NOTICE

The quality of this microform is heavily dependent upon the quality of the original thesis submitted for microfilming. Every effort has been made to ensure the highest quality of reproduction possible.

If pages are missing, contact the university which granted the degree.

Some pages may have indistinct print especially if the original pages were typed with a poor typewriter ribbon or if the university sent us an inferior photocopy.

Reproduction in full or in part of this microform is governed by the Canadian Copyright Act, R.S.C. 1970, c. C-30, and subsequent amendments.

AVIS

La qualité de cette microforme dépend grandement de la qualité de la thèse soumise au microfilmage. Nous avons tout fait pour assurer une qualité supérieure de reproduction.

S'il manque des pages, veuillez communiquer avec l'université qui a conféré le grade.

La qualité d'impression de certaines pages peut laisser à désirer, surtout si les pages originales ont été dactylographiées à l'aide d'un ruban usé ou si l'université nous a fait parvenir une photocopie de qualité inférieure.

La reproduction, même partielle, de cette microforme est soumise à la Loi canadienne sur le droit d'auteur, SRC 1970, c. C-30, et ses amendements subséquents.

Canada

Applying Decomposition and Aggregation Theory to the Analysis of Stochastic Petri Nets and Queueing Networks

Man Li

A Thesis
Submitted to the University of Ottawa
in Partial Fulfillment of
the Requirements for the Degree of
Ph. D in
Electrical Engineering

ISBN 0-315-85788-9

Canada

UNIVERSITÉ D'OTTAWA
UNIVERSITY OF OTTAWA

Abstract

Stochastic Petri Nets and queueing networks are important modelling tools for the performance analysis of computer systems and computer communication networks. Developing efficient analysis methods for them has always been a challenging topic. In this thesis, a class of Stochastic Petri Nets, called Local Balance Stochastic Petri Nets, and a class of queueing networks, called product form queueing networks are investigated. The parametric analysis of Stochastic Petri Nets in general is also studied. The major analysis tool used in this thesis is the Simon and Ando's Decomposition and Aggregation theory.

Local Balance Stochastic Petri Nets have the property that their equilibrium state probability distributions have product form solutions. However, the existing boundary for this class of Stochastic Petri Nets is limited. In this thesis, we extend the boundary of the Local Balance Stochastic Petri Nets, propose a systematic test procedure, as well as a C language program to identify this class of Stochastic Petri Nets, and prove that the Stochastic Petri Nets that have passed the test have product form solutions.

Simon and Ando's Decomposition and Aggregation theory is then applied to the analysis of Local Balance Stochastic Petri Nets. A Decomposition by Subnet method is proposed. The analysis of a Local Balance Stochastic Petri Net is decomposed into the analysis of subnets. The results are combined to obtain the analysis for the original system.

Through the Decomposition by Subnet method, a Norton's theorem for Local Balance Stochastic Petri Nets is developed. By decomposing according to a particular subnet, an aggregated net is constructed. This is that subnet with marking dependent firing rates. We show that the aggregated net may concisely and exactly represent the original Stochas-

tic Petri Net. One of the applications of the Norton's theorem is to facilitate parametric analysis of Local Balance Stochastic Petri Nets.

When a Stochastic Petri Net is not a Local Balance Stochastic Petri Net, the concept of "Ideal Aggregate" is used to develop efficient parametric analysis of it. By following a systematic way of constructing the infinitesimal matrix, the transition rate of interest is confined into a small diagonal submatrix. According to the algorithm, every time that particular rate takes a new value, only a Markov Chain of the order of the small diagonal submatrix needs to be analyzed. As a result, computational time requirements are greatly reduced.

For the product form queueing networks, we first improve the efficiency of the Distribution Analysis by Chain(DAC) algorithm. We show that the calculation of marginal queue length distribution and throughput in each recursion may be avoided. As a result, computational time requirements are reduced. In addition, the improved algorithm has the benefits with regard to reducing possible numerical errors.

Simon and Ando's Decomposition and Aggregation theory is used to develop an Independent Decomposition and Aggregation method for the analysis of product form queueing networks. The queueing network is first transformed into a network with nodes of Infinite Server types. That network is then decomposed independently chain by chain.

Through the Independent Decomposition and Aggregation method, an algorithm called Adaptive Convolution By Chain(ACCAL) is derived. It is efficient in dealing with networks that have many chains and a few nodes. Compared with other algorithms, ACCAL has a smaller number of operations. In addition, ACCAL first converts the network into an equivalent one that may be analyzed more efficiently. There are three independent parts in the algorithm that may be executed in parallel on a multiprocessor system to further improve the efficiency. The adaptive nature and the parallel processing characteristic distinguish ACCAL from other algorithms.

Acknowledgements

I would like to express my deepest gratitude to my thesis supervisor, Dr. N. D. Georganas, for his guidance and consistent encouragement throughout the course of this research.

I would like to thank the members of my advisory committee Dr. E. Petriu and Dr. C. M. Woodside for their suggestions. Also I would like to thank Dr. D. Petriu, Dr. O. Yang and Dr. P. Gaiko for the useful discussions with them.

The help from my colleagues is highly appreciated.

Finally, I would like to thank my family as well as Yiying for their understanding and support.

Contents

List of Figures

List of Tables

Summary of Notations

Stochastic Petri Nets

- p_i: place i.
- t_j: transition j.
- μ: a marking.
- m_i: the number of tokens in p_i.
- m_{0i}: the initial number of tokens in p_i.
- $m_j(\mu)$: the number of tokens in p_j in marking μ.
- $R(\mu_0)$: the Reachability Graph.
- D: incidence matrix.
- D^+: incidence matrix.
- D^-: incidence matrix.
- r_i: firing rate of transition t_i.
- $r_i(\mu)$ firing rate of transition t_i under marking μ.
- S_i: the *ith* cycle of transitions.
- C_i: subnet i.

- $\hat{\mu}_i^j$: Basic Subnet j marking.
- $\hat{\mu}_0^j$: initial Basic Subnet Marking for subnet j.
- μ_i^j: Separate Subnet j marking of μ_i.
- CSS: Communicating Subnet Set.
- nsp: the set of non-resource places.
- ρ: the mapping function.
- $\underline{\mu}$: marking μ with the number of tokens in the common places being set to zero.
- I_x^j: an integer representing the number of tokens in p_x that have been absorbed by subnet j.
- $fm(t_i)$: the set of Favorite Markings of t_i.
- $ps(\mu_i|t_k)$: Post-marking of μ_i with respect to t_k.
- $pr(\mu_i|t_k)$: Pre-marking of μ_i with respect to t_k.
- μ_{ik}: a marking in class k.
- $n(k)$: the number of markings in class k.
- $v^*(\mu_{ik})$: the conditional probability of μ_{ik}.

Queueing Networks

- N: number of nodes.
- R: total number of chains in the network.
- $B(N, \mathbf{K})$: a closed multiple-chain BCMP queueing network with N nodes and a population vector $\mathbf{K}$.

- $\mu_j(k)$: service rate of node j when there are k customers at node j.
- $m_{ic}^{(r)}$: mean service time for a customer of type r in class c an node i.
- T_r: throughput of chain r.
- Q_j: mean number of customers in node j.
- e_{jr}: visit ratio of chain r customer to node j.
- T_{jr}: $T_r e_{jr}$ throughput of chain r customers at node j.
- Q_{jr}: mean number of chain r customers at node j.
- t_{jr}: mean service time for chain r customer at node j.
- w_{jr}: $e_{jr} t_{jr}$
- k_{jr}: the number of chain r customers at node j.
- k_j: the number of customers at node j.
- K_r: total number of customers in chain r.
- $K = \sum_{r=1}^{R} K_r$: total network population.
- $\mathbf{k} = (k_1, ..., k_N)$: joint queue length.
- $\mathbf{k}_i = (k_{i1}, ..., k_{iR})$: state of node i.
- $\underline{\mathbf{k}} = (\mathbf{k}_1, ..., \mathbf{k}_N)$: state of the network.
- $\mathbf{k}(r) = (k_{1r}, ..., k_{Nr})$ state of the rth chain.
- $L_r = \{\mathbf{k}(r) : \sum_{i=1}^{N} k_{ir} = K_r\}$.
- $\mathbf{e}_j$: a compatible vector whose jth element is one and other elements are zero.
- $S^r = \{\mathbf{k} : \sum_{j=1}^{N} k_j = \sum_{l=1}^{r} K_l\}$.

- $\underline{S}^r = \{\underline{\mathbf{k}} : k_{jx} \geq 0,\ \sum_{j=1}^{N} k_{jx} = K_x,\ j = 1,\ldots,N, x = 1,\ldots,r\}$ state space of the network.
- G^r: the normalization constant.
- $P^r(\mathbf{k})$: steady state probability that the joint queue length is $\mathbf{k} \in S^r$.
- $P_j^r(k)$: steady state probability that there are k customers at node j.
- $p^r(\mathbf{k}) = G^r P^r(\mathbf{k})$ unnormalized steady state probability that the joint queue length is $\mathbf{k} \in S^r$.
- $P^{R-1,s}(\mathbf{k} - \mathbf{1}_j)$: the joint queue length distribution of the network with chain s removed.

Chapter 1

Introduction

1.1 Overview

In recent years, advanced technologies and new service requirements have resulted in the rapid development of computer systems and computer communication networks. Broadband ISDN(B-ISDN) will eventually provide all the services from voice, data transmission to video conference, etc. Compared to the conventional synchronous transfer mode, the traffic flow inside a B-ISDN network is highly stochastic, which makes the design and development of new products more complicated. As a result, performance modelling has never been more important. During the design and development phase of a system, modelling is the only way for predicting the system performance and evaluating the cost. When a system is built up, modelling is still an appropriate tool that assists in understanding the behavior of the system and improving its performance. Performance modelling is, in general, less laborious, more flexible than experiments and more reliable than intuition. In the field of computer network performance modelling, Petri Nets and queueing networks are two important and widely used modelling tools.

Petri Nets were introduced by C.A. Petri in 1962[63]. They are an effective modelling tool for the description and the analysis of concurrency and synchronization in parallel systems exhibiting the cooperative actions of different entities. Today, Petri Nets are widely used in modelling computer systems, computer communication networks and

manufacturing systems[5, 6, 23, 24, 54, 50, 52, 53, 31].

Since its origination, many extensions have been added to the basic Petri Net model to extend the use of Petri Nets in different modelling situations. The time concept has been introduced into Petri Nets for the quantitative performance analysis of systems. One of the most widely used timed Petri Nets is the Stochastic Petri Net[55, 59], which is also of our main interest.

The analysis of Stochastic Petri Nets is done by generating the associated Markov Chains and obtaining the performance of interest by solving the Markov Chains. The main reason for the widespread use of Stochastic Petri Net is due to the simplicity of the basic mechanism of the model, which is, however, paid for with the complexity of the description of large systems. The state space of the Markov Chain increases exponentially with the size of the model. In addition, a direct solution of a Markov Chain of N states typically requires about N^3 number of operations(multiplication and addition). When the system has many resources, the analysis of Stochastic Petri Nets becomes very time consuming. As a result, there is a need for efficient analysis methods.

Over the last decade, efforts have been made in finding efficient algorithms for the analysis of Stochastic Petri Nets. We will review them in Section 2.2. Among those works, Lazar and Robertazzi, Marson et al reported the finding of product form solutions for some Stochastic Petri Nets[51, 39, 40, 41]. In this thesis, we call them *Local Balance Stochastic Petri Nets*. The existence of product form solutions simplifies the analysis significantly. However, the boundary for the reported Local Balance Stochastic Petri Nets is very limited. There are many Stochastic Petri Nets that have product form solutions and are not included in the existing reports. In addition, there is a lack of systematic ways of identifying this class of Stochastic Petri Nets.

In this thesis, we begin by investigating the possibility of extending the boundary for Local Balance Stochastic Petri Nets. We propose a systematic procedure for dividing a Stochastic Petri Net into *subnets* and identifying Local Balance Stochastic Petri Nets. The Stochastic Petri Nets that passed the test procedure may be k-bounded for $k \geq 1$ and the transitions in a subnet do not have to be in sequence. In comparison, $k = 1$

and transitions in a subnet being in sequence are required in [41]. We prove that the Stochastic Petri Nets that have passed the test procedure have product form solutions. Therefore, our first contribution is extending the boundary for Local Balance Stochastic Petri Nets and proposing a systematic way of identifying the Stochastic Petri Nets that fall into the new boundary. As a result, more systems may be analyzed efficiently by Local Balance Stochastic Petri Nets and the identification of them is much easier.

Although Local Balance Stochastic Petri Nets have product form solutions and their associated Markov Chains are finite, they are different from the closed product form queueing networks both in their applications and their analysis. The Local Balance Stochastic Petri Nets have been used in modelling multiprocessor systems, computer communication protocols, etc[39, 40, 41, 73]. In fact, this class of Local Balance Stochastic Petri Nets is well suited for the modelling of systems with local balance and synchronization, which cannot be modelled by the BCMP queueing networks. In addition, the total population in a closed BCMP queueing network is constant whereas there is no such constraint for a Local Balance Stochastic Petri Net in general. In a Local Balance Stochastic Petri Net, a token may split into several tokens and several tokens may merge into one token. As a result, the computation algorithms available for the analysis of closed BCMP networks may not be applied to Local Balance Stochastic Petri Nets. Nevertheless, the theory of Decomposition and Aggregation may still be applied to the analysis of Local Balance Stochastic Petri Nets.

Decomposition and Aggregation is a method that may be applied to Markov Chains in general. With this technique, the problem of determining the equilibrium distribution is broken down into a set of smaller problems, the solution of which are combined to yield the distribution for the large systems. This technique is suited naturally to systems that are made up of a set of loosely coupled systems[22]. Ammar applied this technique to the analysis of Stochastic Petri Nets whose transition rates differ by several orders[1, 2]. The loosely coupled subsystems come into picture naturally. Approximations are obtained. However, the Stochastic Petri Nets that we will deal with in this thesis are strongly connected. Our next contribution is that by applying the Decomposition and

Aggregation techniques, we propose a *Decomposition by Subnet* method. This method resolves the analysis of a Local Balance Stochastic Petri Net into the analysis of subnets, the solutions of which are combined to obtain exact solutions for the original system. The Decomposition by Subnet method provides us with an insight into the Markov Chain structure of Local Balance Stochastic Petri Nets.

Our third contribution is that by decomposing according to a specific subnet, we are able to construct an aggregated net, which is that subnet with marking dependent firing rates proportional to its original firing rates, to represent the original system. We call this procedure the *Norton's theorem for Local Balance Stochastic Petri Nets*. This theorem enables us to present the original system in a concise and yet exact way. In addition, the aggregated net facilitates the parametric analysis of that particular subnet of interest with respect to its transition rates. Application of the aggregation process directly at the Stochastic Petri Net level is much more advantageous and desirable than at the Markov Chain level. It is, of course, more difficult due to the complex structure of the Stochastic Petri Nets.

There is a significant difference between the Norton's theorem developed in this thesis and the Norton's theorem for queueing networks. In Norton's theorem for queueing networks there is a flow equivalent queue representing several queues in the original systems. In our case, however, there is no flow equivalent part in the aggregated Stochastic Petri Net representing the rest of the Stochastic Petri Net.

Norton's theorem may be applied to parametric analysis for Local Balance Stochastic Petri Nets. When the system is not a Local Balance Stochastic Petri Net, there is also a need for parametric analysis, because analyzing the whole system repeatedly with different firing rates is a tedious job. Unfortunately, to the best of our knowledge, no one has considered this problem. Our forth contribution is, therefore, applying the so called "*ideal aggregate*" in the Decomposition and Aggregation theory to develop an efficient parametric analysis method for Stochastic Petri Nets in general. We exploit the possibility of confining the firing rate of interest into a small diagonal matrix. Every time that firing rate is changed, only a Markov Chain of the order of the small diagonal matrix needs to

be analyzed. Therefore, this method significantly reduces the number of operations.

Queueing networks are another modelling tool for computer network performance modelling. The analysis of queueing networks is also based on the Markov Chain concepts. However, a class of queueing networks, called BCMP networks[4], enjoys closed form solutions, namely product form solutions. The existence of closed form solutions simplifies the analysis procedure greatly. The analysis is no longer performed at the Markov Chain level. Instead, simple algebra operations are used to calculate a normalization constant or mean performance values directly. Algorithms like Convolution[9, 10, 67], MVA[66], RECAL[15, 16], MVAC[17] and DAC[70] were developed for that purpose.

Among the existing algorithms, the complexity of Convolution[9, 10, 67] and MVA[66] is linear with the number of nodes and exponential with the number of chains in the network. As a result, they are efficient in dealing with networks consisting of many nodes and a few chains. On the other hand, the complexity of RECAL[15, 16], MVAC[17] and DAC[70] is polynomial with the number of chains and exponential with the number of nodes in the network. As a result, they are efficient in dealing with networks consisting of many chains and a few nodes. An open problem is that the algorithms themselves are having excessive computational requirements when the queueing network is getting very large. There is always a need for more efficient algorithms.

In search of efficient algorithms for the analysis of the BCMP networks, Conway and Georganas exploited the possibility of applying the theory of Decomposition and Aggregation and established a methodology for developing efficient algorithms[14]. They showed that all the major existing algorithms, e.g. Convolution, MVA, RECAL, MVAC, DAC, may be derived through decomposing and aggregating of queueing networks. Based on their results, our fifth contribution is that we show that the efficiency of DAC may be further improved by avoiding computing the mean values in each recursive step. We call the resulting algorithm the Improved Distribution Analysis by Chain(IDAC) algorithm. In addition, IDAC has the benefits in terms of reducing possible numerical errors.

We then apply the Decomposition and Aggregation theory and propose an *Independent Decomposition and Aggregation* method for the analysis of queueing networks.

Our method differs from that of Conway and Georganas' in that we first transform a BCMP network into a network consisting of only Infinite Server queues. In comparison, in [14], a BCMP network is first transformed into a network consisting of only Processor Sharing nodes. By the Independent Decomposition and Aggregation method, we derive an Adaptive Convolution by Chain Algorithm(ACCAL). ACCAL is efficient in dealing with networks with many chains and a few nodes. Compared with RECAL, MVAC, DAC or IDAC, ACCAL has three distinct characteristics: a) faster, it has smaller computational requirements than IDAC does; b) adaptive, according to the number of nodes in the network and other information, the algorithm converts the network into an equivalent one that can be more efficiently analyzed by the algorithm. c) parallel Processing, the algorithm has three independent parts that may be executed in parallel in a multiprocessor system. In this way, the computational time is further reduced. Therefore, our sixth contribution is that we propose the Independent Decomposition and Aggregation method and accordingly, we derive the ACCAL algorithm.

As may be seen, Decomposition and Aggregation theory plays an important role in our research. In the next Section, we will give a review of the theory.

1.2 The Theory of Decomposition and Aggregation

The fundamental theory of Decomposition and Aggregation is attributed to the work of Simon and Ando [69] in the early 1960's. In the analysis of economic systems, they presented the Decomposition and Aggregation techniques for analyzing systems that are *nearly completely decomposable*. In such systems, there exists a number of interacting subsystems such that the interactions within each subsystem are stronger than those between the individual subsystems. The main feature of the Decomposition and Aggregation is reducing the analysis of a large system into that of a set of smaller problems. This technique is particular attractive for large scale complex systems, since it reduces the inherent analytical and computational difficulties.

Applying this method to the analysis of Markov Chains, it is assumed in general,

that there exist groups of communicating states such that the coupling among the states belonging to the same group is strong relative to that between states of different groups. Formally, let $\mathbf{Q}$ be the infinitesimal generator matrix of a continuous time Markov Chain. Assume $\mathbf{Q}$ is an $n \times n$ matrix. $\mathbf{Q}$ is partitioned into $M \times M$ matrices as shown in Figure 1.1. Assume the order of $\mathbf{Q}_{ii}$ is $n(i)$. Further let

$$\mathbf{Q} = \begin{vmatrix} \mathbf{Q}_{11} & \mathbf{Q}_{12} & \cdots & \mathbf{Q}_{1M} \\ \mathbf{Q}_{21} & \mathbf{Q}_{22} & \cdots & \mathbf{Q}_{2M} \\ & & & \\ \mathbf{Q}_{M1} & & \cdots & \mathbf{Q}_{MM} \end{vmatrix}$$

Figure 1.1: The Partition of the Matrix Q

$$\mathbf{Q} = \mathbf{Q}^* + \epsilon \mathbf{C}$$

$\mathbf{Q}^* = diag(\mathbf{Q}_1^*, ...\mathbf{Q}_M^*)$. $\mathbf{Q}^*$ is an infinitesimal generator matrix obtained from $\mathbf{Q}$ by distributing the quantities of the off diagonal submatrices over the diagonal submatrices. ϵ is a real positive number, small compared to the elements of $\mathbf{Q}^*$. $\mathbf{C}$ is a square matrix of the same order as $\mathbf{Q}^*$. The quantity ϵ is referred to as the maximum degree of coupling between the subsystems $\mathbf{Q}_{ii}$. If ϵ is sufficiently small, the $\mathbf{Q}$ is said to be *nearly-completely decomposable*[19].

To solve $\pi\mathbf{Q} = \mathbf{0}$ or $\pi(\mathbf{Q} + \mathbf{I}) = \pi$, we can apply Simon and Ando's Decomposition and Aggregation method for obtaining an approximation $\mathbf{z}$ for π as follows[14]:

1. Decomposition. Obtain $\mathbf{Q}^*$ from $\mathbf{Q}$. Solve $\mathbf{v}^*(\mathbf{Q}^* + \mathbf{I}) = \mathbf{v}^*$ subject to $\mathbf{v}_k^* \mathbf{1}^T = 1$ for $1 \leq k \leq M$ where $\mathbf{v}^* = (\mathbf{v}_1^*, ..., \mathbf{v}_M^*)$ and $\mathbf{v}_j^* = (v_{1j}^*, ...v_{n(j)j}^*)$
2. Aggregation. Obtain an aggregated system infinitesimal generator A of order

M with its $(i,j)th$ element a_{ij} as:

$$a_{ij} = \begin{cases} \mathbf{v}_i^* \mathbf{Q}_{ij} \mathbf{1}^T & \text{for } i \neq j \\ -\sum_{k=1, k\neq i}^{M} a_{ik} & \text{for } i = j \end{cases} \tag{1.1}$$

Solve $\mathbf{u}(\mathbf{A} + \mathbf{I}) = \mathbf{u}$ subject to $\mathbf{u}\mathbf{1}^T = 1$

3. Compute the approximation for π as $\mathbf{z} = [(u_1 \mathbf{v}_1^*), \ldots, (u_M \mathbf{v}_M^*)]$

where $\mathbf{1}^T$ is a column vector whose elements are all 1s.

If any of the subsystem $\mathbf{Q}_i^*$s are themselves nearly-completely decomposable, a hierarchical Decomposition and Aggregation technique may be applied.

There are many ways of distributing the off diagonal submatrices over the diagonal matrices. Most of them give approximation results. The resulting error vector $(\mathbf{z} - \pi)$ depends completely on the manner in which the $\mathbf{Q}^*$ is constructed [19]. The following two theorems give the necessary and sufficient conditions for exact decomposition and aggregation.

Theorem 1 *([19]) The Decomposition and Aggregation procedure yields the exact equilibrium distribution π for the continuous-time Markov Chain with infinitesimal generator $\mathbf{Q}$, if, and only if, $\mathbf{Q}^*$ is constructed in such a way that for $1 \leq i \leq M$, the distribution $\mathbf{v}_i^*$ is actually equal to the true conditional distribution $G_i^{-1}\pi_i$, where $G_i = \sum_{j=1}^{n(i)} \pi_{ji}$.*

This theorem is totally equivalent to the following one:

Theorem 2 *([72]) Decomposition and aggregation yields the exact equilibrium distribution if, and only if, $\mathbf{Q}^*$ is constructed in such a way that for $1 \leq k \leq M$, the distribution $\mathbf{v}_k^*$ is subparallel to π_k.*

There exist the so called *ideal aggregates*[21] that nullify the vector error $(\mathbf{z} - \pi)$. For simplicity, consider one subsystem only, e.g. $\mathbf{Q}_1$. $\mathbf{Q}$ is as shown in Figure 1.2. where $\mathbf{Q}_2$ represents the remaining $(M-1)$ subsystems and their mutual interactions. Let

$$\mathbf{Q}_1^* = \mathbf{Q}_1 + \mathbf{S}$$

It is shown[21, 74] that if S is chosen as:

$$S = E(I - Q_2)^{-1}F \tag{1.2}$$

then the Decomposition and Aggregation procedure gives exact results.

$$Q = \begin{vmatrix} Q_1 & E \\ F & Q_2 \end{vmatrix}$$

Figure 1.2: The Partition of the Matrix Q

1.3 Thesis Structure

This thesis is divided into two parts. The first part deals with Stochastic Petri Nets and the second part deals with queueing networks.

