I,V1,V2,V3) = SUM + LL(I,K,V1,V2,V3)
 60        CONTINUE
        ENDIF
 30      CONTINUE
 20    CONTINUE
 10  CONTINUE

    DO  9  J = 1,CENTER
      Q(J,D) = LL(J,CHAIN,0,0,0) * N(D)
  9 CONTINUE
    LAMBDA1(D) = LAMBDA(CHAIN,0,0,0) * N(D)

C   *******  END  OF  BASIC  STEP   ******

    DO 120  L1 = 1 , S-1
      CALL ICHANGE ( A,CENTER,CHAIN,CHAIN-L1 )
      DO 206  K = 1,CHAIN
        SUMA(K) = 0.0
        DO 207  J = 1,CENTER
          SUMA(K) = SUMA(K) + A(J,K)
 207     CONTINUE
 206   CONTINUE

      K0 = CHAIN - L1

      DO  205  K = K0 , CHAIN
      DO  210  KI = CHAIN,K,-1
         TOTAL = CHAIN - KI
      DO  250  V1 = 0,CHAIN
         VV(1) = V1
      DO  250  V2 = 0,CHAIN
```

```
         VV(2) = V2
       DO  250  V3 = 0,CHAIN
         VV(3) = V3
        IF ( V1+V2+V3 .EQ. TOTAL ) THEN
        SUM = 0.0
        DO 260  J = 1,J1
          SUM = SUM + A(J,K)*( L(K-1,J,V1,V2,V3) + VV(J) )
260       CONTINUE
        LAMBDA(K,V1,V2,V3) = 1. / ( SUMA(K) + SUM )
        DO  270  J = 1,CENTER
        LL(J,K,V1,V2,V3) = LAMBDA(K,V1,V2,V3) * A(J,K)
  $               * ( 1. + QQ(J)*( L(K-1,J,V1,V2,V3)+VV(J) ))
270       CONTINUE
        DO 280  I = 1,CENTER
          SUM = 0.0
          DO 200  J = 1,CENTER
            SUM = SUM + LL(J,K,V1,V2,V3) * L(K-1,I,V1+B(J,1),
  $           V2+B(J,2) , V3+B(J,3))
200         CONTINUE
          L(K,I,V1,V2,V3) = SUM + LL(I,K,V1,V2,V3)
280       CONTINUE
        ENDIF
250     CONTINUE
210     CONTINUE
205     CONTINUE

      LAMBDA1(D-L1) = LAMBDA(CHAIN,0,0,0) * N(D-L1)
      DO  208  J  = 1,CENTER
        Q(J,D-L1) = LL(J,CHAIN,0,0,0) * N(D-L1)
208     CONTINUE

120  CONTINUE
   DO  117  I = 1,CENTER
      WRITE(*,3) ( Q(I,J) , J = 1,D ) ! MEAN QUEUE LENGTHS
117  CONTINUE
   WRITE(*,3) ( LAMBDA1(J) , J = 1,D ) ! CHAIN THROUGHPUTS
  3 FORMAT (2X,7F10.6)

   STOP
   END

   SUBROUTINE ICHANGE (B,CENTRE,CHAIN,L)
   INTEGER CENTRE,CHAIN,L
   REAL B(CENTRE,CHAIN),TEMP
   DO 10  I = 1,CENTRE
     TEMP = B(I,L)
     B(I,L) = B(I,CHAIN)
     B(I,CHAIN) = TEMP
10  CONTINUE
   RETURN
   END

   SUBROUTINE DATA (K,M,N,QQ,A)
   INTEGER N(6),QQ(3),K,M
   REAL A(3,6)
   OPEN (1,FILE='NET2.DAT')
   READ (1,*) K               ! NO. OF CHAINS
   READ (1,*) M                ! NO. OF CENTERS
   READ (1,*) (QQ(I) , I=1,M)  ! CENTER TYPES =0:IS , =1:SSFR
   READ (1,*) (N(I) , I=1,K)   ! CHAIN POPULATIONS
   DO 10 I=1,M
     READ (1,*) ( A(I,J) , J=1,K) ! MEAN SERVICE DEMANDS =
10  CONTINUE          ! VISIT RATIO * MEAN SERVICE TIME PER VISIT
   RETURN
   END
```

APPENDIX E

FORTRAN CODE FOR MVAC-2

```
      PROGRAM MVAC_ALGORITHM
C     MVAC-2
C     FILENAME : MVAC83.FOR  ( 3 CENTER )
C     ******************************************************************
C         SUMA(I)       I=1,CHAIN
C         START(I)      I=1,CHAIN
C         VV(I)         I=1,CENTER
C         DV(I)         I=1,CENTER
C         T(I)          I=1,CENTER
C         A(I,J)        I=1,CENTER  J=1,CHAIN
C         V(I,J)        I=1,COUNT   J=1,CENTER
C         LAMBDA(I,J)   I=1,CHAIN   J=1,COUNT
C         L(I,J,K)      I=0,CHAIN   J=1,CENTER  K=1,COUNT
C         LL(I,J,K)     I=1,CENTER  J=1,CHAIN   K=1,COUNT
C
C          WHERE  COUNT = COMBINATION (CHAIN+CENTER-1,CENTER)
C
C     ******************************************************************
C
      INTEGER V( 680,3),START(15),VV(3),DV(3),T(3),N(5),QQ(3)
      INTEGER STARTP,ENDP,COL,P1,P2,COUNT,FOUND,CHAIN,CENTER,Z,S,D
      REAL L(0:15,3, 680),LL(3,15, 680),A(3,15),SUMA(15),
     $    Q(3,5),LAMBDA1(5),LAMBDA(15, 680),AA(3,5)
      LOGICAL SAME
      CALL DATA (D,CENTER,N,QQ,AA)
      S = D
      J1 = 3
C  ** GENERATION OF SERVICE TIMES FOR THE ORIGINAL NETWORK  **
      LK = 0
      DO  21 J=1,D
      DO  31 K=1,N(J)-1
        LK = LK + 1
        DO 41 I = 1,CENTER
          A(I,LK) = AA(I,J)
41      CONTINUE
31    CONTINUE
21    CONTINUE
      DO  17 J=LK+1,LK+D
      DO  18 I=1,CENTER
        A(I,J) = AA(I,J-LK)
18    CONTINUE
17    CONTINUE

      ISUM = 0
      DO  2  K = 1,D
```

```
       ISUM = ISUM + N(K)
   2 CONTINUE
     CHAIN = ISUM
     DO  6  K = 1,CHAIN
        SUMA(K) = 0.0
        DO  7  J = 1,CENTER
           SUMA(K) = SUMA(K) + A(J,K)
 7      CONTINUE
 6   CONTINUE

C  *****  INITIALIZATION  ******

     DO 10 I = 1,CENTER
       V(1,I) = 0
 10  CONTINUE
     DO 20 I = 2,CENTER+1
       V(I,I-1) = 1
 20  CONTINUE
     STARTP = 2
     ENDP   = CENTER+1
     COUNT  = CENTER+1

C  *****   MAIN  LOOP   *******
     DO 22 NN = 0,CHAIN-3
       DO 30 I = STARTP,ENDP
         DO 40 J = CENTER,1,-1
           IF ( V(I,J).NE.0 ) THEN
             COL = J
             GO TO 45
           ENDIF
 40        CONTINUE
 45        CONTINUE
         DO 51 K = COL,CENTER
           DO 61 L1 = 1,CENTER
             DV(L1) = V(I,L1)
 61          CONTINUE
           DV(K) = V(I,K) + 1
           COUNT = COUNT + 1
           DO 71 L1 = 1,CENTER
             V(COUNT,L1) = DV(L1)
 71          CONTINUE
 51        CONTINUE
 30      CONTINUE
       STARTP = ENDP + 1
       ENDP   = COUNT

 22  CONTINUE
C  ****  REVERSING THE ORDER  ****
     DO 81 I = 1,COUNT/2
       DO 97 J = 1,CENTER
         T(J)   = V(I,J)
         V(I,J) = V(COUNT-I+1,J)
         V(COUNT-I+1,J) = T(J)
 97      CONTINUE
 81  CONTINUE
C  ****  INITIALIZATION OF THE ARRAY 'START'  ****
     START(1) = 1
     DO 92 I = 2,CHAIN
       P1 = 1
       DO 93 J = CHAIN-I+2 , CHAIN+CENTER-I
```