In Part I, Chapter 2 gives an introduction to Stochastic Petri Nets, reviews the efforts that have been taken for developing efficient analysis methods. The contributions of the thesis on Stochastic Petri Nets are presented in Chapters 3, 4 and 5. In chapter 3, we extend the boundary for Local Balance Stochastic Petri Nets, prove that the Stochastic Petri Nets that fall into the new boundary have product form solutions and propose a systematic way for identifying the Local Balance Stochastic Petri Nets. In Chapter 4, we propose the Decomposition by Subnet method and through it, develop the Norton's theorem and show its application to parametric analysis. In Chapter 5, we present an efficient parametric analysis algorithm for Stochastic Petri Nets in general.

In part II, Chapter 6 gives an introduction to queueing networks especially the BCMP networks, reviews the different Decomposition and Aggregation methods that have been applied to queueing networks and briefly summarizes the existing algorithms. The contributions of this thesis on queueing networks are presented in Chapters 7 and 8.

In Chapter 7, we present the Improved Distribution Analysis by Chain algorithm(IDAC). In Chapter 8, we propose the Independent Decomposition and Aggregation method and through it, develop the Adaptive Convolution by Chain algorithm.

In Chapter 9, we conclude by summarizing the contributions of this thesis and suggesting further research topics.

The C language program for the identification of Local Balance Stochastic Petri Nets is listed in Appendix A.

1.4 Publications

The publications resulted from this research are:

1. M. Li, N.D. Georganas, "Exact Parametric Analysis of Stochastic Petri Nets " IEEE Trans. on Computers,1992(in press).
2. M. Li, N.D. Georganas, "Parametric Analysis of Stochastic Petri Nets " Fifth International Conference on Modelling and Tools for Computer Performance Evaluation, Torino, Italy, Feb. 1991.
3. M. Li, N.D. Georganas, "IDAC-Improved Distribution Analysis by Chain Algorithm" Submitted for Publication,1991.
4. M. Li, N.D. Georganas, "Norton's Theorem in the Analysis of a Class of Stochastic Petri Nets" Submitted for Publication, 1990.
5. M. Li, N.D. Georganas, "ACCAL- an Efficient Adaptive Convolution by Chain Algorithm" Submitted for Publication, 1991.

Part I

Stochastic Petri Nets

Chapter 2

Introduction

Petri Nets, introduced by C.A. Petri[63], are an abstract formal graph model used for modelling systems that exhibit concurrency and synchronization. In this chapter, we give an introduction to Stochastic Petri Nets, review the different ways that have been taken for the efficient analysis of Stochastic Petri Nets. Definitions and notations that will be used in later chapters are also introduced.

2.1 Introduction to Stochastic Petri Nets

In the development of Petri Net theory, there have been two major phases, namely, Standard Petri Nets, i. e. Petri Nets without time, and Timed Petri Nets, i.e. Petri Nets with time.

2.1.1 Standard Petri Nets

A Standard Petri Net is composed of four parts[62]: a set of places P, a set of transitions T, a set of transition input arcs A_i and a set of transition output arcs A_o. In the graphical representation of Standard Petri Nets, places are drawn as circles and transitions as bars. Arcs connect transitions to places and places to transitions. A place is an input to a transition if an arc exists from the place to the transition. A place is an output of a transition if an arc exists from the transition to the place. A formal definition

of a Standard Petri Net is thus the following[53]:

$$\begin{aligned} PN &= (P, T, A) \\ P &= \{p_1, ..., p_n\} \\ T &= \{t_1, ..., t_m\} \\ A_i &= (P \times T) \\ A_o &= (T \times P) \\ A &= (A_i \cup A_o) \end{aligned} \tag{2.1}$$

An important concept for Standard Petri Nets is a *token*. It is graphically drawn as black dots in places. A *marking* μ is an assignment of tokens to the places. It is usually denoted by $\mu = (m_1, ..., m_n)$. Where m_i is the number of tokens in place i and

$$\mu_0 = (m_{01}, ... m_{0n}) \tag{2.2}$$

is used to denote the initial marking, where m_{0i} denotes the number of tokens in place p_i in the initial marking μ_0. For the convenience of presentation, in the examples of this thesis, we will use the numbers of place or the names of place to represent a marking. For example, marking (178(2)) or ($p_1p_7p_8(2)$) means that p_1 and p_7 each has one token and p_8 has two tokens. Occasionally, we also use $m_j(\mu)$ to denote the number of tokens in p_j for marking μ.

The rules concerning the execution of a Petri Net are:

1. A transition is enabled when all of its input places contain at least one token.
2. An enabled transition can fire, thus removing one token from each of its input places and depositing one token into each of its output places.
3. Each firing of a transition modifies the distribution of tokens on places and thus produces a new marking for the Petri Net.

A marking μ' is defined to be *reachable* from μ, if there exists a sequence of transition firings that changes the Petri Net marking from μ to μ'. A *Reachability Graph* $R(\mu_0)$

of a Petri Net is the set of all the markings that are reachable from μ_0. In $R(\mu_0)$, nodes represent markings generated from μ_0 and its successors and each arc represents a transition firing, which transforms one marking to another. A transition t_j is *live* in a marking μ, if there exists a marking μ' reachable from μ and t_j is enabled in μ' [62].

A Petri Net is k-bounded, if the number of tokens in every place does not exceed k. When k is one, the Petri Net is said to be *safe*[62]. A Petri Net is a *state machine*, if every transition has only one input arc and one output arc.

The $(n \times m)$ *incidence matrices* of a Petri Net are denoted by D^-, D^+ and $D = D^+ - D^-$. An element d_{ij}^- in D^- is the number of arcs from place i to transition j(i.e. inputs to transition j). An element d_{ij}^+ in D^+ is the number of arcs from transition j to place i(i.e, outputs from transition j). The ith column and jth row of D^+ are written as $D^+(.,i)$ and $D^+(j,.)$ respectively.

Standard Petri Nets have been applied to communication protocol specification and verification[5, 6, 23, 24], distributed software system analysis[3, 57], manufacturing system specification[54], etc. Refer to [58] for more references.

2.1.2 Stochastic Petri Net

The introduction of time into Standard Petri Net models allows the description of the dynamic behavior of a system, i.e. the state evolution and the duration of each action of the system. There exist several ways of introducing time into a Standard Petri Nets[55, 59, 75]. The most widely accepted one is the Stochastic Petri Nets, introduced independently by Molloy[55] and Natkin[59]. A Stochastic Petri Net is obtained by associating with each transition in a Petri Net an exponentially distributed random variable that expresses the delay from the enabling to the firing of the transition. A formal definition of a Stochastic Petri Net is the following:

$$SPN = (P, T, A, \mu_0, R) \tag{2.3}$$

where P, T, A and μ_0 are as in 2.1 and 2.2 and

$$R = \{r_1, \ldots, r_m\} \tag{2.4}$$

is a set of possibly marking dependent firing rates associated with the Stochastic Petri Net transitions. When necessary the dependence upon a given marking μ of the firing rate of transition t_i will be denoted $r_i(\mu)$.

Stochastic Petri Nets are shown to be isomorphic to continuous time Markov Chains[55]. The reachability set $R(\mu_0)$ of a Stochastic Petri Net constitutes the state space of the associated Markov Chain. The transition rate from state i(marking μ_i) to state j(marking μ_j) is equal to the firing rate of the transition that is enabled by marking μ_i and whose firing generates marking μ_j. A Stochastic Petri Net is ergodic, if it generates an ergodic continuous time Markov Chain. It is possible to show that a Stochastic Petri Net is ergodic, if μ_0, the initial marking, is reachable from any $\mu_i \in R(\mu_0)$[56].

The analysis of the Stochastic Petri Nets may fall into two kinds[55]:

First, if the associated Markov Chain has absorbing states, the expected number of steps until one of the absorbing states is reached may be calculated[32] and other parameters of interest can be obtained from it.

Second, if the Markov Chain is ergodic, the steady state probability distribution of markings is computed by solving:

$$\pi \mathbf{Q} = \mathbf{0} \tag{2.5}$$

with the additional constraint

$$\sum_i \pi_i = 1$$

where $\mathbf{Q}$ is the infinitesimal generator and π is the vector of the steady state probabilities. Other parameters of interest may be obtained from π. In this thesis, only ergodic Stochastic Petri Nets are considered.

Stochastic Petri Nets have been applied to the analysis of multiprocessor systems[50, 52, 53], communication networks[31],etc. Refer to [58] for more references.

A major difficulty in modelling systems by Stochastic Petri Nets is the explosion of the Markov Chain state space when the system is getting complicated. This has always been an open problem. In the remainder of this Chapter, we will review the efforts that have been made in dealing with this problem.

2.2 Review of Literature

Stochastic Petri Nets are a simple yet very useful modelling tools. As mentioned in Chapter 1, the simplicity is, however, paid with the complexity in analysis when the system being modelled is getting complicated. When the model is becoming large, the state space of the associated Markov Chain tends to explode. As a result, the numerical solution of Equation 2.5 is intractable.

For the last two decades, people have been trying to solve this problem through different approaches. We review their efforts in this section. As may be seen, each method has its strength and weakness. In all the cases, the structure of the Stochastic Petri Nets as well as their associated Markov Chains plays a paramount important role.

The first possibility is to take into account the sparsity of the transition matrix. Numerical solution techniques for solving sparse Markov Chains are available, e.g. [11]. However, this technique reaches its limit when the state space becomes too large.

Then there is the matrix geometric solution method[60, 26]. This method is efficient because it makes use of small matrices. However, it is entirely based on special matrix structures that may not exist or be foreseen in many models.

The use of compact or folded models like Colored Stochastic Petri Nets[34, 25, 42] is an interesting approach. On top of it, the High Level Petri Net method[48, 13] is proposed. It explores the symmetries of systems. The associated Markov Chains of these systems satisfy the lumpability condition[28]. Therefore, instead of generating a Reachability Graph, this method generates a Symbolic Reachability Graph. The time and space requirements of the method depend on the number of markings of the Symbolic Reachability Graph that may be much less than the Reachability Graph. Symmetric behavior is a necessary condition for applying this method.

Giglmayr[29, 30] applied the technique of Decomposition and Aggregation described in Section 1.2 to break a large transition rate matrix into small parts, each of them being of appropriate size, to handle by available computing facility. However, except for some special cases, the total number of computations is at least the same as, if not more than,

that of a direct solution.

Approximate solutions are an interesting approach. Ciardo[12] proposed an approximate method based on the concept of nearly independency. Ammar[1, 2] proposed a time scale decomposition algorithm for a class of Stochastic Petri Net models whose transition rates differ by several orders of magnitude. Approximations are, in general, faster that exact solutions. However, they are always haunted by problems such as error bounds, convergency, etc.

Bubhholz[8] presented a hierarchical description of the models as opposed to "flat models". This method reduces the space requirement significantly. Time requirement is not reduced as much.

A completely different approach from all of the above is the finding of product form solutions for some Stochastic Petri Nets. Marson et al[51] presented a class of multiple bus, multiprocessor system models. The special property of the associated Markov Chains results in product form solutions. However, their results only apply to that particular class of multiple bus, multiprocessor systems.

Florin et al[27] showed that for closed synchronized queueing networks, which may be modelled by Stochastic Petri Nets, the steady state probability distribution can be expressed using matrix products. However, at least for the time being, the method itself has only theoretical interest. The authors admitted the difficulties in solving even small Stochastic Petri Net models.

Lazar and Robertazzi[41, 68] presented another class of product form Stochastic Petri Nets. As long as a Stochastic Petri Net satisfies the conditions given in [41], its steady state probability distribution will satisfy local balance equations. Therefore, this class is more general than the above two cases. We call this class of Stochastic Petri Nets the *Local Balance Stochastic Petri Nets*. This class of Stochastic Petri Nets is well suited for the modelling of systems with local balance and synchronization, which cannot be modelled by product form queueing networks.

The existence of analytical form solutions allows a much more computational efficient method. Instead of trying to solve the global balance equations, all we need is to calculate

the normalization constant in a much simpler way. However, the boundary for the reported Local Balance Stochastic Petri Nets is limited and there is a lack of systematic ways for identifying the Local Balance Stochastic Petri Nets.

Frequently, in the performance analysis by Stochastic Petri Nets, the behavior of the system with respect to one parameter, or one transition, is of interest. Assigning different values to this transition, the effect of this particular transition on the whole system performance is obtained. One drawback of this analysis is that every time the transition rate is changed, the whole net has to be analyzed again. Unfortunately, to our knowledge, no one has dealt with this problem of parametric analysis of Stochastic Petri Nets.

Chapter 3

Local Balance Stochastic Petri Nets

As mentioned in Section 2.2, there are some Stochastic Petri Nets that have product form solutions. However, the Stochastic Petri Nets falling inside the existing boundary for Local Balance Stochastic Petri Nets are very limited. In addition, there is a need for systematic way for identifying this class of Stochastic Petri Nets.

In this Chapter, we extend the boundary, present a systematic way for identifying the Local Balance Stochastic Petri Nets that fall into the new boundary, prove that they have product form solutions and compare our results to that of Lazar and Robertazzi's[41].

The Stochastic Petri Nets considered in this Chapter are k-bounded with $k \geq 1$.

3.1 Identifying subnets

In order to identify subnets, we need the following definitions (some of them are analogous to the electric circuit application of Norton's theorem).

- *Resource place.* A place is a resource place, if a token in it represents that the resource is available.
- *Resource token.* A resource token is the token residing in a resource place.
- *Dormant state of a transition.* A transition is in a dormant state when it does not fire even if it is enabled. Otherwise, the transition is said to be in a *normal* state.

- *Dormant state of a token.* A token is in a dormant state if it cannot move and cannot enable transitions.

- *A cycle of transitions.* Starting from one marking, fire a series of transitions till that marking is reached again. The series of transitions constitutes a cycle of transitions.

Note that a similar definition of resource token was given in [1].

Start with initial marking μ_0 where the system is idle; resources are not used. Choose one non-resource place that contains at least one initial token and put tokens in other non-resource places into the dormant state. Find out all the cycles of transitions S_i. Divide the cycles into groups. S_i is in a group, if, and only if, it has at least one common transition with one of the cycles in that group. One group of transitions and their input/output places constitutes a *subnet*. Two subnets may have some common places. When the common place is a resource place, it is called a *common resource place*. When there is more than one subnet, they may have more than one common resource places. However, there is only one common non-resource place among them, which is the place that contains the initial non-resource tokens.

For example in Figure 3.1, p_7 is a *resource place* and the token in p_7 is a resource token. The tokens in place p_1 and p_8 are non-resource tokens. Put the token in p_8 into the dormant state. Starting from the initial marking (178), three cycles of transitions are identified. $S_1 = (t_1t_2t_3t_4)$, $S_2 = (t_5t_6t_7)$ and $S_3 = (t_8t_7)$. S_2 and S_3 both have t_7 in their cycles. Hence, they are in one group. S_1 is in another group. Therefore transitions t_1, t_2, t_3, t_4 and places p_1, p_2, p_3, p_4, p_7 constitute one subnet while t_5, t_6, t_7, t_8 and p_1, p_5, p_6 constitute another subnet. p_1 is a common non-resource place of the two subnets.

Restart with the initial marking. Put the transitions in the identified subnets into the dormant state. Take another non-resource place that contains at least one initial token and put tokens in other non-resource places into the dormant state. Repeat the above steps to identify other subnets. For a subnet identified this time, it may have more than one common resource places with subnets identified before, but no common non-resource places with them.

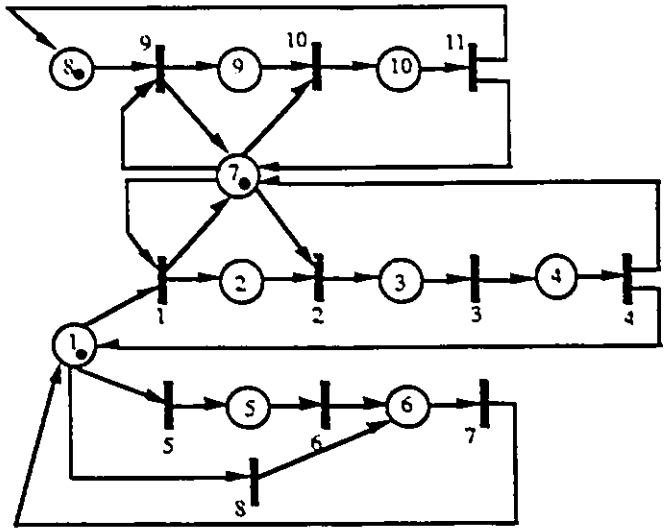

Figure 3.1: An Example of Local Balance Stochastic Petri Net.

For example in Figure 3.1, after putting the transitions one to eight into the dormant state, $S_4 = (t_9 t_{10} t_{11})$ is identified and it is the only cycle left. Hence, transitions t_9, t_{10}, t_{11} and places p_7, p_8, p_9, p_{10} constitute another subnet. In the following, we will refer to this subnet as net one, the subnet consisting of t_1, t_2, t_3, t_4 and p_1, p_2, p_3, p_4, p_7 as net two and the subnet consisting of t_5, t_6, t_7, t_8 and p_1, p_5, p_6 as net three. p_7 is a *common resource place* of subnets one and two. We will use C_i to denote subnet i.

Restart with the initial marking. Put the transitions in the identified subnets into the dormant state. Repeat the steps until all the transitions are put into the dormant state. Hence, all the subnets are identified. If after putting part of the transitions into the dormant state, as described above, some of the remaining transitions are not live under the initial marking, or if there is only one subnet in the system, the Stochastic Petri Net cannot be separated into subnets and we cannot apply to it the theory developed in the following.

Because of the way the subnets are identified, there may be more than one common resource place for a pair of subnets. However, there is at most one common non-resource place for a pair of subnets.

To draw the subnets separately, a common place belonging to L subnets is duplicated into L places, each residing in one of the L subnets. The number of tokens in each place is equal to that in the original Stochastic Petri Net.

Further divide the identified subnets into three categories:

1. *Safe subnet.* A subnet is safe, if the number of tokens in any non-resource place does not exceed one (No restrictions on the number of tokens in resource places).
2. *Unsafe Subnet I.* A subnet is an Unsafe Subnet I, if the number of tokens in any non-resource place may exceed one and there is at least one common resource place in the subnet (No restrictions on the number of tokens in resource places).
3. *Unsafe Subnet II.* A subnet is an Unsafe Subnet II, if the number of tokens in any non-resource places may exceed one and there is no common resource places in the subnet.

For all the above cases, the number of resource tokens in a resource place may be an arbitrary finite integer.

In addition, define:

- *Basic Markov Chain.* A Basic Markov Chain is the Markov Chain associated with a subnet.
- *Basic Subnet Marking.* A Basic subnet marking is a marking in the Basic Markov Chain. We use $\hat{\mu}_l^j$ to present a Basic Subnet j marking and $\hat{\mu}_0^j$ to represent the initial marking for subnet j.
- *Single Token Markov Chain.* A Single Token Markov Chain is the Markov Chain of a subnet when there is only one initial non-resource token in the subnet. When the subnet is Safe, its Single Token Markov Chain is its Basic Markov Chain.

Figure 3.2 shows the Markov Chain of the Stochastic Petri Net in Figure 3.1. Figures 3.3 to 3.5 show the subnets of the Stochastic Petri Net and their Basic Markov Chains.

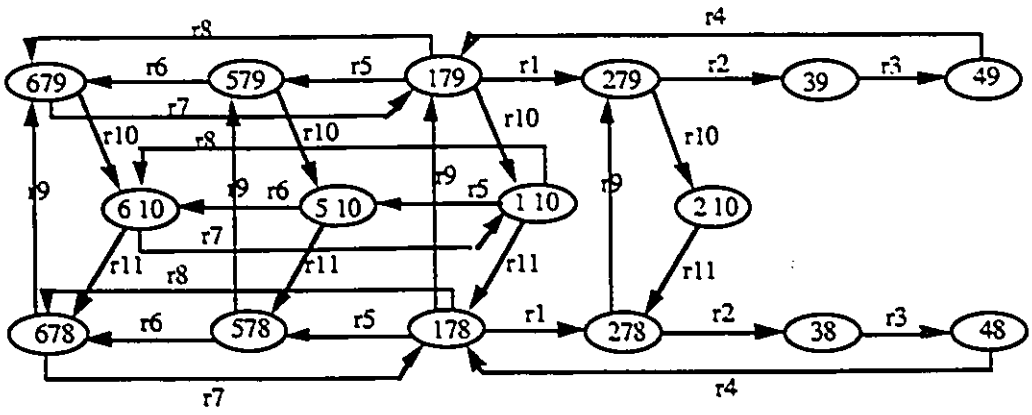

Figure 3.2: The Markov Chain of the Stochastic Petri Net

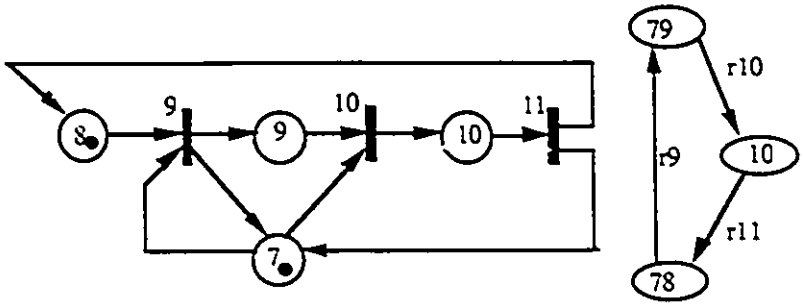

Figure 3.3: Subnet One of the Stochastic Petri Net and Its Markov Chain

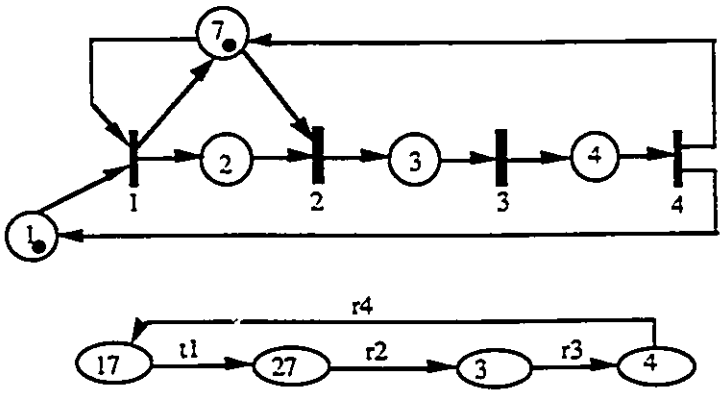

Figure 3.4: Subnet Two of the Stochastic Petri Net and Its Markov Chain

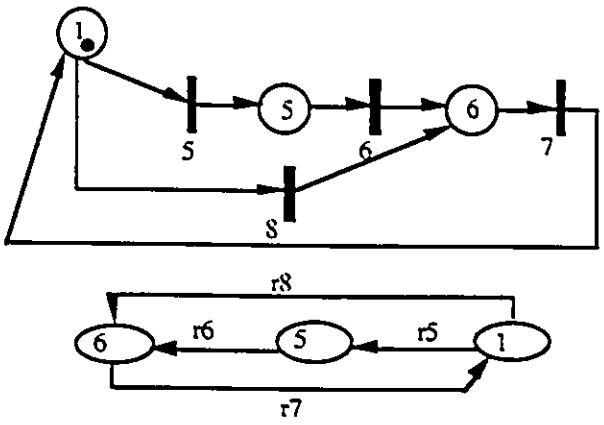

Figure 3.5: Subnet Three of the Stochastic Petri Net and Its Markov Chain

3.2 Product Form Solutions, An Informal Presentation

In Section 3.1, we showed how to identify and separate subnets. Let us look at another example shown in Figure 3.6. p_9 is a resource place. According to the steps described in Section 3.1, we may easily identify two subnets. We draw them separately in Figure 3.7. Call the upper subnet net one and the lower subnet net two. Figures 3.8 and 3.9 show the Basic Markov Chains of the two subnets respectively. Figure 3.10 shows the Single Token Markov Chain of net two.

Now suppose that there is another Stochastic Petri Net that consists of the two independent subnets in Figure 3.7. The Markov Chain of that system is shown in Figure 3.11. It is an interlacement of the Single Token Markov Chains of the two subnets. For example, the transitions from marking (56129) to (562(2)9) to (5623) and back to (56129) is the Single Token Markov Chain of subnet two. The transitions among (782(2)9), (582(2)9), (672(2)9), (562(2)9) and (42(2)9) is the Single Token Markov Chain of subnet one.

The probability of a marking in Figure 3.11 is equal to the product of the probabilities of the corresponding Basic Subnet Markings. For instance,

$$P(2358) = P_1(58) \times P_2(23) \tag{3.1}$$

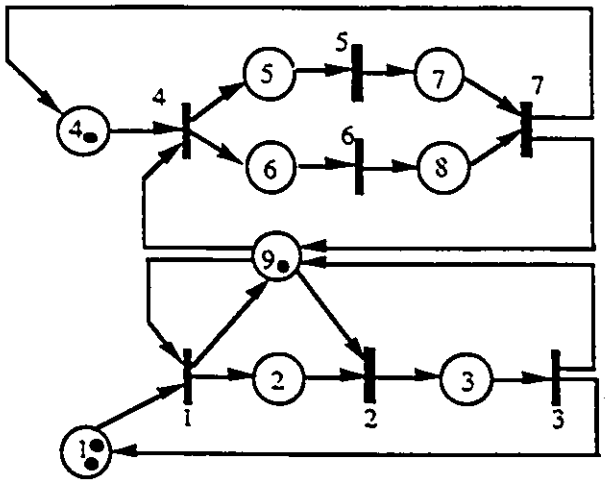

Figure 3.6: An Example of Local Balance Stochastic Petri Net

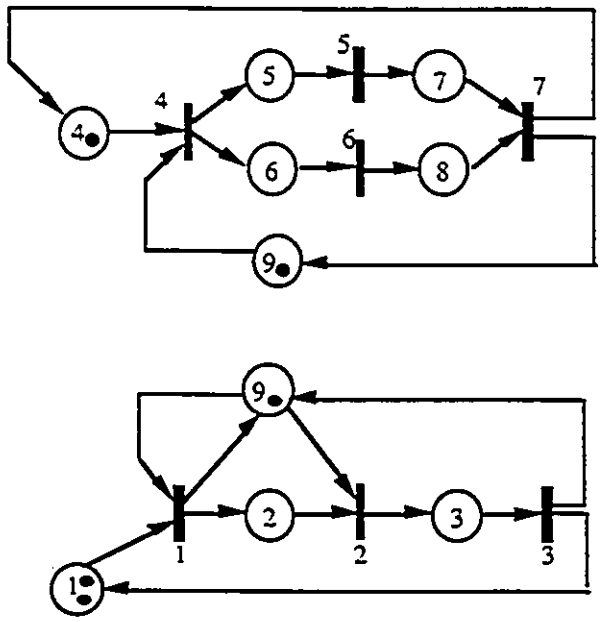

Figure 3.7: Basic Subnets

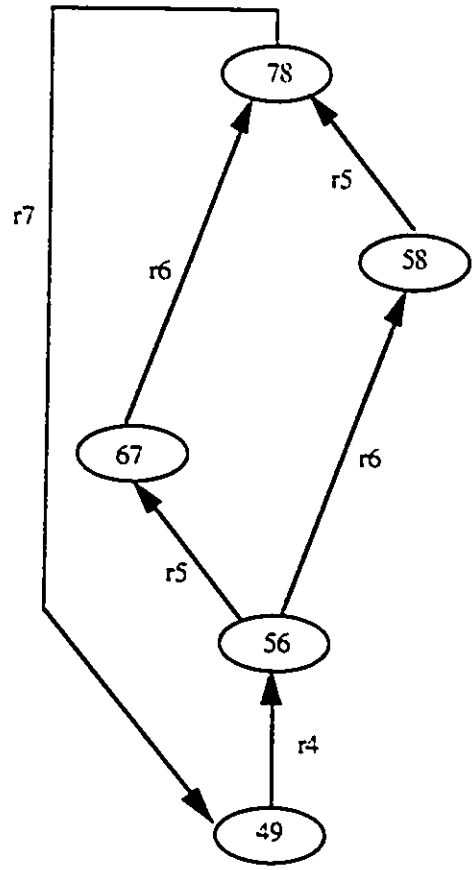

Figure 3.8: The Basic Markov Chain of Net One

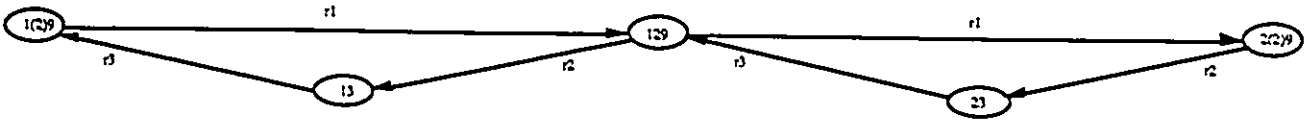

Figure 3.9: The Basic Markov Chain of Net Two

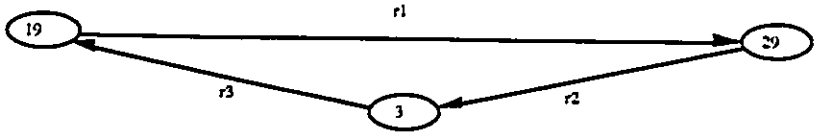

Figure 3.10: The Single Token Markov Chain of Net Two

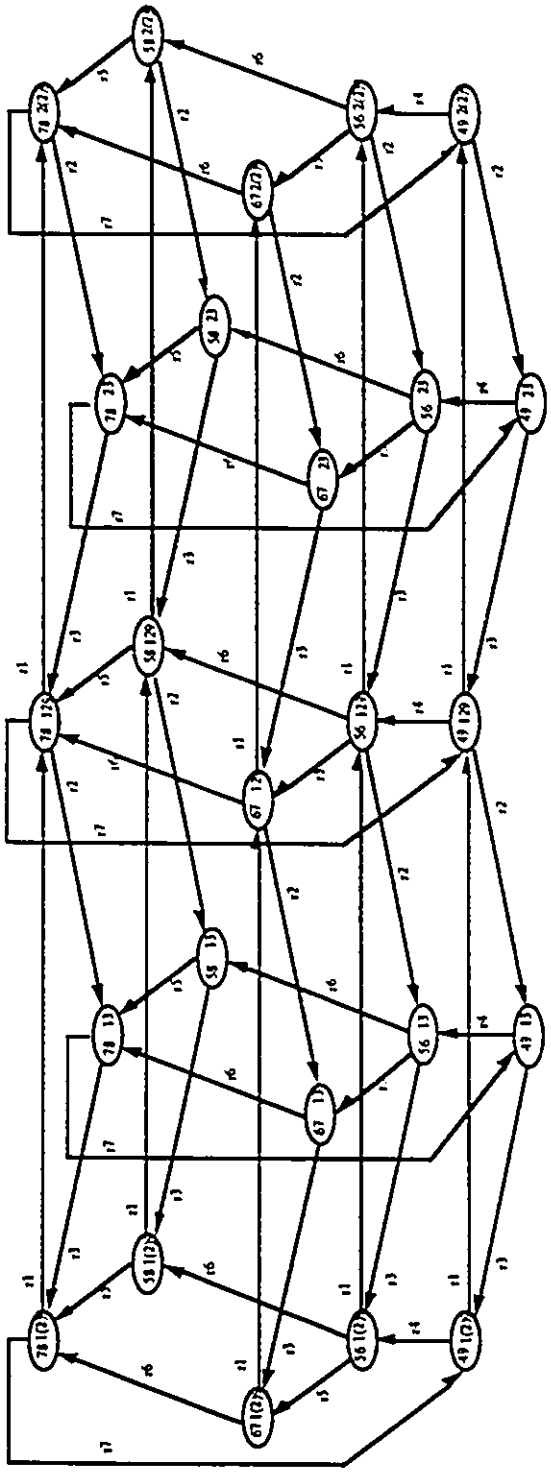

Figure 3.11: The Markov Chain of the System of Independent Subnets

This is obvious, because the two subnets are independent. Also, it may be easily proved by showing that for a given marking, its probability satisfies the global balance equations of each subnet and the sum of the subnet balance equations is in fact the global balance equation of the system.

In general, if a Stochastic Petri Net consists of L independent subnets, the probability of a marking of the system is equal to the product of the probabilities of the corresponding Basic Subnet Markings. In fact, the marking probability satisfies the global balance equations of each subnet and the sum of the subnet balance equations is the global balance equation of the original Stochastic Petri Net.

Now, suppose that the two subnets in Figure 3.7 share a common resource as shown in Figure 3.6. As a result, if one subnet is using the resource, the other may not use it. Therefore, the Markov Chain associated with the Stochastic Petri Net in Figure 3.6 will differ from that in Figure 3.8 in three ways:

1. Removal of states. The states in Figure 3.8 representing that both subnets are using the resource have to be removed. For example state(6713) in Figure 3.11 has to be removed.

2. Removal of transitions. The transitions to and from the removed states should also be removed.

3. Change of marking presentation. If the resource is used by one subnet, the resource token will not show up in the marking. For example, marking (78129) should be replaced by (7812), because subnet one is using the resource.

The Markov Chain of the Stochastic Petri Net in Figure 3.6 is shown in Figure 3.12.

Is the product form solution still valid after these changes? Intuitively, since the product form satisfies the subnet global balance equations, if the total result is nothing but a removal of some Basic Markov Chains from the original Markov Chain, the product form is still valid. In fact, a marking probability still satisfies the global balance equations of the remaining subnets and the sum of these subnet global balance equations is still the

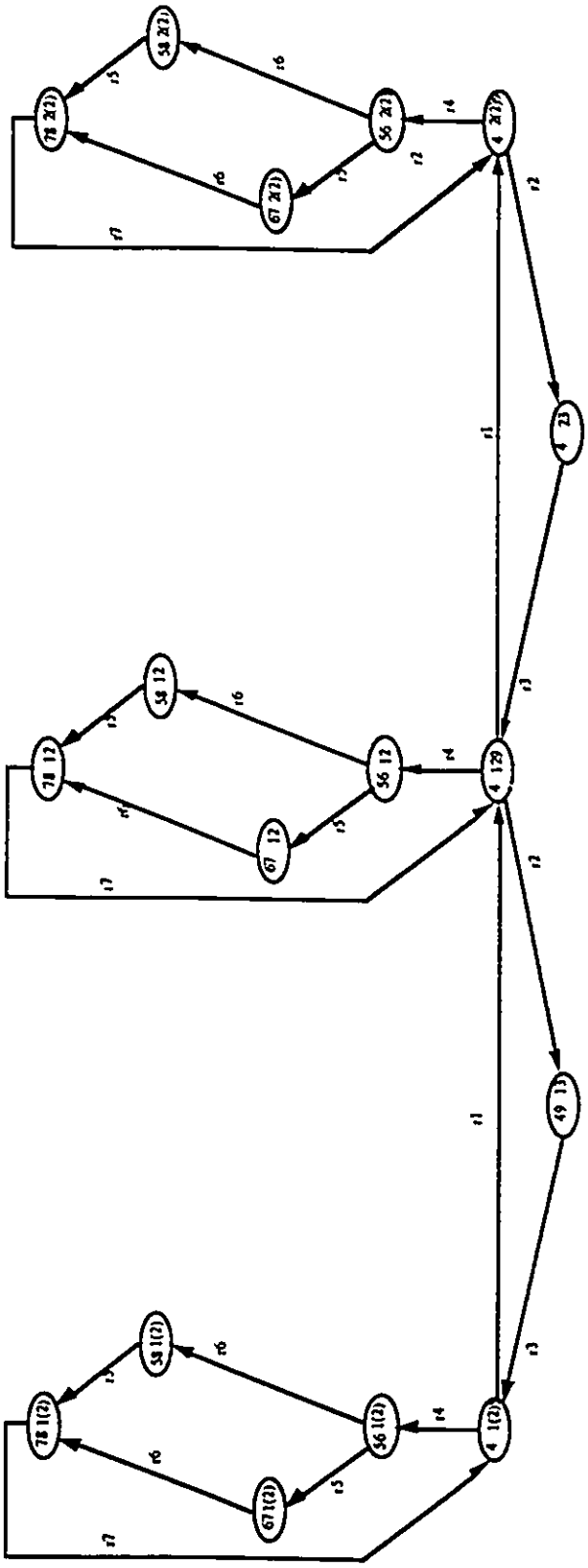

Figure 3.12: The Markov Chain of the Stochastic Petri Net

global balance equation of the system. Note that since the number of states is reduced, the probabilities need to be re-normalized.

Furthermore, suppose that the probabilities of the Basic Subnet Markings satisfy the Single Token Markov Chain balance equations and the sum of the Single Token equations is equal to the subnet global balance equation. Following a similar reasoning as above, if the changes to a Markov Chain are nothing but removal of some Single Token Markov Chains from the original Markov Chain, i.e. there is no "half" or "part" of a Single Token Markov Chain left, our projection is that the product form is still valid. In other words, in the Markov Chain of a Stochastic Petri Net, if one transition t_L from a subnet C_i is present, the Single Token Markov chain of C_i, where t_L is one of the transitions, is also present, the product form solutions still hold.

Our general projection is that suppose a Stochastic Petri Net consists of L subnets, and

1. the probabilities of the Basic Subnet Markings satisfy the Single Token Markov Chain balance equations and the sum of the Single Token equations is equal to the subnet global balance equation,
2. the Markov Chain of the system is such that if one transition t_L from a subnet C_i is present, the whole Single Token Markov chain of C_i, where t_L is one of the transitions, is also present,

then the product form solutions still hold. In Section 3.4, we will formally prove it. In the next Section, we will show how to judge if the Markov Chain of a Stochastic Petri Net has the above properties.

3.3 Identifying Local Balance Stochastic Petri Nets

In the procedure of testing if a Stochastic Petri Net is a Local Balance Stochastic Petri Net, we will make use of the concept of *Separate Subnet Marking*. For a given marking μ_i, there is a *Separate Subnet j Marking*, written as μ_i^j, which is a marking

consisting of the places of only subnet j and the number of tokens in the places is equal to that of μ_i. When there is no token in subnet j, $\mu_i^j = 0$. For example, the corresponding subnet markings of $\mu_i = (39)$ in Figure 3.1 are $\mu_i^1 = (9)$, $\mu_i^2 = (3)$, $\mu_i^3 = (0)$. Note that μ_i^j and $\hat{\mu}_l^j$ may or may not be the same. For example, (3) and (9) are the Basic Subnet Markings of net one and two. But (0) is not a Basic Subnet Marking, because $\hat{\mu}_l^j$ may never be zero.

The test procedure consists of three processes:

Process I

If there is any unsafe subnet I, the transitions have to be in sequence and among the input arcs of a transition, only one of them is from a non-resource place. The same must be true for the output arcs.

Process II

If there is any unsafe subnet II, they have to be state machines.

Process III

Divide the subnets into *Communicating Subnet Sets*(CSS). A subnet is in a CSS, if, and only if, it has at least a common resource place with one of the subnets in the CSS. Since an unsafe subnet II has no common resource places, it is a CSS by itself.

Choose a CSS that consists of more than one subnet. Put the transitions not in the CSS into the dormant state. With the initial marking, build the Reachability Graph of the CSS. For each marking μ reached, suppose that $t_L \in C_i$ is enabled,

1. if C_i is a safe subnet, check that
   - starting by firing t_L, μ may be reached again by firing only transitions in C_i. There may be many such cycles.

- there is a marking reached by some of the cycles whose Separate Subnet i Marking is $\hat{\mu}_0^i$.

2. if C_i is an unsafe subnet I, choose one of the non-resource tokens enabling t_L, and put other non-resource tokens into the dormant state. Check that starting by firing t_L, μ may be reached again by firing only transitions in C_i.

If it is the case, we say that the marking has passed the test. If every marking in the CSS passes the test, take another CSS that consists of more than one subnets and repeat the above test. If all the CSSs that consist of more than one subnets pass the test, the Stochastic Petri Net is a Local Balance Stochastic Petri Net.

Note that in the procedure above, we put no limit on the number of resource tokens in a resource place other than that the number has to be finite.

We have developed a software package in C language to test if a given safe Stochastic Petri Net is a Local Balance Stochastic Petri Net. The program has been verified with many examples of Stochastic Petri Nets. The list of the program is contained in Appendix A.

Let us pause for a while to see that given that a Stochastic Petri Net that has passed the test, what properties its subnets have and how the test procedure ensures that its Markov Chain has the properties listed at the end of Section 3.2.

3.3.1 Properties of Unsafe Subnets

Let *nsp* denote the set of non-resource places and remember that the transitions in an Unsafe Subnet I are in sequence.

Lemma 1 *Divide the non-resource places of an Unsafe Subnet I into two parts according to a resource place p_r. All the places after a transition that has p_r as one of its input places and before the next transition that has p_r as one of its output places belong to set α_1 and the rest of the non-resource places belong to set α_2. The number of tokens in the*

subnet satisfies the following constraints:

$$\sum_{i \in nsp} m_i = \sum_{i \in nsp} m_{0i} \tag{3.2}$$

$$m_r + \sum_{i \in o_1} m_i = m_{0r} \tag{3.3}$$

Recall that m_i denotes the number of tokens in place i and m_{0i} denotes the initial number of tokens in place i.

Proof:

According to the test process I, every transition has only one input/output arc from non-resource places. As a result, there is no join or fork of non-resource tokens in the subnet. Hence, the number of non-resource tokens should be constant. Therefore, Equation 3.2 holds. In addition, every token in a place belonging to set α_1 consumes a token in p_r. As a result, Equation 3.3 holds.

Q.E.D

Take the subnet two(the lower subnet) in Figure 3.7 as an example. It is an Unsafe Subnet I. Take any marking of its Reachability Graph(or Markov Chain) in Figure 3.11, we have $m_1 + m_2 + m_3 = m_{01} = 2$. This verifies Equation 3.2. Divide places one to three according to the resource place p_9. Transition t_1 has p_9 as both its input and output places. Hence there is no place in between the input and output arcs. Transition t_2 has p_9 as its input place and the next transition t_3 has p_9 as its output place. p_3 is in between t_2 and t_3. Therefore, $\alpha_1 = \{p_3\}$. Take any marking in Figure 3.11, we have $m_3 + m_9 = 1$. This verifies Equation 3.3.

If a transition has a resource place as one of its input places, we say that the firing of that transition *absorbs* a token from that place. If a transition has a resource place as one of its output places, we say that the firing of that transition *releases* a token into that place.

Lemma 2 *The following constraint holds in an unsafe subnet II:*

$$\sum_i m_i = \sum_i m_{0i} \tag{3.4}$$

Proof:

This follows directly from the fact that the subnet is a state machine.

Q.E.D.

Lemma 3 *For Unsafe Subnets I and II, we have the following results:*

$$P(\mu_i) = \frac{1}{G(K)} \prod_{i \in nsp} P_i^{m_i} \tag{3.5}$$

where $K = \sum_{i \in nsp} m_{oi}$, $G(K)$ *is a normalization constant:*

$$G(K) = \sum_{\mu_j} \prod_{i \in nsp} P_i^{m_i} \tag{3.6}$$

and P_i *is the probability that, when there is only one non-resource token in the subnet, it is in place* p_i.

Proof:

Equations 3.2 to 3.4 state that for an unsafe subnet, the number of tokens in a resource place may be determined by the number of tokens in the non-resource places. Therefore, when there is only one non-resource token in the subnet, the Markov Chain state may be defined as the disposition of that single token. Let α_i be the set of transitions enabled when the token is in p_i and η_i be the set of transitions the firing of which moves the token into p_i. Then P_i is obtained by solving the following equations:

$$P_i \times \sum_{s \in \alpha_i} r_s = \sum_{s \in \eta_i} P_j \times r_s \tag{3.7}$$

subject to

$$\sum_{i \in nsp} P_i = 1$$

For Unsafe Subnet I, since transitions are in sequence, α_i and η_i both contain a single transition, say for example t_x and t_y respectively. Equation 3.7 simplifies to:

$$P_i \times r_x = P_j \times r_y \tag{3.8}$$

When there are K non-resource tokens in the subnets, let γ_i be the set of transitions enabled in marking μ_i:

$$\gamma_i = \{j : D^-(.,j) \leq \mu_i\} \tag{3.9}$$

Let β_i be the set of transitions the firing of which leads to μ_i:

$$\beta_i = \{j : (\mu_i - D(.,j)) \in R(C, \mu_0), \mu_i \geq D^+(.,j)\} \tag{3.10}$$

The global balance equation for a marking μ_i is:

$$P(\mu_i) \times \sum_{j \in \gamma_i} r_j = \sum_{j \in \beta_i} P(\mu_i - D(.,j)) \times r_j \tag{3.11}$$

For an Unsafe Subnet I, according to the test process III, if t_x is enabled in μ_i, t_y must be enabled in $(\mu_i - D(.,y))$. As a result, Equation 3.11 holds if the following equations hold for all non-source places where $m_j(\mu_i) > 0$:

$$P(\mu_i) \times t_x = P(\mu_i - D(.,y)) \times t_y \quad t_x \in \alpha_j, t_y \in \eta_j \tag{3.12}$$

Insert Equation 3.5 into the above equation, we have:

$$\frac{1}{G(K)} \prod_{l \in nsp} P_l^{m_l} \times r_x = \frac{1}{G(K)} \prod_{l \neq i,j \; l \in nsp} P_l^{m_l} \times P_i^{m_i - 1} \times P_j^{m_j + 1} \times r_y \tag{3.13}$$

which simplifies to Equation 3.8.

For an Unsafe Subnet II, Equation 3.11 holds if the following equations hold for all non-resource places where $m_j(\mu_i) > 0$:

$$P(\mu_i) \times \sum_{s \in \alpha_j} r_s = \sum_{s \in \eta_j} P(\mu_i - D(.,j)) \times r_s \tag{3.14}$$

Insert Equation 3.5 into the above equation, we have:

$$\begin{aligned} & \frac{1}{G(K)} \prod_{l \in nsp} P_l^{m_l} \times \sum_{s \in \alpha_i} r_s \\ = & \sum_{s \in \eta_i} \frac{1}{G(K)} \prod_{l \neq i,j \; l \in nsp} P_l^{m_l} \times P_i^{m_i - 1} \times P_j^{m_j + 1} \times r_s \end{aligned} \tag{3.15}$$

which simplifies to Equation 3.7.

Q.E.D.

The fact that both Equations 3.12 and 3.14 hold indicates that the probability of a Basic Subnet Marking satisfy the Single Token Markov Chain balance equations. In addition, the fact that the sum of 3.12 for Unsafe Subnet I where $m_j(\mu_i) > 0$ is equal to 3.11 and the sum of 3.14 for Unsafe Subnet II where $m_j(\mu_i) > 0$ is equal to 3.11 state that the sum of the Single Token balance equations is equal to the subnet global balance equation. Therefore, the first condition listed at the end of section 3.2 is satisfied, namely, the probabilities of the Basic Subnet Markings satisfy the Single Token Markov Chain balance equations and the sum of the Single Token balance equations is equal to the subnet global balance equation.