```
        P1 = P1 * J
  93    CONTINUE
      P2 = 1
      DO 94 K = 1,CENTER-1
        P2 = P2 * K
  94    CONTINUE
      START(I) = P1/P2 + START(I-1)
  92  CONTINUE
     START(CHAIN+1) = COUNT + 1

     K0 = 1
     DO 5  K = K0,CHAIN

C    ******* BASIC  STEP  **********

      DO 50  Z = START(K),COUNT
        IC = IC + 1
        SUM=0.0
        DO 60  J = 1,J1
          SUM = SUM + A(J,K)*( L(K-1,J,Z)+V(Z,J) )
  60    CONTINUE
        LAMBDA(K,Z) = 1. / ( SUMA(K) + SUM )
        DO 70  J = 1,CENTER
           LL(J,K,Z) = LAMBDA(K,Z) * A(J,K)
   $                 * ( 1. + QQ(J) * ( L(K-1,J,Z)+V(Z,J)))
  70    CONTINUE

        DO 80 I = 1,CENTER
          SUM = 0.0
          DO 100  J = 1,CENTER
           DO 110  JJ = 1,CENTER
             VV(JJ) = V(Z,JJ)
 110         CONTINUE
          VV(J) = VV(J) + 1
          FOUND = 0
          DO 90  I1 = START(K-1),COUNT
            SAME = .TRUE.
            DO 96  I2 = 1,CENTER
              SAME = VV(I2).EQ.V(I1,I2)
              IF (.NOT.SAME)   GOTO 90
  96          CONTINUE
            FOUND = I1
  89        CONTINUE
  90        CONTINUE
          SUM = SUM + LL(J,K,Z)*L(K-1,I,FOUND)
 100        CONTINUE
          L(K,I,Z) = SUM + LL(I,K,Z)
  80      CONTINUE
  50    CONTINUE

   5 CONTINUE

    DO  9  J = 1,CENTER
      Q(J,D) = LL(J,CHAIN,COUNT) * N(D)
   9 CONTINUE
    LAMBDA1(D) = LAMBDA(CHAIN,COUNT) * N(D)

C   *******  END  OF  BASIC  STEP   ******

    DO 120  L1 = 1 , S-1
```

```
      CALL ICHANGE ( A,CENTER,CHAIN,CHAIN-L1 )
      DO 206  K = 1,CHAIN
        SUMA(K) = 0.0
        DO 207  J = 1,CENTER
          SUMA(K) = SUMA(K) + A(J,K)
 207      CONTINUE
 206    CONTINUE

      K0 = CHAIN - L1
      DO  205  K = K0 , CHAIN
      DO  250  Z = START(K) , COUNT
        SUM = 0.0
        DO 260  J = 1,J1
          SUM = SUM + A(J,K)*( L(K-1,J,Z)+V(Z,J) )
260       CONTINUE
        LAMBDA(K,Z) = 1. / ( SUMA(K) + SUM )
        DO  270  J = 1,CENTER
        LL(J,K,Z) = LAMBDA(K,Z) * A(J,K)
  $               * ( 1. + QQ(J)*( L(K-1,J,Z)+V(Z,J) ))
270       CONTINUE
        DO 280  I = 1,CENTER
          SUM = 0.0
          DO 200  J = 1,CENTER
            DO 210  JJ = 1,CENTER
              VV(JJ) = V(Z,JJ)
210           CONTINUE
            VV(J) = VV(J) + 1
            DO 290 I1 = START(K-1),COUNT
              SAME = .TRUE.
              DO 296  I2 = 1,CENTER
                SAME = VV(I2).EQ.V(I1,I2)
                IF (.NOT.SAME )  GOTO 290
296             CONTINUE
              FOUND = I1
289           CONTINUE
290           CONTINUE
            SUM = SUM + LL(J,K,Z) * L(K-1,I,FOUND)
200         CONTINUE
          L(K,I,Z) = SUM + LL(I,K,Z)