Once a transition t_L in subnet C_i fires, the only cause of the absence of the Single Token Markov Chain, where t_L is one of the transitions, is that the resource tokens required for the enabling of some transitions are not present. As a result, once a transition from C_i fires, the state of subnets in other CSSs will not affect the presence or absence of the Single Token Markov Chain of C_i, where t_L is one of the transitions, because they do not share common resource places with C_i. Only the subnets in the same CSS with C_i may affect the presence or absence of the single token Markov Chain of C_i. In addition, if a CSS consists of a single subnet, it is a safe subnet or an Unsafe Subnet I or II. In any of these cases, once a transition in the subnet is enabled, the Single Token Markov Chain will certainly show up. This is why Process III divides the subnets into CSSs and examines only the CSSs that consist of more than one subnets.

Process III.1 says that marking μ and marking μ' whose Separate Subnet i Marking is $\hat{\mu}_0^i$ are reachable from each other. From μ' where $\hat{\mu}_0^i$ is present, firing transitions in C_i is nothing but constructing the Single Token Markov Chain of subnet C_i,(or its Basic Markov Chain of C_i, since it is a Safe Subnet). Since μ and μ' are reachable from each other, the Basic Markov Chain of C_i will be present with μ being one of its state and t_L being one of its transitions.

Processes I and III.2 ensure that if a transition t_L from an Unsafe Subnet I fires, the Single Token Markov Chain of that subnet, where t_L is one of the transitions, is also present.

Therefore, the second condition listed in the end of Section 3.2 is also satisfied, namely, the Markov Chain of the system is such that if one transition t_L from a subnet C_i is present, the whole Single Token Markov chain of C_i, where t_L is one of the transitions, is also present.

As an illustration of the test procedure, we take Figure 3.1 as an example. Since subnets one and two both have p_7 as their common resource place and subnet three does not have any common resource places with either subnet one or two, subnets one and two are in one CSS and subnet three alone is in another CSS. Suppose that we take the CSS consisting of subnets one and two. Both subnets are safe. Put transitions five to eight into the dormant state. In the initial marking $\mu_0 = (1\,7\,8)$, $\hat{\mu}_0^1 = (7\,8)$, $\hat{\mu}_0^2 = (1\,7)$ and t_1 in subnet two is enabled. Firing $t_1 t_2 t_3 t_4$, all of them belonging to subnet two, $(1\,7\,8)$ is reached again and $\hat{\mu}_0^2 = \mu_0^2 = (1\,7)$ is indeed the Separate Subnet Two Marking of marking $(1\,7\,8)$ in the cycle. Firing t_1, $\mu_1 = (2\,7\,8)$ is reached. t_9 in subnet one is also enabled in μ_0. Firing $t_9 t_{10} t_{11}$, all of them belonging to subnet one, $(1\,7\,8)$ is reached again. In addition, $\hat{\mu}_0^1 = \mu_0^1 = (7\,8)$ is the Separate Subnet One Marking of $(1\,7\,8)$ in the cycle. Firing t_9, $\mu_2 = (1\,7\,9)$ is reached. Hence, μ_0 passes the test. In $\mu_1 = (2\,7\,8)$, t_9 in subnet one is enabled. Firing $t_9 t_{10} t_{11}$, all of them belonging to subnet one, $(2\,7\,8)$ is reached again and $\hat{\mu}_0^1 = \mu_1^1 = (7\,8)$ is the Separate Subnet One Marking of $(2\,7\,8)$ in the cycle. Continuing the test we find that the CSS passes the test. Since the other CSS consists of only one subnet, no test is necessary. Therefore, the Stochastic Petri Net in Figure 3.1 has passed the test.

We have presented a method to test if a k-bounded Stochastic Petri Net is a Local Balance Stochastic Petri Net. In the following two sections, we will prove that the Stochastic Petri Nets that have passed the test procedure have product form solutions.

3.4 Mapping Separate Subnet Markings into Basic Subnet Markings

Unless otherwise stated, all the Stochastic Petri Nets considered in the rest of this Chapter have passed the test procedure.

In Section 3.2, we discussed how an associated Markov Chain will change, if previously independent subnets share common resources. Particularly, the presentation of a marking is different. Now we have a completely reverse problem. When some subnets share common resources, given a marking μ_i, how should we pick the $\hat{\mu}_l^j$s so that the product of their probabilities is proportional to the probability of μ_i?

Let us look at another example which is a modified "philosopher dining" problem. Five philosophers, who alternately think and eat, sit around a table. There is a fork between every two philosophers. To start eating, a philosopher needs two forks, and he is only allowed to use the two forks nearest to him. After a while, he gives up one fork and continue to eat with the other one. However, he needs two forks to finish eating. Then he goes back to thinking state. The Stochastic Petri Net is shown in Figure 3.13. The meanings associated with places and transitions are shown in Table 3.1, where $i = 1, 2, ..., 5$. We use Ph_i to denote the ith philosopher. The F_is are resource places representing where the forks are. According to the test procedure, it is a Local Balance Stochastic Petri Net. A single subnet i presenting the behavior of Ph_i is shown in Figure 3.14. Its associated Markov Chain is shown in Figure 3.15, where Ω stands for modulo.

Since it is a Local Balance Stochastic Petri Net, the marking probability is proportional to the product of the probabilities of its corresponding Basic Subnet Markings. Therefore, given a marking, what we need is to decide its corresponding Basic Subnet Markings. In other words, we need to decide what each philosopher is doing in that state. For example, suppose that $\mu_i = (T_1\ F_1\ E_2\ S_3\ T_4\ T_5\ F_5)$, where the presence of the name of a place indicates that there is a token in that place. The Separate Subnet Markings

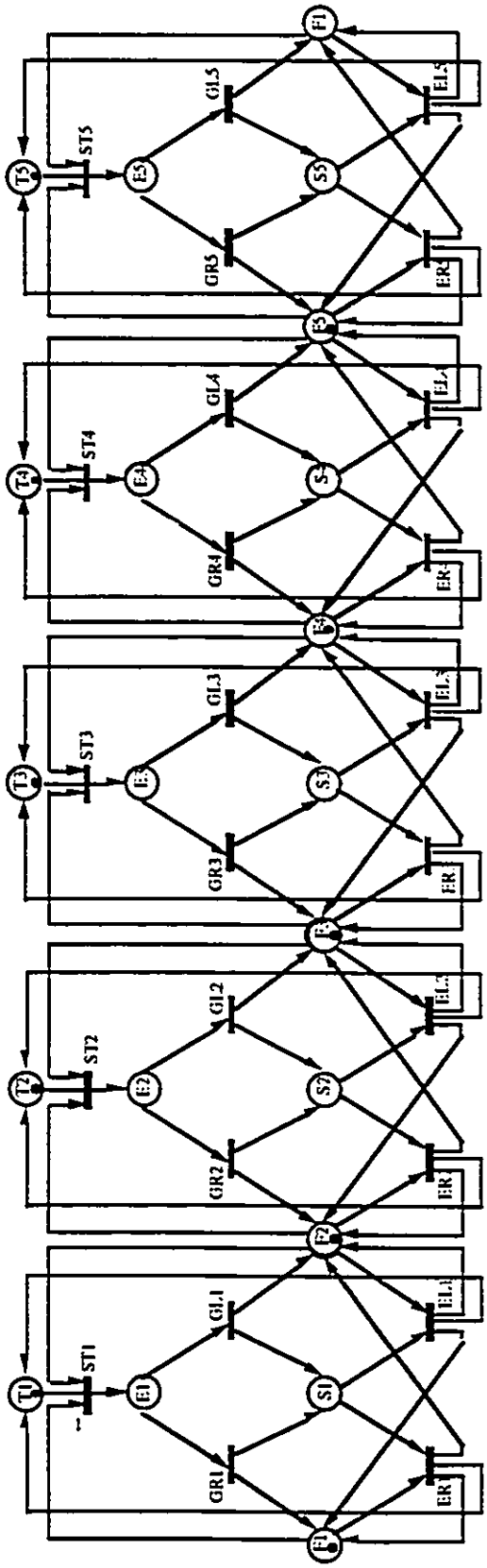

Figure 3.13: Stochastic Petri Net of the Philosopher Dining Problem

Places	
T i	Phi is thinking.
E i	Phi is eating with two forks.
S i	Phi is eating with one fork.
F i	Fork i is available.
Transitions	
ST i	End of thinking period for Phi.
GR i	Phi gives up the fork on his right.
GL i	Phi gives up the fork on his left.
ER i	Phi ends eating by picking up the fork on his right.
EL i	Phi ends eating by picking up the fork on his left.

Table 3.1: Meanings of Transitions and Places for the Dining Problem

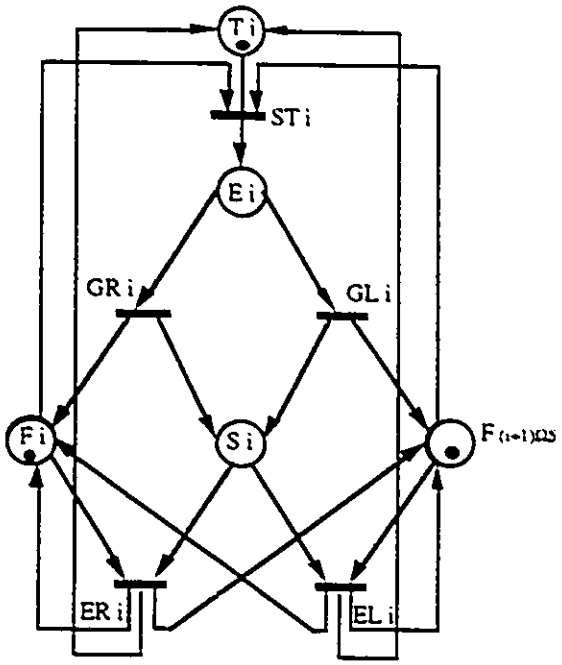

Figure 3.14: Subnet for a Single Philosopher

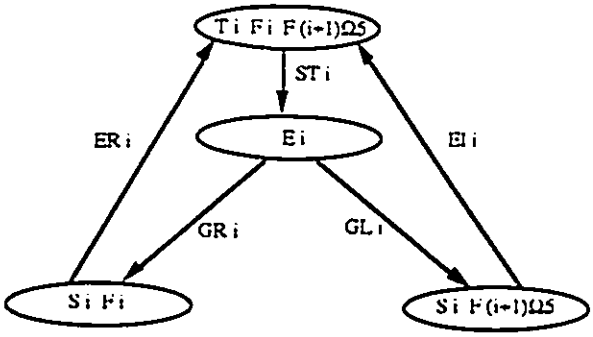

Figure 3.15: Markov Chain of the Stochastic Petri Net of a Single Philosopher

for each subnet are:

$$\begin{aligned}
\mu_i^1 &= (T_1\ F_1) \\
\mu_i^2 &= (E_2) \\
\mu_i^3 &= (S_3) \\
\mu_i^4 &= (T_4\ F_5) \\
\mu_i^5 &= (T_5\ F_1\ F_5)
\end{aligned}$$

Markings μ_i^2 and μ_i^5 are themselves Basic Subnet Markings indicating that Ph_2 is eating with F_1 and F_2 and Ph_5 is thinking.

Markings μ_i^1 and μ_i^4 are not Basic Subnet Markings. However, from Figure 3.15, it may be seen that when there is a token in T_i, there must be a token in F_i and $F_{(i+1)\%5}$ respectively. Therefore, the Basic Subnet Markings for subnets one and four are $(T_1\ F_1\ F_2)$ and $(T_4\ F_4\ F_5)$ respectively.

Marking μ_i^3 is not a Basic Subnet Marking either. However, given a token in S_3, there are two possible states, namely $(S_3\ F_3)$ or $(S_3\ F_4)$, where Ph_3 may be in. In other words, he may be eating with either the fork on his left or on his right. In this case, we need the information from other subnets to decide which state Ph_3 is in.

Since subnet two is in (E_2), i.e. Ph_2 is eating with both F_2 and F_3, Ph_3 may not use F_3. Therefore, $(S_3\ F_4)$ is not the right choice. Suppose that Ph_3 is using F_4, i.e. in state $(S_3\ F_3)$. Since Ph_4 is in a thinking state, no contradiction occurs. Therefore, the Basic Subnet Marking for subnet three is $(S_3\ F_3)$. The product form is, therefore,

$$\begin{aligned}
P(T_1\ E_2\ S_3\ T_4\ T_5\ F_1\ F_5) &= \frac{1}{G} \times P_1(T_1\ F_1\ F_2) \times P_2(E_2) \times P_3(S_3\ F_3) \times \\
&\quad P_4(T_4\ F_4\ F_5) \times P_5(T_5\ F_1\ F_5)
\end{aligned}$$

In summary, when given a Separate Subnet Marking, if there is more than one choice for its corresponding Basic Subnet Marking, we have to look at the state of other subnets in order to decide what state that particular subnet is in. The general rule is that *in a resource place, the number of resource tokens absorbed by subnets plus the resource tokens left should be equal to the total number of the initial tokens in that place.*

We will use an underline below a marking to indicate the same marking with the number of tokens in the common places being set to zero. For a Basic Markov Chain of subnet j, if $\exists \underline{\hat{\mu}_l^j} = \underline{\hat{\mu}_i^j}$, $l \neq i$ we say that $\underline{\hat{\mu}_i^j}$ is not *unique*, i.e. there are at least two markings, where the distribution of tokens in the non-resource places is the same, the difference is only in the token distributions in the common places. Take markings $(S_3\ F_3)$ and $(S_3\ F_4)$ for example. $\underline{(S_3\ F_3)} = \underline{(S_3\ F_4)} = (S_3)$. Therefore, these two markings are not unique.

In order to present the product form in a concise way, we define a function ρ mapping the *Separate Subnet Markings of marking* μ_i into the *Basic Subnet Markings* as follows(recall from Section 3.1 that there may be more than one common resource places for a pair of subnets. However, there is at most one common non-resource place for a pair of subnets.):

1. If subnet j is safe, then:
   - if $\underline{\mu_i^j} = 0$, then $\rho(\mu_i^j) = \hat{\mu}_0^j$;
   - if $\exists \underline{\hat{\mu}_l^j} = \underline{\mu_i^j}$ and $\hat{\mu}_l^j$ is unique, then $\rho(\mu_i^j) = \hat{\mu}_l^j$;
   - if $\exists \underline{\hat{\mu}_l^j} = \underline{\mu_i^j}$ and $\hat{\mu}_l^j$ is not unique, $\rho(\mu_i^j)$ is defined in such a way that:

$$\underline{\rho(\mu_i^j)} = \underline{\hat{\mu}_l^j} \tag{3.16}$$

     The number of tokens in a common place, e.g. p_x, of $\rho(\mu_i^j)$ is $m_{0x} - I_x^j$ so that
     * $\rho(\mu_i^j)$ is one of the Basic Subnet Markings, and
     * I_x^j satisfies:

$$m_x + \sum_{j=1}^{N} I_x^j = m_{0x} \tag{3.17}$$

     Where I_x^j is an integer representing the number of tokens in p_x that have been absorbed by subnet j.

2. If subnet j is an Unsafe Subnet I, then

$$\underline{\rho(\mu_i^j)} = \underline{\hat{\mu}_l^j} \tag{3.18}$$

The number of tokens in the common non-resource place may be determined by Equation 3.2 and the number of tokens in the common resource places, if any, may be obtained from Equation 3.3.

3. If subnet j is an Unsafe Subnet II, then

$$\underline{\rho(\mu_i^j)} = \underline{\hat{\mu}_i^j} \tag{3.19}$$

The number of tokens in the only common non-resource place is determined by Equation 3.4.

3.5 Product Form Solutions, A Formal Proof

In this Section, we prove that the Stochastic Petri Nets that have passed the test procedure have product form solutions. First, we have the following Lemma:

Lemma 4 *For a Local Balance Stochastic Petri Net, suppose that from marking* $(\mu_i - D^-(.,j))$, *transition* t_j *in subnet* C_l *fires and marking* μ_i *is reached. We have* $\rho(\mu_i - D(.,j))^k = \rho(\mu_i^k)$ *for* $k = 1, ..., N,\ k \neq l$.

Proof:

Without loss of generality, suppose that $l = N$. Because of the way the subnets are defined, the firing of transitions in subnet N will change the token distributions in places belonging to subnet N and may also change the number of tokens in common places.

For Unsafe Subnets, according to Equations 3.2 to 3.4, the number of tokens in the common places is completely determined by the token distributions in the non-resource places. As a result, the change of tokens in common places will not affect the mapping function, i.e. $\rho(\mu_i - D(.,j))^k = \rho(\mu_i^k)$ for Unsafe Subnets.

For Safe Subnets,

- $\exists \underline{\hat{\mu}_l^k} = \underline{(\mu_i - D(.,j))^k}$ and $\underline{\hat{\mu}_l^k}$ is unique.

The changes of tokens in common places do not affect the mapping. Therefore, $\rho(\mu_i - D(.,j))^k = \rho(\mu_i^k) = \hat{\mu}_l^k$ for $k = 1, ..., N-1$.

- $\exists \underline{\hat{\mu}_i^k} = \underline{(\mu_i - D(.,j))^k}$ and $\underline{\hat{\mu}_i^k}$ is not unique.

  Suppose that p_x is a common place of subnet N and j, $m_x(\rho(\mu_i - D(.,j))^N) = m_{0x} - h_1^N$, i.e. $I_x^N = h_1^N$. According to Equation 3.17, we have:

$$m_x(\mu_i - D(.,j))^N + \sum_{j=1}^{N-1} I_x^j + h_1^N = m_{0x} \tag{3.20}$$

  Now, suppose that the firing of transition t_L in subnet N changes $(\mu_i - D(.,j))$ into μ_i. Naturally, the firing of t_L changes $\rho((\mu_i - D(.,j))^N)$ into $\rho(\mu_i^N)$. Suppose that

$$m_x(\mu_i^N) = m_x((\mu_i - D(.,j))^N) - h_2^N \tag{3.21}$$

  where h_2^N is an integer.

$$h_2^N > 0, \text{ tokens are absorbed by subnet } N$$
$$h_2^N < 0, \text{ tokens are released by subnet } N$$

  Hence,

$$m_x(\rho(\mu_i^N)) = m_{0x} - h_1^N - h_2^N \tag{3.22}$$

  with $I_x^N = h_1^N + h_2^N$.

$$\begin{aligned} & m_x(\mu_i^N) + \sum_{j=1}^{N-1} I_x^j + I_x^N \\ = \; & m_x((\mu_i - D(.,j))^N) - h_2^N + \sum_{j=1}^{N-1} I_x^j + h_1^N + h_2^N \\ = \; & m_x((\mu_i - D(.,j))^N) + \sum_{j=1}^{N-1} I_x^j + h_1^N \\ = \; & m_{0x} \end{aligned} \tag{3.23}$$

  Therefore, if I_x^j, $j = 1, ..., N-1$ keep the same,

  - the $\rho(\mu_i^k)$s are still Basic Subnet Markings, and
  - Equation 3.23 indicates that Equation 3.17 is satisfied.

If a subnet has more than one common place with subnet N, each common place may be treated as described above. Therefore, we have $\rho((\mu_i - D(.,j))^k) = \rho(\mu_i^k)$, $k = 1, ..., N-1$.

Q. E. D.

Theorem 3 *A Stochastic Petri Net that has passed the test procedure has product form solutions. Furthermore, if it has N subnets, then:*

$$P(\mu_i) = \frac{1}{G} \prod_{j=1}^{N} P_j(\rho(\mu_i^j)) \tag{3.24}$$

where G is a normalization constant:

$$G = \sum_{\mu_i} \prod_{j=1}^{N} P_j(\rho(\mu_i^j)) \tag{3.25}$$

ρ is the mapping function defined in Section 3.4 and $P_j(\rho(\mu_i^j))$ is the probability obtained from the basic Markov Chain of subnet j.

Proof:

Define γ_i as the set of transitions enabled in μ_i:

$$\gamma_i = \{j : D^-(.,j) \leq \mu_i\} \tag{3.26}$$

Define β_i as the set of transitions the firing of which leads to μ_i:

$$\beta_i = \{j : (\mu_i - D(.,j)) \in R(C, \mu_0), \mu_i \geq D^+(.,j)\} \tag{3.27}$$

The global balance equation for state μ_i is:

$$P(\mu_i) \sum_{j \in \gamma_i} r_j = \sum_{j \in \beta_i} P(\mu_i - D(.,j)) \times r_j \tag{3.28}$$

Next, define γ_{il} as the set of transitions in subnet l enabled in μ_i:

$$\gamma_{il} = \{j : D^-(.,j) \leq \mu_i,\ t_j \in C_l\} \tag{3.29}$$

Also define β_{il} as the set of transitions in subnet l, the firing of which leads to μ_i:

$$\beta_{il} = \{j : (\mu_i - D(.,j)) \in R(C, \mu_0), \mu_i \geq D^+(.,j),\ t_j \in C_l\} \tag{3.30}$$

The global balance equation holds if the following local balance equations hold:

$$P(\mu_i) \sum_{j \in \gamma_{il}} r_j = \sum_{j \in \beta_{il}} P(\mu_i - D(.,j)) \times r_j \quad \forall l = 1, ..., N. \tag{3.31}$$

Without the loss of generality, we show that the above equation holds when $l = N$, i.e.

$$P(\mu_i) \sum_{j \in \gamma_{iN}} r_j = \sum_{j \in \beta_{iN}} P(\mu_i - D(.,j)) \times r_j \tag{3.32}$$

Inserting Equation 3.24 into Equation 3.32, we have:

$$\begin{aligned} & \frac{1}{G} \prod_{k=1}^{N-1} P_k(\rho(\mu_i^k)) \times P_N(\rho(\mu_i^N)) \times \sum_{j \in \gamma_{iN}} r_j \\ = & \frac{1}{G} \sum_{j \in \beta_{iN}} \prod_{j=k}^{N-1} P_k(\rho(\mu_i - D(.,j))^k) \times P_N(\rho(\mu_i - D(.,j))^N) \times r_j \end{aligned} \tag{3.33}$$

According to Lemma 4, $\rho(\mu_i - D(.,j))^k = \rho(\mu_i^k)$ for $k = 1, ..., N-1$. Hence, Equation 3.33 simplifies to:

$$P_N(\rho(\mu_i^N)) \times \sum_{j \in \gamma_{iN}} r_j = \sum_{j \in \beta_{iN}} P_N(\rho((\mu_i - D(.,j))^N)) \times r_j \tag{3.34}$$

If subnet N is safe, Equation 3.34 is the global balance equation of subnet N; if it is an Unsafe subnet I or II, Equation 3.34 may be further reduced to Equation 3.7.

Q. E. D.

We proved that a Stochastic Petri Net that has passed the test procedure has product form solutions. Let us examine Equation 3.24 carefully. Since the associated Markov Chain of a Local Balance Stochastic Petri Net is finite, Equation 3.24 looks very similar to the product form solutions for closed BCMP queueing networks. However, a major difference lies in the fact that the total population in a closed BCMP queueing network is constant whereas there is no such constraint for a Local Balance Stochastic Petri Net in general, except for Unsafe Subnets. This difference prevents the application of computation algorithms available for closed BCMP networks to the analysis a Local Balance Stochastic Petri Nets. Also, the absence of constraints for Local Balance Stochastic Petri Nets complicates the analysis.

3.6 Comparisons with Lazar and Robertazzi's results

Lazar and Robertazzi proposed that there is a class of Petri Nets which has product form solutions. They stated in [41] that "A safe Markovian Petri Net consisting of a number of task sequences that are comprised of a series of sequential sub-tasks has a product form solution for the equilibrium state probabilities if the state transition lattice can be naturally associated with a Cartesian coordinate system (alternatively: naturally embedded into an aggregation of toroidal manifolds) and if the state transition lattice is comprised of integral building blocks and corresponding consistent set of local balance equations".

At the Stochastic Petri Net structure level, they considered "A safe Markovian Petri Net consisting of a number of task sequences that are comprised of a series of sequential sub-tasks. The product form solution for the equilibrium state probabilities exists, if and only if, a task sequence is only allowed to proceed if there is a non-zero probability that it can return to its current state without the need for a state change in other task sequences".