280       CONTINUE
250     CONTINUE
205     CONTINUE

      LAMBDA1(D-L1) = LAMBDA(CHAIN,COUNT) * N(D-L1) ! CHAIN THROUGHPUTS
      DO  208  J  = 1,CENTER
        Q(J,D-L1) = LL(J,CHAIN,COUNT) * N(D-L1)! MEAN QUEUE LENGTHS
208     CONTINUE

120  CONTINUE
    DO  117  I = 1,CENTER
      WRITE(*,3) ( Q(I,J) , J = 1,D ) ! MEAN QUEUE LENGTHS
117  CONTINUE
    WRITE(*,3) ( LAMBDA1(J) , J = 1,D ) ! CHAIN THROUGHPUTS
  3 FORMAT (2X,7F10.6)

    STOP
    END

    SUBROUTINE ICHANGE (B,CENTRE,CHAIN,L)
  INTEGER CENTRE,CHAIN,L
  REAL B(CENTRE,CHAIN),TEMP
  DO 10  I = 1,CENTRE
    TEMP = B(I,L)
    B(I,L) = B(I,CHAIN)
    B(I,CHAIN) = TEMP
10  CONTINUE
  RETURN
  END

  SUBROUTINE DATA (K,M,N,QQ,A)
  INTEGER N(5),QQ(3),K,M
  REAL A(3,6)
  OPEN (1,FILE='NET2.DAT')
  READ(1,*) K                  ! NO. OF CHAINS
  READ(1,*) M                  ! NO. OF CENTERS
  READ(1,*) ( QQ(I),I=1,M)     ! CENTER TYPES  =0 if IS, =1 if SSFR
  READ(1,*) ( N(I),I=1,K)      ! CHAIN POPULATIONS
  DO 10 I=1,M
    READ (1,*) ( A(I,J) , J=1,K )  ! MEAN SERVICE DEMANDS =
10  CONTINUE    !  VISIT RATIO * MEAN SERVICE TIME PER VISIT
  RETURN
  END
```

APPENDIX F

FORTRAN CODE FOR MVAC-3

```
      PROGRAM MVAC_ALGORITHM
C     IMPLEMENTATION NO.3  ( MVAC-3 )
C     FILENAME : DENEC.FOR  ( FOR 3 CENTERS )
C     ****************************************************************
C         SUMA(I)       I=1,CHAIN
C         A(I,J)        I=1,CENTER  J=1,CHAIN
C         V(I,J)        I=1,COUNT   J=1,CENTER
C         LAMBDA(I,J)   I=1,CHAIN   J=1,COUNT
C         L(I,J,K)      I=0,CHAIN   J=1,CENTER  K=1,COUNT
C         LL(I,J,K)     I=1,CENTER  J=1,CHAIN   K=1,COUNT
C
C          WHERE  COUNT = COMBINATION (CHAIN+CENTER-1,CENTER)
C
C     ****************************************************************
C
      INTEGER V( 680,3),START(21),DV(3),T(3),N(5),QQ(3)
      INTEGER STARTP,ENDP,COL,P1,P2,COUNT,CHAIN,CENTER,Z,S,D,COL2
      INTEGER ZZ,INDEX( 680,3)
      REAL L(0:15,3, 680),LL(3,15, 680),A(3,15),SUMA(15),
     +     Q(3,5),LAMBDA1(5),LAMBDA(15, 680),AA(3,5),T1,T2
C
C      CALL CPUTIME(T1)
      CALL DATA (D,CENTER,N,QQ,AA)
      S = D
      J1 = 3
C  **  GENERATION OF SERVICE TIMES FOR THE ORIGINAL NETWORK  **
      LK = 0
      DO 32 J=1,D
      DO 42 K=1,N(J)-1
        LK = LK + 1
        DO 52 I=1,3
          A(I,LK) = AA(I,J)
52      CONTINUE
42    CONTINUE
32    CONTINUE
      DO  17 J=LK+1,LK+D
      DO  18 I=1,3
        A(I,J)=AA(I,J-LK)
18    CONTINUE
17    CONTINUE

      ISUM = 0
      DO  2  K = 1,D
        ISUM = ISUM + N(K)
  2 CONTINUE
```

```
      CHAIN = ISUM
      DO 6  K = 1,CHAIN
         SUMA(K) = 0.0
         DO 7  J = 1,CENTER
            SUMA(K) = SUMA(K) + A(J,K)
 7       CONTINUE
 6    CONTINUE

C  ****** INITIALIZATION  ******
      DO 10 I = 1,CENTER
        V(1,I) = 0
 10   CONTINUE
      DO 20 I = 2,CENTER+1
        V(I,I-1) = 1
 20   CONTINUE
      STARTP = 2
      ENDP   = CENTER+1
      COUNT  = CENTER+1
C **** INITIALIZATION OF THE INDEX MATRIX  ****
      DO 21  J = 1,CENTER
        INDEX(1,J) = J+1
 21   CONTINUE

C  *****  MAIN  LOOP  ******
      DO 22 NN = 0,CHAIN-3
        COL2 = 0
        DO 30 I = STARTP,ENDP
          DO 40 J = CENTER,1,-1
            IF ( V(I,J).NE.0 ) THEN
              COL = J
              GO TO 45
            ENDIF
 40       CONTINUE
 45       CONTINUE
          DO 51 K = COL,CENTER
            DO 61 L1 = 1,CENTER
              DV(L1) = V(I,L1)
 61         CONTINUE
            DV(K) = V(I,K) + 1
            COUNT = COUNT + 1
            INDEX(I,K) = COUNT
            DO 71 L1 = 1,CENTER
              V(COUNT,L1) = DV(L1)
 71         CONTINUE
 51       CONTINUE
          IF ( COL.EQ.2 ) THEN
            COL2 = COL2 + 1
            INDEX(I,1) = COUNT - COL2 - 2
          ELSE IF ( COL.EQ.3 ) THEN
            INDEX(I,1) = COUNT - (COL2+2)
            INDEX(I,2) = COUNT - 1
          ENDIF

 30     CONTINUE
        STARTP = ENDP + 1
        ENDP   = COUNT
 22   CONTINUE
C **** REVERSING THE ORDER ****
      DO 81 I = 1,COUNT/2
        DO 97 J = 1,CENTER
```

```
          T(J)  = V(I,J)
          V(I,J) = V(COUNT-I+1,J)
          V(COUNT-I+1,J) = T(J)
  97    CONTINUE
  81  CONTINUE
C   ****  INITIALIZATION OF THE ARRAY 'START'  ****
      START(1) = 1
      DO 92 I = 2,CHAIN
        P1 = 1
        DO 93 J = CHAIN-I+2 , CHAIN+CENTER-I
          P1 = P1 * J
  93    CONTINUE
        P2 = 1
        DO 94 K = 1,CENTER-1
          P2 = P2 * K
  94    CONTINUE
        START(I) = P1/P2 + START(I-1)
  92  CONTINUE
      START(CHAIN+1) = COUNT + 1
      K0 = 1
      DO 5  K = K0,CHAIN

C    *****  BASIC  STEP  *****

        DO 50  Z = START(K),COUNT
          IC = IC + 1
          SUM=0.0
          DO 60  J = 1,J1
            SUM = SUM + A(J,K)*( L(K-1,J,Z)+V(Z,J) )
 60       CONTINUE
          LAMBDA(K,Z) = 1. / ( SUMA(K) + SUM )
          DO 70  J = 1,CENTER
             LL(J,K,Z) = LAMBDA(K,Z) * A(J,K)
   +                   * ( 1. + QQ(J) * ( L(K-1,J,Z)+V(Z,J)))
 70       CONTINUE
          DO 80 I = 1,CENTER
            SUM = 0.0
            DO 100  J = 1,CENTER
              ZZ = COUNT-INDEX(COUNT-Z+1,J)+1
              SUM = SUM + LL(J,K,Z)*L(K-1,I,ZZ)
100         CONTINUE
            L(K,I,Z) = SUM + LL(I,K,Z)
 80       CONTINUE
 50     CONTINUE

  5 CONTINUE

    DO  9  J = 1,CENTER
      Q(J,D) = LL(J,CHAIN,COUNT) * N(D)
  9 CONTINUE
    LAMBDA1(D) = LAMBDA(CHAIN,COUNT) * N(D)