The product in their notation is:

$$P(k_1, k_2, ..., k_N) = \prod_{l=1}^{N} \frac{q^{l0}}{q^{lk_l}} p(0, 0, ..., 0) \tag{3.35}$$

Where k_l is the k_lth sub-task of the lth task sequence and $P(k_1, k_2, ..., k_N)$ is their probability. q^{kl} is the firing rate associated with the lth sub-task of the kth task sequence and $p(0, 0, ..., 0)$ is a normalization constant.

The word "task" is not a common term in Stochastic Petri Nets. Further, it is not easy to verify if a given Stochastic Petri Net is a Local Balance Stochastic Petri Net by the above statements, especially when the Stochastic Petri Net is relatively large. In addition, There are two restrictions on the Stochastic Petri Nets, namely, "sequential sub-tasks" and "safe Stochastic Petri Nets".

In comparison, we presented a systematic way and a software package to test if

a Stochastic Petri Net is a Local Balance Stochastic Petri Net. This makes the job of identification of Local Balance Stochastic Petri Nets much easier. By doing so, we are able to get rid of the terms like "toroidal manifolds", "building blocks", "task", "pasting", etc. as were used by Lazar and Robertazzi.

More important, we divided a k-bounded Local Balance Stochastic Petri Net into subnets. The subnets may be one of the following cases:

1. A Safe Subnet, where the number of tokens in the non-resource places does not exceed one.(No restrictions on the number of tokens in resource places).
2. An Unsafe Subnet I with the transitions in sequence and among the input arcs of a transition, only one of them is from a non-resource place. The same must be true for output arcs.
3. An Unsafe subnet II which is a Finite Machine.

Therefore, a Local Balance Stochastic Petri Net does not have to be safe and the transitions in subnets("sub-task" in their notations) do not have to be in sequence(except for Unsafe Subnet I).

As an example, Figure 3.16 shows a Stochastic Petri Net. According to the test procedure, it is still a Local Balance Stochastic Petri Net. However, it is not a safe Stochastic Petri Net. In addition, subnet one consists of two cycles of transitions, namely $\{t_9t_{10}t_{11}t_{12}\}$ and $\{t_9t_{11}t_{10}t_{12}\}$. Subnet three also consists of two cycles of transitions, namely $\{t_5t_6t_7\}$ and $\{t_8t_7\}$.

By allowing more than one tokens in a place, jobs with the same statistical behavior and shared resources may be modelled by an Unsafe Subnet inside a Local Balance Stochastic Petri Net, e.g. the bottom subnet in Figure 3.16. By allowing non-sequential firing of transitions inside a subnet, the modelling of blocking and job splitting within a subnet by Local Balance Stochastic Petri Nets is made possible. For instance, in Figure 3.16, t_9 represents a job being split into two and t_{12} represents a blocking phenomenon. If there is one token in p_{10} and p_{11} respectively, t_{12} cannot fire. It has to wait for the firing of t_{10}. In other words, the event modelled by t_{12} is blocked.

On the other hand, the extension of boundary introduces the difficulty in the mapping of markings. Indeed, if a Stochastic Petri Net is safe and the subnet transitions are all in sequence, the Markov Chain states are uniquely determined by the disposition of the single token in each subnet. No mapping is necessary.

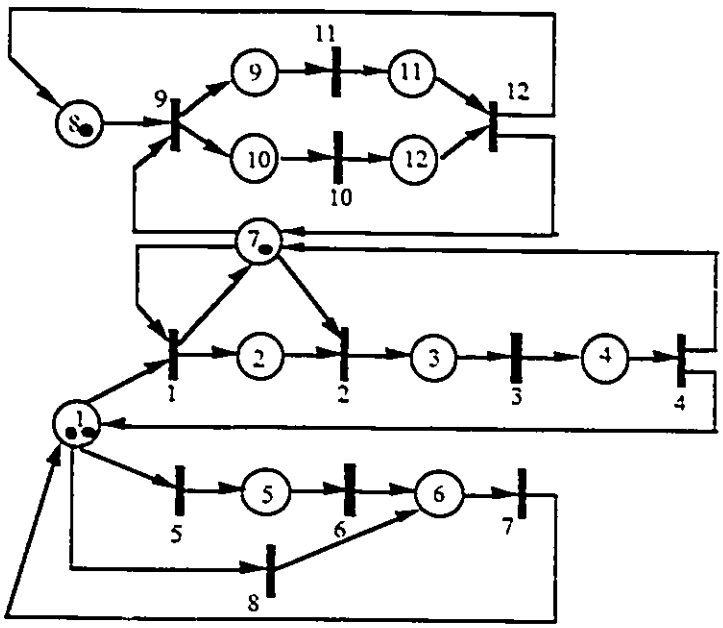

Figure 3.16: Local Balance Stochastic Petri Net with blocking

In Equation 3.35, the product form is a product of the sequential firing rates. It is a special case of Equation 3.24, which is a product of the Basic Subnet Marking probabilities. When a Local Balance Stochastic Petri Net is safe and each subnet consists of only one cycle of transitions, Equation 3.24 reduces to Equation 3.35.

In summary, our results differ from that of Lazar and Robertazzi's in the following ways:

1. The Local Balance Stochastic Petri Nets do not have to be safe(especially that the number of resource tokens in a resource place may be more than one) whereas "a safe Stochastic Petri Net" is required in [41].

2. The transitions in a Local Balance Stochastic Petri Net's subnets do not have

to be in sequence all the time(only required when the subnet is unsafe subnet I) whereas "sequential firing" is required in [41].

3. A systematic way for testing the Local Balance Stochastic Petri Nets is presented.

4. A mapping function is introduced.

5. The product form solution is a product of subnet marking probabilities whereas it was the product of the sequential firing rates in [41].

Since Lazar and Robertazzi first reported the product form solutions for a class of Stochastic Petri Nets, some people have informally questioned how practical this class of Stochastic Petri Nets is. Our opinions are the following:

1. Lazar and Robertazzi showed that multiprocessor systems, communication protocols such as CSMA, alternating bit as well as the philosopher dining problem may be modelled by Local Balance Stochastic Petri Nets. These certainly present the practical aspect of Local Balance Stochastic Petri Nets.

2. It is true that the cases of Local Balance Stochastic Petri Nets are limited. That only points out the need for more work to extend the boundary, i.e. to gradually make it more and more practical. We believe that this is the path of evolution for many theories. The development of the Petri Net theory from its very original proposal is itself a very good illustration.

3.7 Summary

We extended the boundary of Local Balance Stochastic Petri Nets by allowing non-safe Stochastic Petri Nets and non-sequential firing of transitions inside a subnet. We presented a systematic way as well as a software package for identifying the Local Balance Stochastic Petri Nets and proved that the Stochastic Petri Nets that have passed the test procedure have product form solutions. The results in this Chapter contribute in

extending the boundary of Local Balance Stochastic Petri Nets so that more systems may be analyzed efficiently and making the identification of Local Balance Stochastic Petri Nets much easier.

Chapter 4

Norton's Theorem for Local Balance Stochastic Petri Nets

From time to time, we would like to have a simple presentation of a system. A concise presentation enables better understanding of the system. In this Chapter, we show that for a Local Balance Stochastic Petri Net, we may use a subnet with marking dependent firing rates to represent the original Stochastic Petri Net. We call the procedure *the Norton's Theorem.*

We first apply the theory of Decomposition and Aggregation to the associated Markov Chains of Local Balance Stochastic Petri Nets and develop a *Decomposition by Subnet* method. Then we use the Decomposition by Subnet method to derive the Norton's theorem at the net level. Finally, we discuss the application of Norton's theorem to parametric analysis.

4.1 Decomposition by Subnet

Firstly, define:

- *Favorite Marking of* t_i. A marking μ is a favorite marking of transition t_i, if t_i is enabled in μ. The set of favorite markings of t_i is written as $fm(t_i)$.

- *Post-marking* and *Pre-marking* with respect to a transition. If in marking μ_i, transition t_k fires and μ_j is reached, we call μ_j the Post-marking of μ_i with respect to t_k, written as $\mu_j = ps(\mu_i|t_k)$ and μ_i the Pre-marking of μ_j, written as $\mu_i = pr(\mu_j|t_k)$.

Take Figure 3.1 as an example. The markings (38) and (39) are Favorite Markings of t_3. Hence, $fm(t_3) = \{(38)(39)\}$. In addition, $ps((38)|t_3) = (48)$, $pr((48)|t_3) = (38)$.

We choose to decompose the Markov Chain according to the states of a subnet. Without loss of generality, suppose that the subnet is subnet one. In other words, the markings are divided into classes. A marking μ_i is in class k, if and only if, $\rho(\mu_i^1) = \hat{\mu}_k^1$. As a result, the number of classes is equal to the number of $\hat{\mu}_k^1$. A marking inside class k is denoted as μ_{ik}. We use $n(k)$ to denote the number of markings in the kth class.

For example, in Figure 3.2, markings (6 10), (5 10), (1 10), (2 10) are in the same class, because their $\rho(\mu_i^1) = (10)$. In fact in Figure 3.2, all the markings on the same horizontal line are in the same class. We will refer the top markings as class one, the middle class two and the bottom class three.

A further examination of the connections of markings in the same class will be interesting. In figure 3.2, class one and three are horizontally connected. In other words, a marking inside a class may reach another marking in the same class without passing through markings in other classes. However, markings in class two are not all connected. Markings (6 10), (5 10) and (1 10) may reach each other without passing markings in other classes. But marking (2 10) has to pass makings in class one or class three in order to reach (6 10), (5 10) or (1 10). In general, the markings in the same class may be:

1. All connected, where the markings in a class may reach each other without passing any markings in other classes.
2. All not connected, where None of the markings in a class may reach each other without passing markings in other classes.
3. Some are connected and others are not, where some of the markings may reach each other without passing any markings in other classes. Others have to pass

markings in other classes in order to reach any of the markings in the same class.

According to Lemma 4, the firing of transitions in subnet one will change the token distributions inside subnet one and may also change the number of tokens in the common places. However, it will not alter $\rho(\mu_i^l)$ for $l = 2, \ldots, N$. Therefore, *the transitions among markings of the same class, if any, are the results of the firing of transitions in subnets other than subnet one. The transitions between markings of two different classes, if any, are the results of the firing of a transition in subnet one.*

According to Simon and Ando's technique step one, we need to obtain $\mathbf{v}_I^*$. Also, if exact results are expected from the Decomposition and Aggregation procedure, $\mathbf{v}_I^*$ has to be the conditional probabilities(Theorem 1). We use $v^*(\mu_{ik})$ to denote the conditional probability of marking μ_{ik} given that it is in class k.

$$v^*(\mu_{ik}) = \frac{P(\mu_{ik})}{\sum_{l=1}^{n(k)} P(\mu_{lk})} \tag{4.1}$$

One way to obtain the $v^*(\mu_{ik})$s would be to calculate the probabilities and insert them into the above Equation. However, we would later like to be able to calculate them at the net level. For that purpose, we distinguish three different cases to obtain the $v^*(\mu_{ik})$:

1. μ_{ik}s, $i = 1, \ldots, n(k)$ may reach each other without passing any markings in other classes. An example is the class one markings in Figure 3.2. Inserting Equation 3.24 into Equation 4.1, we have:

$$v^*(\mu_{ik}) = \frac{\frac{1}{G} P_1(\rho(\mu_{ik}^1)) \prod_{j=2}^{N} P_j(\rho(\mu_{ik}^j))}{\frac{1}{G} P_1(\rho(\mu_{ik}^1)) \sum_{l=1}^{n(k)} \prod_{j=2}^{N} P_j(\rho(\mu_{lk}^j))} \tag{4.2}$$

$$= \frac{\prod_{j=2}^{N} P_j(\rho(\mu_{ik}^j))}{\sum_{l=1}^{n(k)} \prod_{j=2}^{N} P_j(\rho(\mu_{lk}^j))} \tag{4.3}$$

   This is because the markings in the same class have the same $\rho(\mu_{ik}^1)$.

2. None of the μ_{ik}s may reach each other without passing markings in other classes. $v^*(\mu_{ik})$s may be obtained from the $v^*(\mu_{il})$s of class l that are already

known. Suppose that $\mu_{i\,k-1} = pr(\mu_{ik}|t_s)$, $t_s \in C_1$ and $v^*(\mu_{i\,k-1})$s are known. We have:

$$\frac{P(\mu_{i\,k-1})}{P(\mu_{ik})} = \frac{P_1(\rho(\mu^1_{i\,k-1}))}{P_1(\rho(\mu^1_{ik}))} = constant \quad \forall i \tag{4.4}$$

since according to Lemma 4, $\rho(\mu^j_{ik}) = \rho(\mu^j_{i\,k-1})$ for $j = 2, \ldots, N$. Also, the markings in the same class have the same $\rho(\mu^1_{ik})$. Hence,

$$\begin{aligned} v^*(\mu_{ik}) &= \frac{P(\mu_{ik})}{\sum_{l=1}^{n(k)} P(\mu_{lk})} = \frac{P(\mu_{i\,k-1})}{\sum_{\mu_{l\,k-1} \in fm(t_s)} P(\mu_{l\,k-1})} \\ &= \frac{v^*(\mu^1_{i\,k-1})}{\sum_{\mu_{l\,k-1} \in fm(t_s)} v^*(\mu^1_{l\,k-1})} \end{aligned} \tag{4.5}$$

Therefore, if $\frac{v^*(\mu^1_{i\,k-1})}{\sum_{\mu_{l\,k-1} \in fm(t_s)} v^*(\mu^1_{l\,k-1})}$ is known, $v^*(\mu_{ik})$ is also known.

3. Some of the μ_{ik}s, $i = 1, \ldots, n(k)$ may reach each other without passing any markings in other classes. Denote this set of markings as ξ_{1k}. Others have to pass markings in other classes in order to reach any of the μ_{ik}, denote this set of markings as ξ_{2k}. An example is the class two markings in Figure 3.2. Markings (6 10), (5 10) and (1 10) are in ξ_{12}. Marking (2 10) is in ξ_{22}.

$$v^*(\mu_{ik}) = \frac{P(\mu_{ik})}{\sum_{m \in \xi_{1k}} P(\mu_{mk}) + \sum_{l \in \xi_{2k}} P(\mu_{lk})} = \frac{\frac{P(\mu_{ik})}{\sum_{m \in \xi_{1k}} P(\mu_{mk})}}{1 + \sum_{l \in \xi_{2k}} \frac{P(\mu_{lk})}{\sum_{m \in \xi_{1k}} P(\mu_{mk})}} \tag{4.6}$$

Let $P^*(\mu_{ik}) = \frac{P(\mu_{ik})}{\sum_{\forall m \in \xi_{1k}} P(\mu_{mk})}$, $i \in \xi_{1k}$. According to the test procedure, if one transition t_L from subnet C_i appears, the Single Token Markov Chain of C_i, where t_L is one of the transitions, is also present. In addition, transitions between classes are transitions from subnet one only. As a result, the transitions inside the same class must be the Single Token Markov Chain of some subnets other than subnet one. Denote this set of subnets by γ. Therefore:

$$\begin{aligned} P^*(\mu_{ik}) &= \frac{\frac{1}{G} \prod_{j=1}^{N} P_j(\rho(\mu^j_{ik}))}{\frac{1}{G} \sum_{l \in \xi_{1k}} \prod_{j=1}^{N} P_j(\rho(\mu^j_{lk}))} \\ &= \frac{\prod_{j \in \gamma} P_j(\rho(\mu^j_{ik}))}{\sum_{l \in \xi_{1k}} \prod_{j \in \gamma} P_j(\rho(\mu^j_{lk}))} \end{aligned} \tag{4.7}$$

because, according to Lemma 4, the firing of transitions belonging to the subnets in γ will not alter the $\rho(\mu_{ik}^j)$ for $j \notin \gamma$.

Suppose $\mu_{i\ k-1} = pr(\mu_{ik}|l_s)$, $i \in \xi_{2k}$, then given $x \in \xi_{1k}$, following the same reasoning as that for Equation 4.4, we have

$$\frac{P(\mu_{ik})}{P(\mu_{i\ k-1})} = \frac{P(\mu_{xk})}{P(\mu_{x\ k-1})} \quad i \in \xi_{2k} \tag{4.8}$$

i.e.

$$P(\mu_{ik}) = \frac{P(\mu_{xk})}{P(\mu_{x\ k-1})} \times P(\mu_{i\ k-1}) \quad i \in \xi_{2k} \tag{4.9}$$

Therefore for $x \in \xi_{1k}$ and $i \in \xi_{2k}$

$$\begin{aligned} \frac{P(\mu_{ik})}{\sum_{m\in\xi_{1k}} P(\mu_{mk})} &= \frac{P(\mu_{xk})}{P(\mu_{x\ k-1})} \times \frac{P(\mu_{i\ k-1})}{\sum_{m\in\xi_{1k}} P(\mu_{mk})} \\ &= P^*(\mu_{xk}) \times \frac{P(\mu_{i\ k-1})}{P(\mu_{x\ k-1})} \\ &= P^*(\mu_{xk}) \times \frac{v^*(\mu_{i\ k-1})}{v^*(\mu_{x\ k-1})} \end{aligned} \tag{4.10}$$

where the first Equation is a result of replacing $P(\mu_{ik})$ with Equation 4.9. Therefore, Equation 4.6 becomes:

$$v^*(\mu_{ik}) = \begin{cases} \frac{P^*(\mu_{ik})}{1+\sum_{l\in\xi_{2k}} \frac{v^*(\mu_{l\ k-1})}{v^*(\mu_{x\ k-1})} \times P^*(\mu_{xk})} & i \in \xi_{1k},\ x \in \xi_{1k} \\ \frac{P^*(\mu_{xk}) \times \frac{v^*(\mu_{i\ k-1})}{v^*(\mu_{x\ k-1})}}{1+\sum_{l\in\xi_{2k}} \frac{v^*(\mu_{l\ k-1})}{v^*(\mu_{x\ k-1})} \times P^*(\mu_{xk})} & i \in \xi_{2k},\ x \in \xi_{1k} \end{cases} \tag{4.11}$$

where $v^*(\mu_{ik})$ is related to $P^*(\mu_{i\ k})$ and $v^*(\mu_{i\ k-1})$ from another class. If ξ_{2k} is empty, $v^*(\mu_{ik}) = P^*(\mu_{ik})$.

After $v^*(\mu_{ik})$s are obtained, according to the Decomposition and Aggregation procedure step two, we need to obtain transition rates between classes. Since the firing of transitions in subnet one will alter only $\rho(\mu_i^1)$ and the markings in the same class have the same $\rho(\mu_i^1)$, *the transition rates from markings of one class to markings of another class, if any, are the same.* For example, in Figure 3.2, the transitions from markings in class one to markings in class two equal to r_{10}, from class two to class three equal to

r_{11} Suppose that the firing of transitions $t_i \in C_1$ leads from class k markings to class l markings. Then according to Equation 1.1, the transition rate between them is:

$$\lambda_{kl} = r_i \times \sum_{\mu_{ik} \in fm(t_i)} v^*(\mu_{ik}) \tag{4.12}$$

We then analyze the reduced Markov Chain to obtain its steady state probabilities. Denote the probability of class k as P_k.

According to Decomposition and Aggregation technique step three, to obtain the results for the original system, we have:

$$P(\mu_{ik}) = v^*(\mu_{ik}) \times P_k \tag{4.13}$$

In this way, we reduce the analysis of the original system into the analysis of subsystems. In the next Section, we will translate the results from this Section into detail steps to perform at the net level.

4.2 Norton's Theorem for Local Balance Stochastic Petri Nets

In this section, we show that a Local Balance Stochastic Petri Net may be concisely represented by one of its subnet with marking dependent firing rates. Firstly, define:

- *Generating Markings of Subnet j.* Starting from μ_0, all the markings reached by firing the transitions only in subnet j are defined as Generating Markings of Subnet j, written as $\tilde{\mu}_k^j$.

For example, in Figure 3.1, the generating markings of subnet one are (178), (179), (1 10). Next, we decide which subnet will be used to replace the whole net. Without loss of generality, suppose that it is subnet one.

Draw subnet one separately. To distinguish it from the net one inside the original Stochastic Petri Net C, we call it the *aggregated net A*. Denote the markings of A by a_k. Since $\tilde{\mu}_k^1$s are obtained by firing only subnet one transitions, the Separate Subnet one

markings of $\tilde{\mu}_k^1$s are in fact the a_ks. Hence, there is a one to one correspondence between $\tilde{\mu}_k^1$s, a_ks and the classes. Suppose that the number of a_ks is M.

Again, we choose to decompose the Stochastic Petri Net by subnet one. To perform the Decomposition and Aggregation at the net level, we combine Simon and Ando's Steps one and two, i.e. we obtain the $v^*(\mu_{ik})$s and then immediately compute λ_{kl} according to Equation 4.12. Since there is a unique association of the classes with a_ks, we may assign the rates according to a_k and t_i:

$$r_i(a_k) = r_i \times \sum_{\mu_{ik} \in fm(t_i)} v^*(\mu_{ik}) \tag{4.14}$$

The firing of t_i will lead a_k to a_l. Equation 4.14 may be regarded as the firing rate of t_i when the aggregated net marking is a_k. In this way, the aggregated Markov Chain obtained is actually equivalent to the Markov Chain of subnet one with marking dependent firing rates determined by Equation 4.14.

From $\tilde{\mu}_0^1 = \mu_0$, generate $\tilde{\mu}_k^1$s(hence a_ks). For each $\tilde{\mu}_k^1$ generated, we will obtain $v^*(\mu_{ik})$s. Since all the markings in the same class have the same $\rho(\mu_i^1)$, we put the transitions in C_1 into the dormant state. Hence, $\rho(\mu_i^1)$ will not change. The transitions in other subnets may be in one of the following situations:

1. All the transitions are live.
2. All the transitions are not live.
3. The transitions in some subnets are live while others are not live.

For each situation, $v^*(\mu_{ik})$s are obtained by one of the three ways presented in the above Section. In the following, we will discuss them separately. Remember that the transitions between markings of two different classes, if any, are the results of the firing of one transition in subnet one. The transitions among markings of the same classes, if any, are the results of the firing of transitions in subnets other than subnet one.

Situation One

In this case, the markings may reach each other without passing any markings in other classes. Hence, $v^*(\mu_{ik})$s may be obtained by the first method described in the above section, i.e. by using Equation 4.3. For convenience, we write it again :

$$\frac{\prod_{j=2}^{N} P_j(\rho(\mu_{ik}^j))}{\sum_{l=1}^{n(k)} \prod_{j=2}^{N} P_j(\rho(\mu_{lk}^j))} \tag{4.15}$$

Note that the product form is the product of marking probabilities from subnet two to N. The sum is over all the markings such that $\rho(\mu_i^1) = a_k$. Therefore, $v^*(\mu_{ik})$ is in fact the probability of the original Stochastic Petri Net with initial marking $\tilde{\mu}_k^1$ and the transitions of subnet one being in the dormant state. Therefore, simply analyze the net with the transitions in C_1 being in the dormant state and obtain the $v^*(\mu_{ik})$s.

In a_k, if transitions $t_i \in C_1$ is enabled, the marking dependent firing rate $r_i(a_k)$ is obtained by Equation 4.14.