C   *******  END  OF  BASIC  STEP   ******

    DO 120  L1 = 1 , S-1
      CALL ICHANGE ( A,CENTER,CHAIN,CHAIN-L1 )
      DO 206  K = 1,CHAIN
        SUMA(K) = 0.0
        DO 207  J = 1,CENTER
          SUMA(K) = SUMA(K) + A(J,K)
 207     CONTINUE
 206   CONTINUE

      K0 = CHAIN - L1
      DO  205  K = K0 , CHAIN
      DO  250  Z = START(K) , COUNT
        SUM = 0.0
        DO 260  J = 1,J1
          SUM = SUM + A(J,K)*( L(K-1,J,Z)+V(Z,J) )
 260     CONTINUE
        LAMBDA(K,Z) = 1. / ( SUMA(K) + SUM )
        DO  270  J = 1,CENTER
        LL(J,K,Z) = LAMBDA(K,Z) * A(J,K)
   +              * ( 1. + QQ(J)*( L(K-1,J,Z)+V(Z,J) ))
 270     CONTINUE
        DO 280  I = 1,CENTER
          SUM = 0.0
          DO 200  J = 1,CENTER
            ZZ = COUNT-INDEX(COUNT-Z+1,J)+1
            SUM = SUM + LL(J,K,Z) * L(K-1,I,ZZ)
 200       CONTINUE
          L(K,I,Z) = SUM + LL(I,K,Z)
 280     CONTINUE
 250   CONTINUE
 205   CONTINUE

      LAMBDA1(D-L1) = LAMBDA(CHAIN,COUNT) * N(D-L1)
      DO  208  J  = 1,CENTER
        Q(J,D-L1) = LL(J,CHAIN,COUNT) * N(D-L1)
 208   CONTINUE

 120  CONTINUE
    DO  117  I = 1,CENTER
      WRITE(*,3) ( Q(I,J) , J = 1,D )  ! MEAN QUEUE LENGTHS
 117  CONTINUE
    WRITE(*,3) ( LAMBDA1(J) , J = 1,D )  ! CHAIN THROUGHPUTS
  3 FORMAT (2X,6F10.6)
C    CALL CPUTIME(T2)
C    WRITE(*,*) 'TIME ELAPSED = ',(T2-T1)/1000000.
    STOP
    END

    SUBROUTINE ICHANGE (B,CENTRE,CHAIN,L)
    INTEGER CENTRE,CHAIN,L
    REAL B(CENTRE,CHAIN),TEMP
    DO 10  I = 1,CENTRE
      TEMP = B(I,L)
      B(I,L) = B(I,CHAIN)
      B(I,CHAIN) = TEMP
 10  CONTINUE
    RETURN
    END

    SUBROUTINE DATA (K,M,N,QQ,A)
    INTEGER N(8),QQ(3),K,M
    REAL A(3,8)
    OPEN (1,FILE='NET2.DAT')
    READ (1,*) K                  ! NO. OF CHAINS
    READ (1,*) M                  ! NO. OF CENTERS
    READ (1,*) ( QQ(I) , I=1,M )  ! CENTER TYPE  =0 if IS, =1 if SSFR
   READ (1,*) ( N(I)  , I=1,K )  ! CHAIN POPULATIONS
   DO  10 I = 1,M
      READ (1,*) ( A(I,J) , J=1,K ) ! MEAN SERVICE DEMANDS =
10  CONTINUE          !  VISIT RATIO * MEAN SERVICE TIME PER VISIT
   RETURN
   END
```

APPENDIX G

FORTRAN CODE FOR THE LINEARIZER ALGORITHM

```
      PROGRAM LINEARIZER_ALGORITHM
C     FILENAME : LINS.FOR
      PARAMETER ( MAXM=20 , MAXK=10 , MAXN=25 )
      REAL D,L,S,LL,L2,FF,LLC,F,LC1,W,Y
      DIMENSION
D(MAXM,MAXK,MAXK),LL(MAXM,MAXK,MAXK),FF(MAXM,MAXK,MAXK),
     $ L2(MAXM,MAXK), LLC(MAXM,MAXK), F(MAXM,MAXK), LC1(MAXM,MAXK),
     $ W(MAXM,MAXK), Y(MAXM,MAXK), L(MAXM,MAXK), S(MAXM,MAXK)
      INTEGER N(MAXK), Q(MAXM)
      CALL DATA ( K,M,N,Q,S,MAXM,MAXK )
      CALL LINEAR ( Q,N,M,K,D,L,S,LL,L2,FF,LLC,F,LC1,W,Y,MAXM,MAXK )
      STOP
      END