For example, in Figure 3.1, $\tilde{\mu}_0^1 = (178)$, $a_0 = (78)$. Putting transitions nine to eleven into the dormant state, all the transitions in net two and three are live. Hence, this is situation one. Analyze the Stochastic Petri Net to obtain $v^*(\mu_{i0})$. The Markov Chain of the net is shown in Figure 4.1. In a_0, $t_9 \in C_1$ is enabled and

$$fm(t_9) = \{(678), (578), (178), (278)\} \tag{4.16}$$

Therefore, Equation 4.14 gives:

$$r_9(78) = r_9 \times \{v^*(678) + v^*(578) + v^*(178) + v^*(278)\} \tag{4.17}$$

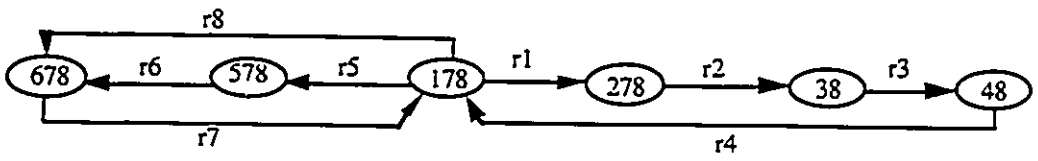

Figure 4.1: Markov Chain Associated with $a_0 = (78)$

Firing t_9, $\tilde{\mu}_1^1 = (179)$, $a_1 = (79)$ and putting transitions nine to eleven into the dormant state, all transitions in net two and three are live. Hence, this is, again, situation one. Analyze the Stochastic Petri Net to obtain $v^*(\mu_{i1})$s. Its Markov Chain is shown in Figure 4.2. In a_1, $t_{10} \in C_1$ is enabled and

$$fm(t_{10}) = \{(679), (579), (179), (279)\}$$

Hence

$$r_{10}(79) = r_{10} \times \{v^*(679) + v^*(579) + v^*(179) + v^*(279)\} \tag{4.18}$$

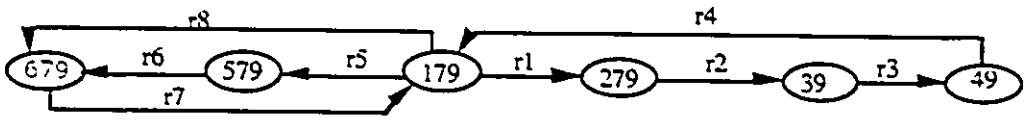

Figure 4.2: Markov Chain Associated with $a_1 = (79)$

Situation Two

In this case, none of the μ_{ik}s may reach each other without passing markings in other classes. Hence, $v^*(\mu_{ik})$s may be obtained by the second method described in the above section, i.e. by Equation 4.11.

In a_k, if transitions $t_i \in C_1$ is enabled, the marking dependent firing rate $r_i(a_k)$ is obtained through Equation 4.14.

Situation Three

In this case, some of the μ_{ik}s, $i = 1, ..., n(k)$ may reach each other without passing any markings in other classes. Others have to pass markings in other classes to reach another marking in class k. Hence, $v^*(\mu_{ik})$s may be obtained by the third method described in the above section, i.e. Equation 4.11. However, in order to use Equation 4.11, we need

to obtain $P^*(\mu_{ik})$ by equation 4.7. For convenience, we write it again:

$$P^*(\mu_{ik}) = \frac{\prod_{j\in\gamma} P_j(\rho(\mu_{ik}^j))}{\sum_{l\in\xi_{1k}} \prod_{j\in\gamma} P_j(\rho(\mu_{lk}^j))}$$

Recall that γ is the set of subnets whose transition firings connect some of the markings in the class. Also note that the product form is the product of marking probabilities of subnets that are in γ. The sum is over all the markings connected by transitions from subnets in γ and $\rho(\mu_i^1) = a_k$. Therefore, $P^*(\mu_{ik})$ is the probability of the original Stochastic Petri Net with initial marking $\tilde{\mu}_k^1$ and the transitions of subnet one being in the dormant state. Therefore, put the transitions in C_1 into the dormant state and analyze the net to obtain $P^*(\mu_{ik})$, $i \in \xi_{1k}$. In addition, we need to find the set of markings that are in ξ_{2k}. Suppose $\tilde{\mu}_k^1 = ps(\tilde{\mu}_{k-1}^1|t_s)$, then for each $\mu_{i\ k-1} \in fm(t_s)$, find out $\mu_{ik} = ps(\mu_{i\ k-1}|t_s)$. Some of these Post markings are already in ξ_{1k}. Those that are not in ξ_{1k} belong to ξ_{2k}. After μ_{ik}s and ξ_{1k}, ξ_{2k} are identified and $P^*(\mu_{ik})$s, $i \in \xi_{1k}$ are obtained, Equation 4.11 is used to obtain the $v^*(\mu_{ik})$s.

In a_k, if transitions $t_i \in C_1$ is enabled, the marking dependent firing rate $r_i(a_k)$ is obtained through Equation 4.14.

For example, from $\tilde{\mu}_1^1 = (179)$, Firing t_{10}, we have $\tilde{\mu}_2^1 = (1\ 10)$, $a_2 = (10)$. Putting transitions nine to eleven into the dormant state, all the transitions in subnet two are not live while all the transitions in subnet three are live. Hence, it is situation three. Its Markov Chain is shown in Figure 4.3. $\xi_{12} = \{(6\ 10), (5\ 10), (1\ 10)\}$. Analyze the Stochastic Petri Net to obtain $P^*(6\ 10)$, $P^*(5\ 10)$ and $P^*(1\ 10)$. In addition, we have

$$ps((679)|t_{10}) = (6\ 10),\ ps((579)|t_{10}) = (5\ 10)$$
$$ps((179)|t_{10}) = (1\ 10),\ ps((279)|t_{10}) = (2\ 10)$$

Marking (2 10) is not in ξ_{12} Therefore $\xi_{22} = (2\ 10)$. We choose (6 10) as μ_{xk}. Hence, according to Equation 4.11,

$$v^*(6\ 10) = \frac{P^*(6\ 10)}{1 + P^*(6\ 10) \times \frac{v^*(279)}{v^*(679)}}$$
$$v^*(5\ 10) = \frac{P^*(5\ 10)}{1 + P^*(6\ 10) \times \frac{v^*(279)}{v^*(679)}}$$

$$v^*(1\ 10) = \frac{P^*(1\ 10)}{1 + P^*(6\ 10) \times \frac{v^*(279)}{v^*(679)}}$$

$$v^*(2\ 10) = \frac{P^*(6\ 10) \times \frac{v^*(279)}{v^*(679)}}{1 + P^*(6\ 10) \times \frac{v^*(279)}{v^*(679)}}$$

Where $v^*(279)$ and $v^*(679)$ were obtained before by analyzing the Markov Chain in Figure 4.2.

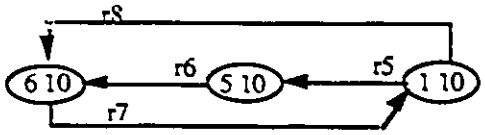

Figure 4.3: Markov Chain Associated with $a_2 = (10)$

In a_2, t_{11} is enabled and all the markings are favorite markings of t_{11}. hence,

$$\begin{aligned} r_{11}(10) &= r_{11} \times (v^*(6\ 10) + v^*(5\ 10) + v^*(1\ 10) + v^*(2\ 10)) \\ &= r_{11} \end{aligned} \tag{4.19}$$

After the marking dependent rates are obtained, the aggregated net is constructed. For example, the aggregated net for Figure 3.1 is subnet one with marking dependent rates specified by Equations 4.17, 4.18 and 4.19. We then analyze the reduced net to obtain its marking probabilities $P(a_k)$s.

Since the $v^*(\mu_{ik})$s obtained are the true conditional probabilities, according to Theorem 1 in Section 1.2, the Decomposition and Aggregation procedure gives exact results. Therefore, the aggregated net may concisely and exactly replace the original Stochastic Petri Net.

To retrieve the performance of the original Stochastic Petri Net from the aggregated net, Equations 4.13 may be used to obtain the marking probabilities. Since there is a unique association of classes and aggregated marking a_is, Equation 4.13 becomes:

$$P(\mu_{ik}) = v^*(\mu_{ik}) \times P(a_k) \tag{4.20}$$

To obtain the probability of the number of tokens in a place, we have:

$$P(m_i = n) = \sum_{k=1}^{M} P(a_k) \times P_k(m_i = n) \quad (4.21)$$

$$P_k(m_i = n) = \sum_{\forall \vec{m}: m_i = n \text{ in } \mu_{ik}} v^*(\mu_{ik}) \quad (4.22)$$

where m_i is the number of tokens in place i.

For example in Figure 3.1,

$$P(m_7 = 1) = P(78) \times P_1(m_7 = 1) + P(79) \times P_2(m_7 = 1) + P(10) \times P_3(m_7 = 1)$$

where

$$P_1(m_7 = 1) = v^*(678) + v^*(578) + v^*(178) + v^*(278)$$

$$P_2(m_7 = 1) = v^*(679) + v^*(579) + v^*(179) + v^*(279)$$

$$P_3(m_7 = 1) = 0$$

and $P(78)$, $P(79)$ and $P(10)$ are obtained from the aggregated net A.

We have developed the Norton's theorem for Local Balance Stochastic Petri Nets. In the next Section, we will show one of its applications to parametric analysis.

4.3 Application to Parametric Analysis

Suppose that we are interested in the influence of one transition on the whole system. Without loss of generality, suppose that the transition is in subnet one. We select subnet one to construct the aggregated net. Note that when analyzing the Stochastic Petri Net to obtain the marking dependent firing rates, we always put the transitions in net one into the dormant state. As a result, the transition rates of net one do not affect the marking probabilities. They appear only in the marking dependent firing rates. In fact, according to Equation 4.14 the marking dependent firing rate of a transition in the aggregated net is proportional to its firing rate in the original net. Hence, every time the firing rate of the transition of interest takes a new value, only the aggregated net needs to be analyzed.

Take the Stochastic Petri Net in Figure 3.1 as an example. Suppose that we are interested in finding out the effect of transition t_{11} on the performance of the whole system. We construct the aggregated net as before. Note that r_{11} only appears in Equation 4.19, i.e. the changing of r_{11} results in the changing of $r_{11}(10)$ only. Therefore, every time r_{11} takes a new value, only the aggregated net needs to be analyzed. The Markov Chain of the aggregated net has three states. Compared with the original sixteen-state Markov Chain, the savings of time and space are obvious. When the Markov Chain is large, the savings may be significant. For instance, the Markov Chain of the modified philosopher dining problem has two hundred and twelve states, whereas its aggregated net has only four states.

4.4 Summary

By applying the theory of decomposition and aggregation to the Markov Chains associated with Local Balance Stochastic Petri Nets, we developed the Decomposition by Subnet method, which provides us with an insight into the structure of the Markov Chains. Then by the Decomposition by Subnet method, we derived the Norton's theorem for Local Balance Stochastic Petri Nets. One subnet of the Stochastic Petri Net with marking dependent firing rates is used to replace the original net. Application of the aggregation process directly at the Stochastic Petri Net level is much more advantageous and desirable than at the Markov level. It is, of course, more difficult due to the complex structure of the Stochastic Petri Nets.

A significant difference between Norton's theorem for queueing networks and Norton's theorem developed in this Chapter is that there is no flow equivalent elements in the aggregated net to represent the rest of the Stochastic Petri Nets.

The idea may be interesting for system or protocol design as well as parametric analysis.

Chapter 5

Exact Parametric Analysis of Stochastic Petri Nets

Norton's theorem gives an exact and simplified presentation of a subnet. It may be used for exact parametric analysis of Local Balance Stochastic Petri Nets. When a Stochastic Petri Net is not a Local Balance Stochastic Petri Net, we also need parametric analysis. That is the subject of this Chapter.

In this Chapter, an algorithm for parametric analysis is derived. The computational gain achieved by the algorithm is then considered. The implementation of the algorithm to parametric analysis of Stochastic Petri Nets is discussed and an example is used to illustrate the algorithm. The results obtained are published in [43, 45].

5.1 Algorithm for Parametric Analysis

We will apply the "ideal aggregates" described in Section 1.2 to the analysis of Stochastic Petri Nets. First of all, it is possible to confine a given transition rate into the Q_1 in Figure 1.2. To do this, the state space of the Markov Chain associated with the Stochastic Petri Net is partitioned into two parts. The states immediately before or after the firing of the transition of interest belong to S_1. The rest of the states belong to S_2. The transition rates between the states in S_1 constitute the block matrix Q_1. The transition

rates between the states in S_2 constitute the block matrix $\mathbf{Q}_2$. The transition rates from the states in S_1 to the states in S_2 constitute the block matrix $\mathbf{E}$. The transition rates from the states in S_2 to the states in S_1 constitute the block matrix $\mathbf{F}$. In this way, the transition rate of interest will appear in $\mathbf{Q}_1$ only.

Next, from

$$(\pi_1 \ \pi_2)\begin{bmatrix} \mathbf{Q}_1 & \mathbf{E} \\ \mathbf{F} & \mathbf{Q}_2 \end{bmatrix} = \mathbf{0} \tag{5.1}$$

We have:

$$\pi_1\mathbf{E} + \pi_2\mathbf{Q}_2 = \mathbf{0} \tag{5.2}$$

$$\pi_2 = -\pi_1\mathbf{E}\mathbf{Q}_2^{-1} \tag{5.3}$$

Theorem 1 in Section 1.2 states that if exact results are obtained:

$$\pi_1 = G_1\mathbf{v}_1^* \tag{5.4}$$

Insert Equation 5.4 into Equation 5.3:

$$\pi_2 = -G_1\mathbf{v}_1^*\mathbf{E}\mathbf{Q}_2^{-1} \tag{5.5}$$

Assume $\mathbf{p}_1 = (p_{11}, \ldots p_{1n_1})$, $\mathbf{p}_2 = (p_{21}, \ldots p_{2n_2})$ and let:

$$\begin{aligned} \mathbf{p}_1 &= \frac{\pi_1}{G_1} \\ \mathbf{p}_2 &= \frac{\pi_2}{G_1} \end{aligned} \tag{5.6}$$

Divide Equations 5.4 and 5.5 by G_1, we have:

$$\begin{aligned} \mathbf{p}_1 &= \mathbf{v}_1^* \\ \mathbf{p}_2 &= -\mathbf{v}_1^*\mathbf{E}\mathbf{Q}_2^{-1} \end{aligned} \tag{5.7}$$

From Equation 5.6:

$$\sum_{i=1}^{n_1} p_{1i} + \sum_{i=1}^{n_2} p_{2i} = (\sum_{i=1}^{n_1} \pi_{1i} + \sum_{i=1}^{n_2} \pi_{2i})/G_1 = \frac{1}{G_1} \tag{5.8}$$

Noting that $\sum_{i=1}^{n_1} p_{1i} = 1$, we have:

$$G_1 = \frac{1}{1 + \sum_{i=1}^{n_2} p_{2i}} \tag{5.9}$$

According to Equation 5.6:

$$\begin{aligned} \pi_1 &= \mathbf{p}_1 G_1 = \frac{\mathbf{p}_1}{1 + \sum_{i=1}^{n_2} p_{2i}} \\ \pi_2 &= \mathbf{p}_2 G_1 = \frac{\mathbf{p}_2}{1 + \sum_{i=1}^{n_2} p_{2i}} \end{aligned} \tag{5.10}$$

If $\mathbf{v}_1^*$ is obtained by solving $\mathbf{v}_1^* \mathbf{Q}_1^* = \mathbf{0}$, $\mathbf{p}_1, \mathbf{p}_2$ may be obtained by Equation 5.7. Hence, π_1, π_2 may be obtained by Equation 5.10. Our algorithm is thus the following:

1. Compute $\mathbf{B} = \mathbf{E}\mathbf{Q}_2^{-1}$.
2. Compute $\mathbf{C} = \mathbf{B}\mathbf{F}$.
3. Compute $\mathbf{Q}_1^* = \mathbf{Q}_1 - \mathbf{C}$.
4. Solve $\mathbf{p}_1 \mathbf{Q}_1^* = \mathbf{0}$ subject to $\sum_{i=1}^{n_1} p_{1i} = 1$.
5. Compute $\mathbf{p}_2 = -\mathbf{p}_1 \mathbf{B}$.
6. Compute $G = 1 + \sum_{i=1}^{n_2} p_{2i}$.
7. Compute $\pi_i = \mathbf{p}_i / G \quad i = 1, 2$.

Since the transition rate of interest is in $\mathbf{Q}_1$ only, every time its value is changed, only steps 3 to 7 are repeated. The next section shows that the computational cost is greatly reduced in this way.

5.2 Time and Space Requirements

According to [61], the solution of n equations with n unknowns by Gaussian elimination requires $n^3/3 + n^2 - n/3$ multiplications and $n^3/3 + n^2/2 - 5n/6$ additions. The inversion of a matrix of order n requires n^3 number of multiplications and $n^3 - 2n^2 + n$ number of additions. Multiplying two matrices of orders $n_1 \times n_2$ and $n_2 \times n_3$ requires

$n_1 \times n_2 \times n_3$ number of multiplications and $n_1 \times (n_2 - 1) \times n_3$ number of additions. Therefore, the number of multiplications and the number of additions of each step in our algorithm are as follows:

Steps	Multiplications	Additions
1	$n_2^3 + n_1 n_2^2$	$n_2^3 - 2n_2^2 + n_2 + n_1(n_2 - 1)n_2$
2	$n_1^2 n_2$	$n_1^2(n_2 - 1)$
3	0	n_1^2
4	$n_1^3/3 + n_1^2 - n_1/3$	$n_1^3/3 + n_1^2/2 - 5n_1/6$
5	$n_1 n_2$	$(n_1 - 1)n_2$
6	0	n_2
7	$n_1 + n_2$	0

If the parameters of interest are assigned k different values, the total number of multiplications is:

$$cm_1 = n_2^3 + n_1 n_2^2 + n_1^2 n_2 + k(n_1^3/3 + n_1^2 + 2n_1/3 + n_1 n_2 + n_2) \tag{5.11}$$

and the total number of additions is:

$$\begin{aligned} ca_1 &= n_2^3 - 2n_2^2 + n_2 + n_1(n_2 - 1)n_2 + n_1^2(n_2 - 1) + \\ &+ k(n_1^3/3 + 3n_1^2/2 - 5n_1/6 + n_1 n_2) \end{aligned} \tag{5.12}$$

Direct solution of $\pi \mathbf{Q} = \mathbf{0}$ k times requires number of multiplications:

$$cm_2 = k(n^3/3 + n^2 - n/3) \tag{5.13}$$

and number of additions:

$$ca_2 = k(n^3/3 + n^2/2 - 5n/6) \tag{5.14}$$

Hence, the differences are :

$$\begin{aligned} cm_2 - cm_1 &= (k-1)(n_1^2 n_2 + n_1 n_2^2) + (k/3 - 1)n_2^3 + k(n_2 - 1)n_1 + kn_2(n_2 - 4/3) \\ ca_2 - ca_1 &= n_1^2(kn_2 - n_2 - k) + (k-1)n_1 n_2^2 + (k/3 - 1)n_2^3 + \\ &+ kn_2/2(n_2 - 5/3) + n_2(2n_2 - 1) + n_1 n_2 + n_1^2 \end{aligned} \tag{5.15}$$

For $k \geq 3, n_2 \geq 2$, all the terms are non-negative. Hence, our algorithm requires less number of computations than Gaussian elimination does. Figures 5.1 and 5.2 give an example of a moderate Markov chain with 100 states and $n_1 = 40$, $n_2 = 60$. The savings are obvious.

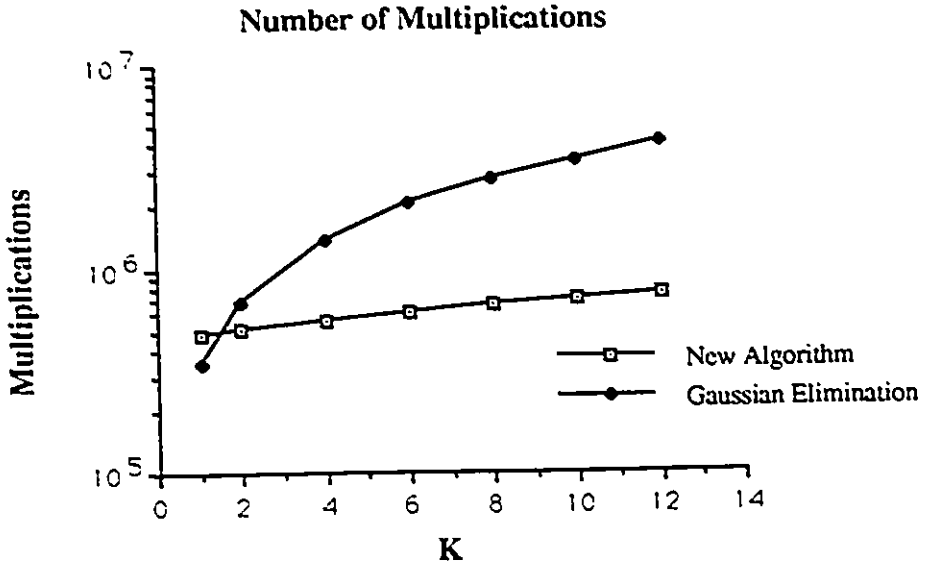

Figure 5.1: Number of Multiplications

In terms of space requirements, at the beginning, $\mathbf{Q}_1$, $\mathbf{Q}_2$, $\mathbf{E}$ and $\mathbf{F}$ are stored. After step two, however, only $\mathbf{B}$, $\mathbf{C}$ and $\mathbf{Q}_1$ are needed. Hence, the parametric analysis does not require any more space than the Gaussian elimination does. Besides, execution is much faster.

5.3 Implementation Consideration

The existing software for the performance analysis of Stochastic Petri Nets generally consists of three parts. In the first part, the user inputs the essential information about a Stochastic Petri Net. Examples are number of places, number of transitions, input(output) places of transitions, firing rates of transitions and initial number of tokens

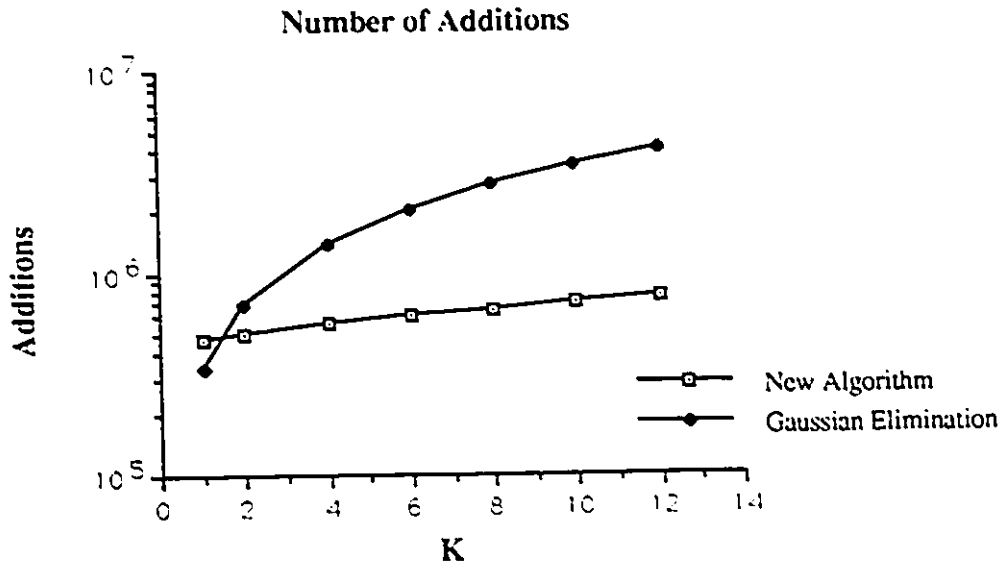

Figure 5.2: Number of Additions

in places. In the second part, the associated Markov Chain is generated. In the third part, the Markov Chain is solved and steady state probabilities are obtained. A typical way of solving the Markov Chain is the Gaussian elimination Method.

To implement our exact parametric analysis idea, the first part of the software is unchanged. In the second part, the user is asked if it is a parametric analysis. If the answer is "No", the usual procedure starts: If the answer is "yes", he is further asked to input the transition of interest. Then, at the same time of generating the associated Markov Chain, $\mathbf{Q}_1, \mathbf{Q}_2, \mathbf{E}, \mathbf{F}$ are identified. In the third part, our algorithm in Section 5.1 is adopted. Every time the transition rate of interest is changed, only steps 3 to 7 of our algorithm need to be repeated.

5.4 An Example

Consider a manufacturing machine repairing system. When a machine needs repair, it is sent to a waiting room. From there, it may be repaired either by technician 1 or by

technician 2 first and then technicians 1 and 2 together. After repairing, it is sent back to the factory. Suppose that the waiting room may contain at most 2 machines at the same time. The Stochastic Petri Net model is shown in Figure 5.3. The meanings associated with transitions and places are shown in table 5.1.

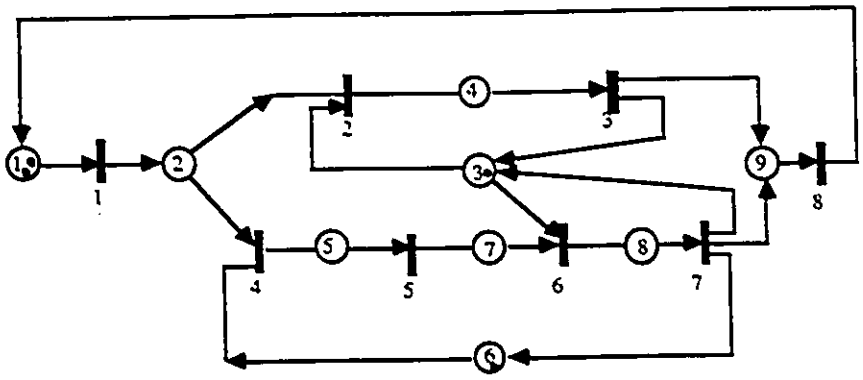

Figure 5.3: Stochastic Petri Net Model of the Manufacturing Machine Repairing System

Suppose that we are interested in knowing how the speed of technicians 1 and 2 working together will affect the whole system. Then transition t_7 is of interest and the infinitesimal matrix can be arranged as shown in Figure 5.4 so that r_7 is only in the first diagonal block matrix. The comparisons of computational costs for parameter analysis are shown in Figures 5.5 and 5.6.

5.5 Summary

We presented a method for exact parametric analysis of Stochastic Petri Nets. Following this method, every time a new value is assigned to the transition of interest, a smaller Markov Chain is analyzed. As a result, the computational cost is greatly reduced. The storage requirement is the same as that of Gaussian elimination. The algorithm is readily automated.

Places	
1	Machines in factories
2	Machines in the waiting room
3	Technician 1 repairing the machine
4	Technician 1 is available
5	Technician 2 repairing the machine
6	Technicians 2 and 3 repairing the machine together
7	Technician 3 is available
8	Technician 2 is available
9	Machine(s) is waiting to be sent to the factories
Trans	
1	Machines arrive for repairing
2	Technician 1 takes the machine
3	Technician 2 takes the machine
4	End of repairing the machine by technician 1
5	End of repairing the machine by technician 2 alone
6	End of repairing the machine by technicians 2 and 3 together
7	End of sending the machine to the factories

Table 5.1: Meanings of Transitions and Places of the Manufacturing Problem

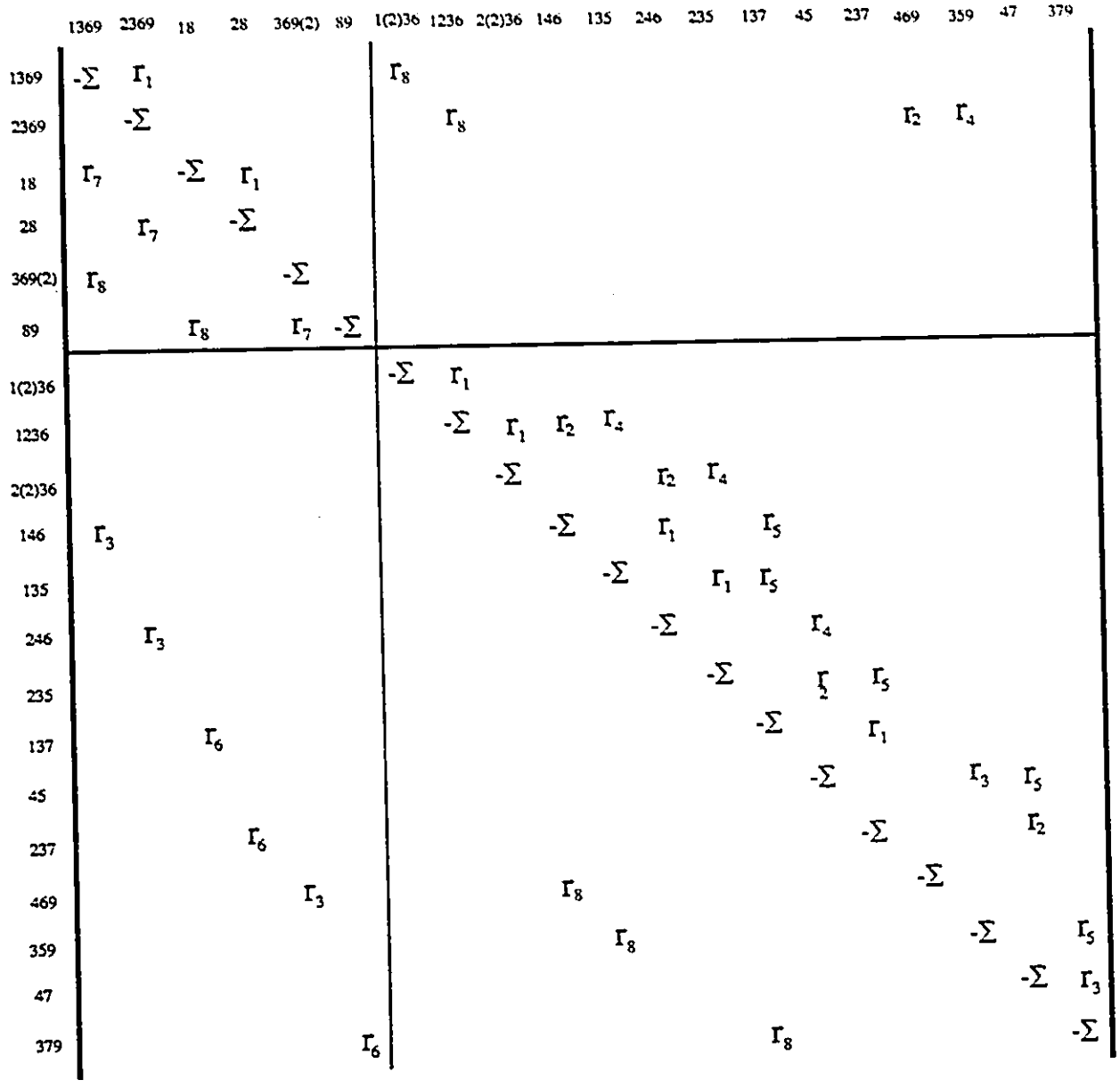

Figure 5.4: Matrix Q When t_7 Is of Interest

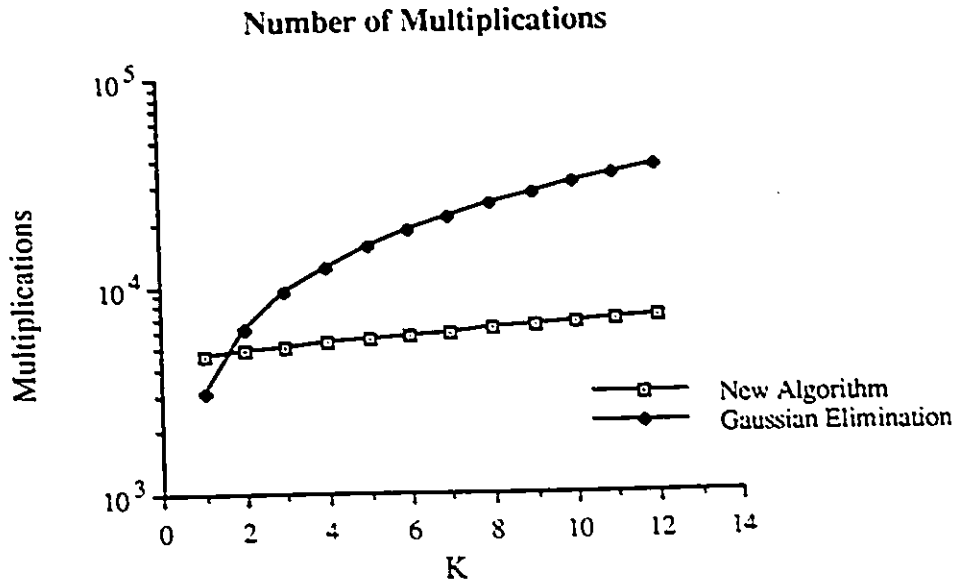

Figure 5.5: Number of Multiplications.

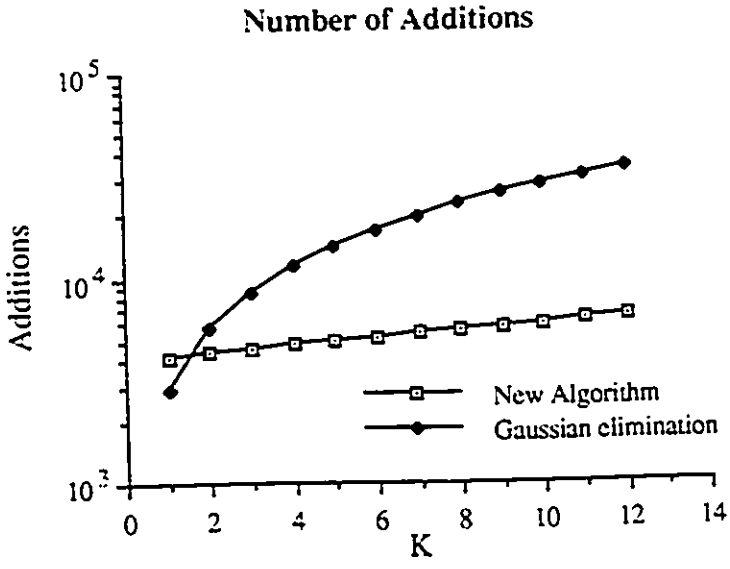

Figure 5.6: Number of Additions

Part II

Queueing Networks

Chapter 6

Introduction

In this Chapter, we give an introduction to queueing networks, review the different ways in applying Decomposition and Aggregation techniques to their analysis and introduce major algorithms developed over the last two decades for the analysis of BCMP queueing networks.

6.1 Introduction to Queueing Networks

A queue is a center, where customers come, wait in line for service, are served by servers according to certain service scheduling, and depart after service. A network of queues consists of a set of queues. A customer enters the network, queues for service and, upon the departure from a given queue, he either proceeds to some other queues to receive additional service or leaves the network.

The properties of a network are determined by the interarrival time distributions, service time distributions, service scheduling and customer routing. It is in general very difficult to obtain the transient behavior of a network. Instead, the steady state behavior of the process is analyzed. By assuming exponential time distributions of the interarrival time and the service time, a continuous time Markov Chain is recognized. If Q is the infinitesimal generator for the continuous time Markov Chain associated with the queueing network and π is the vector of equilibrium state distribution probabilities, then π may be

found by solving:

$$\pi Q = 0 \tag{6.1}$$

subject to

$$\sum_i \pi_i = 1$$

The mean performance measures of interest may be found from π.

In a network of queues, the number of states of the corresponding Markov Chain may be extremely large. As a result, direct solution of Equation 6.1 is generally prohibitive. Fortunately, by imposing certain restrictions on the characteristics of the workload and on the operation of the network resources, the associated Markov Chains satisfy the local balance equations. As a result, the networks themselves possess a simple form, the so called product form, solutions. These networks are often referred to as *product form queueing networks*. The discovery of the simple close form solutions greatly simplified the analysis.

The most simple product form network is the Jackson queueing network [33], which is a network with exponential service queues, each of the first-come first-served multi-server type. Subsequent generalizations to the classical Jackson network have been made over the years. In the following, we will consider, as the theory stands today, the product form queueing networks presented by Baskett, Chandy, Muntz and Palacios[4], the BCMP networks.

In the BCMP queueing networks, there may be different types of customers that are distinguished by their routing parameters, service time distributions and the possible class memberships they may take on. Assume that there are R different types of customers. A customer of type r may, in time, take different class memberships within a finite set of classes C_r. The sets of classes $C_1, ..., C_R$ are assumed to be mutually disjoint. Different classes may have different service time distributions and routing parameters. A customer of type r and in class c at node i which completes its service proceeds next to node j in class d with probability $p^{(r)}_{ic;jd}$ or leaves the network with probability $p^{(r)}_{ic;0}$.

Let N be the number of nodes in the network and C_{jr} be the set of classes that

a customer of type r may take on when at node j. Then $C_r = C_{1r} \cup ... \cup C_{Nr}$. The mean service time for a customer of type r in class c at node i is written as $m_{ic}^{(r)}$. The transition probability matrix $\mathbf{P}^{(r)} = [p_{ic,jd}^{(r)} : 1 \leq i,j \leq N, c \in C_{ir}, d \in C_{jr}]$ is referred to as the *routing chain* for type r customers. Customers of type r are usually called *chain* r customers. A BCMP network with multiple types of customers is commonly defined as a *multiple-chain queueing network*.

The service schedules allowed at the queues in a BCMP network are:

- First Come First Serve (FCFS), in which customers are served according to the order of their arrival;
- Last Come First Serve Preemptive Resume (LCFSPR), in which the latest arrival is served first, and service is resumed at the point of interruption;
- Infinite Server (IS), in which customers are served immediately after their arrival, no queue is formed;
- Processor Sharing (PS), in which customers are served simultaneously, everyone served with equal rate, the total rate being equal to the service rate.

If node i has the FCFS service scheduling, the service time distributions at node i have to be identical and exponential for all classes and all chains of customers. For the other type of nodes, the service time distributions should have rational Laplace transforms [4].

In a BCMP queueing network, customers may arrive from outside the network according to a Poisson process. Let $\lambda_{ic}^{(r)}$ denote the arrival rate to node i of type r customers in class c. If $\lambda_{ic}^{(r)} > 0$ for some $c \in C_{ir}$ and $i \in \{1, ..., N\}$, then the routing chain r is *open*. If $\lambda_{ic}^{(r)} = 0$ for all $c \in C_{ir}$ and $i \in \{1, ..., N\}$, then the routing chain r is *closed*. In a network, if all the routing chains are open, then it is an *open network*. If all the routing chains are closed, then it is a *closed network*. If some of the chains are open whereas others are closed, then it is a *mixed network*. When chain r is closed, there is a fixed population K_r of type r customers that circulate indefinitely inside the network. A closed

multiple-chain BCMP queueing network with N nodes and a population vector $\mathbf{K}$ is often denoted as $B(N,\mathbf{K})$, where $\mathbf{K} = (K_1, ... K_R)$. In this thesis, we will consider $B(N,\mathbf{K})$ only, since it is the most interest in practice.

Without loss of generality, assume that node 1 to J are of FCFS, PS or LCFSPR type and node $J+1$ to N are of IS type. Let k_{ir} denote the number of chain r customers at node i, regardless of their class membership, and let $k_i = \sum_{r=1}^{R} k_{ir}$, $k^{(r)} = \sum_{i=1}^{N} k_{ir}$, $\mathbf{k}_i = (k_{i1}, ..., k_{iR})$ and $\underline{\mathbf{k}} = (\mathbf{k}_1, ..., \mathbf{k}_N)$. The marginal state-distribution of the joint number of customers of each type at the nodes in $B(N,\mathbf{K})$ is given by [4]:

$$P^R(\underline{\mathbf{k}}) = G_N(\mathbf{K})^{-1} \prod_{i=1}^{N} f_i(\mathbf{k}_i) \tag{6.2}$$

where $\underline{\mathbf{k}} \in \underline{S}^R$, $\underline{S}^R$ is the state space of $B(N,\mathbf{K})$

$$\underline{S}^R = \{\underline{\mathbf{k}} : k_{ir} \geq 0,\ k^{(r)} = K_r,\ i = 1, ..., N, r = 1, ...R\}$$

$G_N(\mathbf{K})$ is a normalization constant of the network $B(N,\mathbf{K})$.

$$G_N(\mathbf{K}) = \sum_{\underline{\mathbf{k}} \in \underline{S}^R} \prod_{i=1}^{N} f_i(\mathbf{k}_i)$$

and

$$f_i(\mathbf{k}_i) = \begin{cases} k_i! \prod_{r=1}^{R} w_{ir}^{k_{ir}} / k_{ir}! & 1 \leq i \leq J \\ \prod_{r=1}^{R} w_{ir}^{k_{ir}} / k_{ir}! & J+1 \leq i \leq N \end{cases}$$

where

$$\begin{aligned} w_{ir} &= e_{ir} t_{ir} \\ e_{ir} &= \sum_{c \in C_{ir}} \alpha_{ic}^{(r)} \\ t_{ir} &= \sum_{c \in C_{ir}} \alpha_{ic}^{(r)} m_{ic}^{(ic)} / e_{ir} \end{aligned}$$

where the quantities $\alpha_{ic}^{(r)}$, $i = 1, ..., N$, $c \in C_{ir}$, satisfy the set of equations:

$$\alpha_{ic}^{(r)} = \sum_{j=1}^{N} \sum_{d \in C_{jr}} \alpha_{jd}^{(r)} p_{jd;ic}^{(r)} \quad c \in C_{ir},\ i = 1, ..., N$$

e_{ir} is the overall visit ratio of chain r customer at node i. t_{ir} is the overall mean service time of chain r customer at node i and w_{ir} is the *relative traffic intensity* of chain r customer at node i. Note that the marginal distribution only depends on the service time distributions through their means, and the distribution depends on $\mathbf{P}^{(r)}, m_{ic}^{(r)}$ through e_{ir} and t_{ir}. Therefore, as far as the marginal equilibrium state distribution is concerned, a multiple-chain BCMP network with class-switching may be mapped, almost trivially, into an equivalent network having exponential distributions and no class switching [65].

The BCMP queueing networks may accommodate several forms of state dependency. The service rate may be made to depend on the number of customers at the node. If $\mu_i(k_i)$ is the rate at which work is accomplished at node i when the population of the node is k_i, then the state distribution is the same as Equation 6.2, but with $f_i(\mathrm{k}_i)$ replaced by[4]:

$$f_i(\mathrm{k}_i) / \prod_{x=1}^{k_i} \mu_i(x)$$

Further generalizations have been made, refer to [14] for a summary.

6.2 Decomposition and Aggregation in Queueing Networks

The application of Simon and Ando's technique to the analysis of product form queueing networks was originated by Courtois[18, 19, 20]. He proposed the so called *Decomposition by Service Center* approach, where the analysis of $B(N, \mathbf{K})$ is decomposed into the analysis of:

- a set of queueing networks $\{B(N-1, \mathbf{K}-\mathrm{k}_p) : \mathrm{k}_p = (k_1, ..., k_R), 0 \leq k_r \leq K_r,\ r = 1, ..., R\}$.
- a reduced system which may be characterized as a network with two queues of state dependent service rates.

As a result, the analysis of the original network is broken down into the analysis of a group of subnetworks that consist of subnets of the queues.

Recently, Conway and Georganas established that a BCMP network may be transformed into a reversible network that consists of PS nodes only. The $P^R(\underline{k})$ of the two networks are equal. Since it is shown[19] that the Decomposition and Aggregation technique may be applied to a reversible network arbitrarily and exact results are guaranteed, the general Decomposition and Aggregation technique invariably yields exact results for product form queueing networks[14]. Hence, the $\mathbf{Q}_i^*$ mentioned in Section 1.2 may be constructed by simply truncating $\mathbf{Q}$ and taking $\mathbf{Q}_{ii}$. As a result, they chose the *Decomposition by Chain* approach where the analysis of $B(N,\mathbf{K})$ is broken down into the analysis of [14]:

- a set of queueing networks $\{\mathbf{M}(R-1,\mathbf{k}(R)) \ : \ \mathbf{k}(R) \in L_R\}$.
- a reduced system which may be characterized as a special type of single chain closed queueing network.

where $L_R = \{\mathbf{k}(R) \ : \ k_{iR} \geq 0 \ for \ 1 \leq i \leq N, \sum_{i=1}^{N} k_{iR} = K_R\}$ and $\mathbf{M}(R-1,\mathbf{k}_R)$ is a queueing network that has $R-1$ chains and there are k_{iR} single customer self looping chains at node i.

The value of the Decomposition and Aggregation methodology lies not so much in the numerical results it provides, but rather, in the theoretical insight which is provided concerning the structure of the problem at hand. It helps in developing computational algorithms for product form queueing networks. In fact, $\mathbf{Q}_i^*$ may itself be further decomposed. The general hierarchical multiple level decomposition of $B(N,\mathbf{K})$ is shown in Figure 6.1(from [14]). In the hierarchical Decomposition and Aggregation procedure, there may exist equivalent subsystems, i.e. systems with the same equilibrium state distribution, at each level. As a result, it is sufficient to analyze one subsystem from each of the equivalent subsystem sets. The inter-relationship between the equivalent subsystems at adjacent levels suggests the existence of computational algorithms. The overall computational requirements of the general recursive algorithm described above depend directly on the number of adopted levels L and the number of equivalence classes that may be identified at level n. This number is determined by the initial choice of state

space partitioning adopted.

Conway and Georganas further showed how all the existing major exact computational algorithms may be derived by applying the Decomposition and Aggregation procedure to $B(N, \mathbf{K})$[14]. For example, the Convolution algorithm may be derived from the decomposition by service center procedure; MVA and DAC may be derived from the decomposition by the disposition of a particular customer procedure; RECAL, MVAC may by derived from the decomposition by chain procedure. This unified point of view further suggests the existence of possibly more efficient algorithms [14]. In the next section, we will first review the existing algorithms.

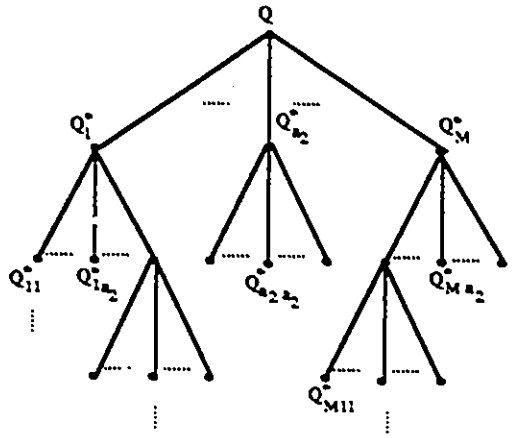

Figure 6.1: A General Hierarchical Multiple Level Decomposition of $B(N, \mathbf{K})$

6.3 Computational Algorithms for Queueing Networks

In this section, we summarize the major computational algorithms for product form queueing networks. All the existing computational algorithms are in the form of recursion. The recursion may be in terms of the normalization constant, the mean performance values or the joint queue length distributions.

6.3.1 The Convolution Algorithm

The Convolution algorithm involves the computation of the normalization constant. The mean performance measures are obtained in terms of this normalization constant and some by-products that have been obtained in the course of computing.

The recursion is over the nodes and the population vector as follows[7]:

$$G_m(\mathbf{k}_p) = \sum_{n_R=0}^{k_R} \cdots \sum_{n_2=0}^{k_2} \sum_{n_1=0}^{k_1} f_m(\mathbf{n}) G_{m-1}(\mathbf{k}_p - \mathbf{n}) \tag{6.3}$$

where $G_m(\mathbf{k}_p)$ is the normalization constant of $B(m, \mathbf{k}_p)$.

The initial condition is $G_1(\mathbf{k}_p) = f_1(\mathbf{k}_p)$. $G_N(\mathbf{K})$ is the normalization constant of interest. If node m has a constant service-rate function, Equation 6.3 may be simplified and $G_m(\mathbf{k}_p)$ may be computed by using the simpler recursive formula[7]:

$$G_m(\mathbf{k}_p) = G_{m-1}(\mathbf{k}_p) + \sum_{r=1}^{R} w_{mr} G_m(\mathbf{k}_p - \mathbf{1}_r) \tag{6.4}$$

Where $G_m(\mathbf{k}_p - \mathbf{1}_r) = 0$, if $k_r = 0$. $\mathbf{1}_r$ is a vector where the rth element is one and the rest of them are zeros.

The mean performance measures of $B(N, \mathbf{K})$ may be calculated as follows[38]:

$$\begin{aligned}
T_{ir}(\mathbf{K}) &= e_{ir} G_N(\mathbf{K} - \mathbf{1}_r)/G_N(\mathbf{K}) \\
U_{ir}(\mathbf{K}) &= t_{ir} T_{ir}(\mathbf{K}) \\
Q_{ir}(\mathbf{K}) &= \sum_{a=1}^{k_r} w_{ir}^a G_N(\mathbf{K} - a\mathbf{1}_r)/G_N(\mathbf{K}) \\
W_{ir}(\mathbf{K}) &= Q_{ir}/T_{ir}(\mathbf{K})
\end{aligned}$$

where $T_{ir}(\mathbf{K})$, $U_{ir}(\mathbf{K})$, $Q_{ir}(\mathbf{K})$ and $W_{ir}(\mathbf{K})$ are the throughput, utilization, mean queue length and mean waiting time of chain r customer at node i.