      SUBROUTINE LINEAR (Q,N,M,K,D,L,S,LL,L2,FF,LLC,F,LC1,W,Y,MAXM,MAXK)
      REAL L,S,LL,L2,FF,LLC,F,LC1,W,Y
      DIMENSION
D(MAXM,MAXK,MAXK),LL(MAXM,MAXK,MAXK),FF(MAXM,MAXK,MAXK)
     $ ,LLC(MAXM,MAXK),F(MAXM,MAXK),LC1(MAXM,MAXK),W(MAXM,MAXK)
     $ ,L(MAXM,MAXK),S(MAXM,MAXK),L2(MAXM,MAXK),Y(MAXM,MAXK)
      INTEGER N(MAXK),Q(MAXM),POP(20)

C *********   STEP 1  ( INITIALIZATION )  ************
C
      DO  10  KK = 1,K
      DO  10  MM = 1,M
        L(MM,KK) = N(KK) / REAL(M)
 10   CONTINUE
      DO  100  MM = 1,M
      DO  100  KK = 1,K
      DO  100  JJ = 1,K
        LL (MM,KK,JJ) = ALFA (N,K,KK,JJ) / M
 100  CONTINUE
      I = 1

C **************  STEP  2  **************

  3   CONTINUE
C     CORE FOR FULL NETWORK ENTERENCE
      CALL CORE (M,K,D,L,LC1,N,S,Q,W,MAXM,MAXK)
C
C **************  STEP  3  ***************

      IF ( I.EQ.3 )  GOTO 600
```

```
C ************** STEP 4 ***************

    DO 200 JJ = 1,K
      DO 210 KK = 1,K
         POP (KK) = N(KK)
 210   CONTINUE
      POP (JJ) = N(JJ) - 1
      DO 220 MM = 1,M
      DO 220 KK = 1,K
         L2 (MM,KK) = LL (MM,KK,JJ)
 220   CONTINUE
C    CORE FOR REDUCED NETWORK
      CALL CORE ( M,K,D,L2,LLC,POP,S,Q,W,MAXM,MAXK)
C
      DO 230 MM = 1,M
      DO 230 KK = 1,K
         IF ( POP (KK).EQ.0 ) THEN
            FF (MM,KK,JJ) = 0.0
         ELSE
            FF (MM,KK,JJ) = LLC (MM,KK) / POP (KK)
         ENDIF
         LL (MM,KK,JJ) = LLC (MM,KK)
 230   CONTINUE

 200 CONTINUE

C *************** STEP 5 ***************

    DO 300 MM = 1,M
    DO 300 KK = 1,K
       F (MM,KK) = LC1 (MM,KK) / N(KK)
 300 CONTINUE

    DO 400 MM = 1,M
    DO 400 KK = 1,K
    DO 400 JJ = 1,K
       D (MM,KK,JJ) = FF (MM,KK,JJ) - F (MM,KK)

 400 CONTINUE

    I = I + 1

    DO 500 MM = 1,M
    DO 500 KK = 1,K
         L (MM,KK) = LC1 (MM,KK)
 500 CONTINUE

    GOTO 3
 600 CONTINUE          ! LINEARIZER RESULTS
    DO 610 MM = 1,M
       WRITE (*,12) (LC1 (MM,KK) , KK = 1,K ) ! MEAN QUEUE LENGTHS
 12    FORMAT ( 3X,5F13.6 )
 610 CONTINUE
    DO 700 MM = 1,1
    DO 700 KK = 1,K
     IF ( W(MM,KK).EQ.0.0 ) THEN
       Y(MM,KK) = 0.0
     ELSE
       Y(MM,KK) = LC1 (MM,KK) / W(MM,KK) ! CHAIN THROUGHPUTS
     ENDIF
```

```
700  CONTINUE
    WRITE (*,12) (Y(1,KK), KK=1,K)
    DO  800  MM = 1,M
       WRITE (*,12) (W(MM,KK), KK=1,K)    ! MEAN WAITING TIMES
800  CONTINUE
    RETURN
    END

    SUBROUTINE CORE (M,K,D,LE,LA,N,S,Q,W,MAXM,MAXK )
    INTEGER M,K,Q
    REAL  D(MAXM,MAXK,MAXK),LL(30,20,20),  X(20)
    REAL  S(MAXM,MAXK),L(30,20),OLDL(30,20),LE(MAXM,MAXK)
    REAL  LA(MAXM,MAXK),W(MAXM,MAXK),F(30,20)
    REAL  SUMN,CUTOFF,MAX,DIFF
    DIMENSION N(MAXK), Q(MAXM)
    I = 1
    DO  5  MM = 1,M
    DO  5  KK = 1,K
         L (MM,KK) = LE (MM,KK)
 5  CONTINUE
 2  CONTINUE
    DO 10  MM = 1,M
    DO 10  KK = 1,K
       IF ( N(KK) .NE. 0 )  THEN
         F (MM,KK) = L(MM,KK) / N(KK)
       ELSE
         F (MM,KK) = 0.0
       ENDIF
 10  CONTINUE

    DO 20  MM = 1,M
    DO 20  KK = 1,K
    DO 20  JJ = 1,K
       LL(MM,KK,JJ) = ALFA(N,K,KK,JJ) * ( F(MM,KK) + D(MM,KK,JJ) )
 20 CONTINUE

    DO 30  MM = 1,M
    DO 30  JJ = 1,K
       SUML = 0.0
       DO 40  KK = 1,K
          SUML = SUML + LL(MM,KK,JJ)
 40     CONTINUE
       W (MM,JJ) = S(MM,JJ) * ( 1.0 + SUML * Q(MM) )
 30 CONTINUE

    DO 50  KK = 1,K
      SUMW = 0.0
      DO 55  MM = 1,M
        SUMW = SUMW + W(MM,KK)
 55   CONTINUE
      X(KK) = N(KK) / SUMW
 50 CONTINUE

    DO 60  MM = 1,M
    DO 60  KK = 1,K
      OLDL (MM,KK) = L (MM,KK)
      L(MM,KK) = X(KK) * W(MM,KK)
60  CONTINUE

    MAX = 0.0
```

```
   DO 70  MM = 1,M
   DO 70  KK = 1,K
     IF ( N(KK).EQ.0 )  GOTO  70
     DIFF = ABS( L(MM,KK)-OLDL(MM,KK) ) / N(KK)
     IF ( DIFF .GT. MAX )   MAX = DIFF
70  CONTINUE
   SUMN = 0.0
   DO 75  KK = 1,K
     SUMN = SUMN + N(KK)
75  CONTINUE
   CUTOFF = 1. / ( 4000. + 16.* SUMN )
   IF ( MAX.GT.CUTOFF ) GOTO 2

   DO 80 MM = 1,M
   DO 80 KK = 1,K
      LA(MM,KK) = L(MM,KK)
80  CONTINUE
   RETURN
   END

   FUNCTION ALFA (N,K,Z,J)
   INTEGER N,K,J,Z
   DIMENSION N(K)
   IF ( Z.EQ.J ) THEN
     ALFA = N(J) - 1.
   ELSE
     ALFA = N(Z)
   ENDIF
   IF ( ALFA .LT. 0. )   ALFA = 0.
   RETURN
   END

   SUBROUTINE DATA (K,M,N,QQ,A,MAXM,MAXK)
   INTEGER N(MAXK),QQ(MAXM),K,M
   REAL A(MAXM,MAXK)
   OPEN (1,FILE='NET2.DAT')
   READ (1,*) K                 ! NUMBER OF CHAINS
   READ (1,*) M                 ! NUMBER OF CENTERS
   READ (1,*) ( QQ(I) , I=1,M )  ! TYPE OF CENTER
   READ (1,*) ( N(I) , I=1,K)    ! CHAIN POPULATIONS
   DO  10  I=1,M
      READ (1,*) (A(I,J),J=1,K) ! MEAN SERVICE DEMANDS =
 10 CONTINUE         !  VISIT RATIO * MEAN SERVICE TIME PER VISIT
   RETURN
   END
```