In the case of state-dependent service-rate functions, Q_{ir} is computed by using[38]:

$$\begin{aligned}
Q_{ir} &= \sum_{x=1}^{k_r} x p(k_{ir} = x) \\
p(k_{ir} = x) &= \sum_{\mathbf{n}_i \in A_{ir}(x)} p(\mathbf{k}_i = \mathbf{n}_i) \\
A_{ir}(x) &= \{\mathbf{n}_i : \mathbf{n}_i = (n_{i1}, ..., n_{iR}), 1 \le n_{is} \le K_s \ for \ 1 \le s \le R, s \ne r, n_{ir} = x\} \\
p(\mathbf{k}_i &= \mathbf{n}_i) = f_i(\mathbf{n}_i) G_{N-\{i\}}(\mathbf{K} - \mathbf{n}_i)/G_N(\mathbf{K})
\end{aligned}$$

where $G_{N-(i)}(\mathbf{k}_p)$ is the normalization constant of a network which is identical to $B(N,\mathbf{k})$ but with the ith node removed.

If the computation of the initial conditions is ignored and we assume that all nodes have constant service-rate functions, the total number of operations (multiplications and additions) to compute $G_N(\mathbf{K})$ using Equation 6.4 is

$$2R(N-1)\prod_{r=1}^{R}(K_r+1) \tag{6.5}$$

Note that 6.5 is exponential with R and linear with N. The storage space requirement, in number of array locations, is[7]:

$$2\prod_{r=1}^{R}(K_r+1) \tag{6.6}$$

If node m has a state dependent service rate function, then Equation 6.3 has to be used, which requires operations of the order of[36]:

$$\prod_{r=1}^{R}(K_r+1)(K_r+2)/2 \tag{6.7}$$

The space requirement is still as in Equation 6.6.

6.3.2 MVA-Mean Value Analysis Algorithm

The Mean Value Analysis algorithm requires a recursion that is directly in terms of mean performance measures and, sometimes, marginal queue-length distributions. MVA is based on the *Arrival theorem*[35, 37, 64] for product form queueing networks. Informally, the Arrival theorem says that the state distribution for $B(N,\mathbf{K})$ at arrival time of chain r customer is equal to the state distribution of $B(N,\mathbf{K}-\mathbf{1}_r)$ at arbitrary time. The recursion is over the population vector $\mathbf{k}_p$, $\mathbf{0} \leq \mathbf{k}_p \leq \mathbf{K}$. Let $P_i(j,\mathbf{k}_p)$ denote the probability that there are j customers at node i. The MVA consists of the following

recursive equations[7, 38, 66, 14].

$$W_{ir}(\mathbf{k}_p) = \begin{cases} t_{ir}[1 + Q_i(\mathbf{k}_p - \mathbf{1}_r)]; \\ \quad 1 \le i \le J, \text{ node } i \text{ has constant service rate} \\ t_{ir}[\sum_{j=1}^{k} j P_i(j-1, \mathbf{k}_p - \mathbf{1}_r)]/\mu_j; \\ \quad 1 \le i \le J, \text{ node } i \text{ has state dependent service rate} \\ t_{ir}; \quad J+1 \le i \le N \end{cases}$$

$$T_{ir}(\mathbf{k}_p) = e_{ir} K_r / \sum_{j=1}^{N} e_{jr} W_{jr}(\mathbf{k}_p)$$

$$Q_{ir}(\mathbf{k}_p) = T_{ir}(\mathbf{k}_p) W_{ir}(\mathbf{k}_p)$$

$$P_i(j, \mathbf{k}_p) = \sum_{s=1}^{R} t_{is} T_{is}(\mathbf{k}_p) P_i(j-1, \mathbf{k}_p - \mathbf{1}_s)/\mu_i(j)$$

$$P_i(0, \mathbf{k}_p) = 1 - \sum_{j=1}^{k} P_i(j, \mathbf{k}_p)$$

where $i = 1, ...N$, $r = 1, ...R$, $j = 1, ...k$, $k = \sum_{s=1}^{R} k_s$, $\mathbf{k}_p = (k_1, ...k_R)$. The initial conditions are $Q_{is}(\mathbf{0}) = 0$, $p_i(j, \mathbf{0}) = 0$ for $j > 0$, $p_i(0, \mathbf{0}) = 1$. If there is no state-dependent service-rate nodes in the network, the MVA algorithm does not require the computation of $P_i(j, \mathbf{k}_p)$. The time requirement in this case is approximately the same as Equation 6.5. The space requirement is [7]:

$$N \prod_{r=1}^{R} (K_r + 1)$$

Note that the time requirements of both Convolution and MVA algorithms are linear with N and exponential with R.

6.3.3 RECAL-Recursion by Chain Algorithm

The Recursion by Chain algorithm involves a recursion in terms of the normalization constant. To start, every chain that has more than one customer is converted into identical single customer chains. This transformation involves no loss in terms of the mean performance measures since a routing chain r consisting of K_r identical customers is entirely equivalent physically to K_r single-customer routing chains that have identical routing-parameters and mean service-time requirements[15].

The basic recursion of RECAL is as follows[15]:

$$G_r(\mathbf{v}_r) = \sum_{i=1}^{N}(1+v_{ir}\delta_i)w_{ir}G_{r-1}(\mathbf{v}_r+\mathbf{1}_i) \tag{6.8}$$

$$\delta_i = \begin{cases} 1 & \text{if node } i \text{ is FCFS,LCFSPR or PS} \\ 0 & \text{if node } i \text{ is IS} \end{cases} \tag{6.9}$$

$v_{ir} \in \{0,1,...\}$ for $1 \leq i \leq N$, $\sum_{i=1}^{N} v_{ir} = R - r$ where $G_r(\mathbf{v}_r)$ is the normalization constant of the network $\mathbf{M}(r,\mathbf{v}_r)$ which consists of the first r chains of the converted queueing network and v_{ir} single customer self-looping chains at node i. It is not difficult to see that $\mathbf{v}_R = \mathbf{0}$. The initial conditions are $G_0(\mathbf{v}_0) = 1$ for all $\mathbf{v}_0$.

The mean performance measures are:

$$T_{iR} = \begin{cases} e_{iR}\sum_{j=1}^{N} G_{R-1}(\mathbf{1_j})/(G_R(\mathbf{0})(N+K-1)) & N = J \\ e_{iR}G_{R-1}(\mathbf{1_N})/G_R(\mathbf{0}) & N > J \end{cases}$$

$$\begin{aligned} U_{iR} &= t_{ir}T_{iR} \\ Q_{iR} &= w_{ir}G_{R-1}(\mathbf{1}_i)/G_R(\mathbf{0}) \\ W_{iR} &= Q_{iR}/T_{iR} \end{aligned}$$

where K is the total population of customers in the network.

The above results are based on the assumption that $K_R = 1$. If the total number of single-customer routing chains, identical to chain R, is K_R^*, then the waiting time of the aggregated routing chain is the same as W_{ir} while the throughput, utilization and the mean queue length are multiplied by K_R^*. To obtain the mean performance measures for other chains, simply re-enumerate the chains so that the chain of interest is assigned the label R. Afterward, recompute $G_R(\mathbf{0})$. In [15], an efficient method to implement the algorithm is described in detail.

When there are state-dependent service-centers, the basic recursion of RECAL is modified as:

$$G_r(\mathbf{v}_r) = \sum_{i=1}^{N}(1+v_{ir}\delta_i)w_{ir}G_{r-1}(\mathbf{v}_r+\mathbf{1}_i)/\mu_i(1+v_{ir})$$

$$
\begin{aligned}
T_{iR} &= e_{iR} G_{R-1}(\mathbf{1}_N)/G_R(\mathbf{0}) \quad \text{when } N > P \\
Q_{iR} &= w_{ir} G_{R-1}(\mathbf{1}_i)/(G_R(\mathbf{0})\mu_i(1)) \\
W_{iR} &= Q_{iR}/T_{iR}
\end{aligned}
$$

The space requirement of RECAL is[15]:

$$\binom{K+N-1}{N-1} + \binom{R+N-1}{N-1}$$

and the time requirement (additions and multiplications) when the service rate is state independent is:

$$(4N-1)[\binom{K+N-1}{N} + \binom{R+N-1}{N+1} + \binom{R+N-1}{N}] + R(N+8)$$

the time requirement when the service rate is state dependent is[15]:

$$(5N-4)[\binom{K+N-1}{N} + \binom{R+N-1}{N+1} + \binom{R+N-1}{N}] + 8R$$

where K is the total population of customers in the network. When $R \to \infty$, the time requirement becomes polynomial with R and exponential with N. Hence, when R is large, RECAL is more efficient than Convolution and MVA algorithms.

6.3.4 MVAC-Mean Value Analysis by Chain Algorithm

The Mean Value Analysis by Chain algorithm is similar in structure to RECAL. But it calculates the mean performance values directly without computing the normalization constant first.

After transforming the network into single customer chains as RECAL does. MVAC involves the following recursion[17]:

$$
\begin{aligned}
T_r^{[r]}(\mathbf{v}_r) &= \frac{1}{w_r + \sum_{j=1}^{P} w_{jr}(Q_j^{[r-1]}(\mathbf{v}_r) + v_j)} \\
Q_{jr}^{[r]}(\mathbf{v}_r) &= T_r^{[r]}(\mathbf{v}_r) w_{jr}(1 + Q_j^{[r-1]}(\mathbf{v}_r) + v_j) \\
Q_j^{[r]}(\mathbf{v}_r) &= \sum_{i=1}^{N} Q_{ir}^{[r]}(\mathbf{v}_r) Q_j^{[r-1]}(\mathbf{v}_r + \mathbf{1}_i) + Q_{jr}^{[r]}(\mathbf{v}_r) \\
W_{jr}^{[r]}(\mathbf{v}_r) &= t_{jr}(1 + Q_j^{[r-1]}(\mathbf{v}_r) + v_j)
\end{aligned}
$$

where $T_r^{[r]}(\mathbf{v}_r)$ is the throughput of chain r customer of $\mathbf{M}(r, \mathbf{v}_r)$, $Q_{jr}^{[r]}(\mathbf{v}_r)$ and $Q_j^{[r]}(\mathbf{v}_r)$ are the queue length of node j and the queue length of chain r customer at node j respectivelly, $W_{jr}^{[r]}(\mathbf{v}_r)$ is the mean waiting time of chain r customer at node j and $w_r = \sum_{i=1}^{N} w_{ir}$. When $\mathbf{v}_r = \mathbf{0}$, the recursion ends. The results for chain R are thus obtained.

To obtain the mean performance measures for other chains, simply re-enumerate the chains so that the chain of interest is assigned the label R. In [17], an efficient method to implement the algorithm is described in detail.

The number of operations when there are no state dependent service queues is between

$$(N^2 + 2N)\{\binom{N+K}{N+1} + \binom{N+K+1}{N+2} - 1\}$$

and

$$(N^2 + 2N)\{\binom{N+K}{N+1} - 1\} + N^2\binom{N+R-1}{N+1}$$

where K is the total population in the network. When $R \to \infty$, the requirement is between R^{N+1} to R^{N+2}. It is polynomial with R and exponential with N.

6.3.5 DAC-Distribution Analysis by Chain Algorithm

The Distribution Analysis by Chain algorithm employs a recursion in terms of the joint marginal queue length distributions (The total number of customers at each center). It also provides the mean performance measures of the network. DAC has the recursive by chain nature as RECAL and MVAC do. As a result, its time requirement is still polynomial with the number of chains but exponential with the number of nodes. However, it computes the marginal queue length distributions, mean queue length and throughput more efficiently than both RECAL and MVAC algorithms.

As was done in RECAL and MVAC, every chain that has more than one customers is converted into identical single customer chains. Joint marginal queue length distributions are not affected by this conversion[70].

The fundamental equations used by DAC are [70]:

$$P^r(\mathbf{k}) = T_r \sum_{j=1}^{N} w_{jr} k_j P^{r-1}(\mathbf{k} - \mathbf{1}_j)/\mu_j(k_j) \tag{6.10}$$

$$T_r = [\sum_{j=1}^{N} w_{jr} \sum_{x=1}^{r} x P_j^{r-1}(x)/\mu_j(x)]^{-1} \tag{6.11}$$

$$P_j^{r-1}(x) = \sum_{k_j=x \ \mathbf{k} \in S^r} P^{r-1}(\mathbf{k}) \tag{6.12}$$

where $P^r(\mathbf{k})$ is the steady state probability that the joint queue length is $\mathbf{k} \in S^r$, $\mathbf{k} = \{k_1, ..., k_N\}$, T_r is the throughput of chain r and $P_j^r(k)$ is the steady state probability that there are k customers at node j.

The DAC algorithm is thus as follows[70]:

1. Set $r = 1$ and $P_j^1(1) = w_{j1} / \sum_{j=1}^{N} w_{j1}$ for $j = 1, ..., N$.
2. For $r = 2, ..., R$.
   (a) Use Equation 6.11 to calculate T_r.
   (b) Use Equation 6.10 to calculate $P^r(\mathbf{k})$ for all $\mathbf{k} \in S^r$.
   (c) Use Equation 6.12 to calculate $P_j^r(k)$ for all $j = 1, ..., N$ and $k = 0, ..., r$.

The set of recursions yield the joint queue length distributions of the network as well as the mean performance measures for chain R. To obtain throughputs for other routing chains, the following equation is used:

$$P^R(\mathbf{k}) = T_s \sum_{j=1}^{N} w_{js} k_j P^{R-1,s}(\mathbf{k} - \mathbf{1}_j)/\mu_j(k_j) \tag{6.13}$$

where $P^{R-1,s}(\mathbf{k} - \mathbf{1}_j)$ denotes the joint queue length distribution of the network with chain s removed. Therefore, T_d is computed as follows:

1. Assume $T_d = 1$.
2. By choosing:

$$\mathbf{k} = (r, 0, ...0), (r-1, 1, 0, ...0), \ ..., \ (0, ...0, r)$$

$P^{R-1,d}(\mathbf{k} - \mathbf{1}_j)$ can be derived from Equation 6.13.

3. $T_d = \sum_{\mathbf{k} \in S^{R-1,d}} P^{R-1,d}(\mathbf{k})$

The mean queue length for chain d is obtained by

$$Q_{jd}^R = T_d w_{jd} \sum_{k=1}^{R} k_j P^{R-1,s}(\mathbf{k} - \mathbf{1}_j)/\mu_j(k_j)$$

The number of multiplications of DAC is calculated as follows.

Step 1 requires N multiplications. In step 2 for a given r:

1. Calculating T_r requires $2Nr + N + 1$ multiplications.
2. In computing $P^r(\mathbf{k})$, $\mathbf{k} \in S^r$, every $P^{r-1}(\mathbf{k} - \mathbf{1}_j)$, $j = 1, ..., N$ will be multiplied by every w_{ir}, $i = 1, ..., N$ once and only once and there are $\binom{N-1+r-1}{N-1}$ values of $P^{r-1}(\mathbf{k})$. In addition, every $P^r(\mathbf{k})$, $\mathbf{k} \in S^r$ will be multiplied by T_r once and there are $\binom{N+r-1}{N-1}$ values of $P^r(\mathbf{k})$. As a result, the total number of multiplications for a given r is:

$$3N\binom{N+r-2}{N-1} + \binom{N+r-1}{N-1}$$

Hence, the total number of multiplications is:

$$\begin{aligned} & N + \sum_{r=2}^{R} [2Nr + N + 1 + 3N\binom{N+r-2}{N-1} + \binom{N+r-1}{N-1}] \\ = \; & (3N + \frac{N}{R} + 1)\binom{N+R-1}{N} + NR(R+2) + R - 6N - 2 \end{aligned} \tag{6.14}$$

The number of additions of DAC is calculated as follows.

In step 1, $N - 1$ additions are required.

In step 2, for a given r:

1. Computing T_r by Equation 6.11 requires $N(r-1) + (N-1)$ additions.
2. In step 2 for a given r, we showed that there are $3N\binom{N+r-2}{N-1}$ number of multiplications. Since in Equation 6.10, each term of the summation has 3 multiplications, there are $N\binom{N+r-2}{N-1}$ terms for a given r. To calculate $P^r(\mathbf{k})$s, we need to separate the $N\binom{N+r-2}{N-1}$ terms into $\binom{N+r-1}{N-1}$ groups and sum up the terms in

each group. As a result, the total number of additions for a given r should be $N\binom{N+r-2}{N-1} - 1$ less $\binom{N+r-1}{N-1} - 1$, i.e.

$$N\binom{N+r-2}{N-1} - \binom{N+r-1}{N-1}$$

The total number of additions in step 2 is:

$$\sum_{r=2}^{R-1}[N\binom{N+r-2}{N-1} - \binom{N+r-1}{N-1}]$$

3. Computing $P_j^r(k)$ by Equation 6.12 for a given j and k requires

$$\binom{N-2+r-k}{N-2} - 1$$

additions. The total number of additions are then:

$$N\sum_{k=0}^{r}[\binom{N-2+r-k}{N-2} - 1] = N[\binom{N+r-1}{N-1} - r - 1] \qquad (6.15)$$

Therefore, the total number of additions of DAC is:

$$\begin{aligned}
&(N-1) + \sum_{r=2}^{R}[N(r-1) + (N-1) + N\binom{N+r-2}{N-1} - \binom{N+r-1}{N-1}] + \\
&N\sum_{r=2}^{R-1}[\binom{N+r-1}{N-1} - r - 1] \\
=\ &(2N - \frac{N}{R} - 1)\binom{N+R-1}{N} - N^2 + 2N - R + 1 \qquad (6.16)
\end{aligned}$$

When $R \to \infty$, the time costs are at most of the order of R^N. Although its computational costs still grow exponentially with the number of nodes, it computes the mean queue lengths and throughputs more efficiently that RECAL and MVAC. DAC claims to be the algorithm of choice, where MVAC or RECAL would previously have been the choice[70].

Chapter 7

IDAC - Improved Distribution Analysis by Chain Algorithm

Conway and Georganas showed how DAC may be constructively developed by the theory of Decomposition and Aggregation[14]. The DAC algorithm is a MVA-like algorithm in the sense that no normalization constants are involved. We show in the following sections that DAC is unnecessarily complex. If the normalization constants are calculated instead, the algorithm requires less computational time. Thus the efficiency of DAC is improved. In addition, the improved algorithm has benefit with regard to reducing possible numerical errors.

7.1 The IDAC Algorithm

Since T_r is the throughput of chain r, we have[7]:

$$T_r = G^{r-1}/G^r$$

where G^r is the normalization constant of the network when only the first r chains are present. Replacing T_r in Equation 6.10 by the above expression and multiplying both side by G^r, we have:

$$G^r P^r(\mathrm{k}) = \sum_{j=1}^{N} w_{j,r} k_j G^{r-1} P^{r-1}(\mathrm{k} - \mathbf{1}_j)/\mu_j(k_j) \tag{7.1}$$

which is

$$p^r(\mathbf{k}) = \sum_{j=1}^{N} w_{jr} k_j p^{r-1}(\mathbf{k} - \mathbf{1}_j)/\mu_j(k_j) \tag{7.2}$$

where $p^r(\mathbf{k})$ is the unnormalized steady state probability that the joint queue length is $\mathbf{k} \in S^r$.

Equation 7.2 is the fundamental equation of our IDAC algorithm. The IDAC algorithm is thus the following:

1. Set $r = 1$ and $p_j^1(1_j) = w_{j1}$ for $j = 1, ..., N$.
2. For $r = 2, ..., R-1$. Use Equation 7.2 to calculate $p^r(\mathbf{k})$ for all $\mathbf{k} \in S^r$.
3. Compute $G^{R-1} = \sum_{\mathbf{k} \in S^{R-1}} p^{R-1}(\mathbf{k})$.
4. Set $r = R$. Use Equation 7.2 to calculate $p^R(\mathbf{k})$ for all $\mathbf{k} \in S^R$.
5. Compute $G^R = \sum_{\mathbf{k} \in S^R} p^R(\mathbf{k})$
6. Compute $P^R(\mathbf{k}) = p^R(\mathbf{k})/G^R$ for all $\mathbf{k} \in S^R$
7. Compute $T_R = G^{R-1}/G^R$

This set of recursions yields the same results as the basic DAC algorithm does. We will observe later that the number of operations is less. As a result, IDAC also has the benefit of reducing possible numerical errors in computation.

7.2 Dynamic Scaling

Depending on the choice of w_{jr}, $j = 1, ..., N$, $r = 1, ...R$, the IDAC algorithm may suffer from the potential floating point underflow or overflow problem. The solution to this problem is, however, surprisingly simple. When the underflow or overflow is about to occur, simply multiply the present $p^r(\mathbf{k})$s by a scaling factor α. Then continue the algorithm. Since unnormalized joint queue length distributions are nothing but a set of values proportional to the joint queue length distributions, the scaling does not affect the final results. No memories are required to store the scaling factors, since there is no need

for re-multiplying by the scaling factors as the Convolution algorithm does. Note that if $p^{R-1}(\mathbf{k})$s are multiplied by a scaling factor so that the $p^R(\mathbf{k})$s as well as the sum of them are in an acceptable range, G^{R-1} should be the sum of the scaled $p^{R-1}(\mathbf{k})$s. This is to ensure that T_R obtained is the correct value.

Though the scaling introduces some additional multiplications, the number of operations of the IDAC is still less than that of the DAC. Suppose that scaling is done for every r. We include the scaling operation into the number of operations of Equation 7.2. As a result, the number of operations in Equation 7.2 will be equal to that of Equation 6.10. Since the evaluation of Equations 6.11 and 6.12 is avoided, the number of operations of IDAC is still less than that of DAC. In reality, scaling is performed few times, if necessary at all.

7.3 Computational Requirements

Assume that all nodes in the network have queue length dependent service rates. Savings can be achieved if some nodes have queue length independent service rates.

Multiplications

In step 2 of the IDAC algorithm, to calculate $p^r(\mathbf{k})$, $\mathbf{k} \in S^r$ by Equation 7.2, every $p^{r-1}(\mathbf{k}-\mathbf{1}_j)$, $j = 1, ..., N$ will be multiplied by every w_{ir}, $i = 1, ..., N$ once and only once. Since there are $\binom{N-1+r-1}{N-1}$ values of $p^{r-1}(\mathbf{k})$, the total number of multiplications for a given r is:

$$3N\binom{N+r-2}{N-1}$$

The total number of multiplications in step 2 is:

$$\sum_{r=2}^{R-1} 3N\binom{N+r-2}{N-1}$$

The total number of multiplications in step 4 is:

$$3N\binom{N+R-2}{N-1}$$

	DAC	IDAC	DAC	IDAC
N / R	2	2	4	4
10	632	336	10,045	8,855
20	2,377	1,276	118,640	108,020
40	9,167	4,956	1,623,405	1,493,250
60	20,357	11,036	7,798,270	7,187,680
80	35,947	19,516	24,007,236	22,143,312

Table 7.1: Number of Multiplication Requirements

The total number of multiplications in step 6 is:

$$\binom{N+R-1}{N-1}$$

The total number of multiplications in step 7 is 1.

Hence the total number of multiplications of the IDAC algorithm is:

$$\begin{aligned} & \sum_{r=2}^{R} 3N\binom{N+r-2}{N-1} + \binom{N+R-1}{N-1} + 1 \\ = \; & (3N + \frac{N}{R})\binom{N+R-1}{N} - 3N + 1 \end{aligned} \tag{7.3}$$

This value is smaller than that of Equation 6.14. Table 7.1 compares the number of multiplications of the two algorithms.

Additions

In step 2 for a given r, we showed that there are $3N\binom{N+r-2}{N-1}$ number of multiplications. Since in Equation 7.2, each term of the summation has 3 multiplications, there are $N\binom{N+r-2}{N-1}$ terms for a given r. To calculate $p^r(\mathbf{k})$s, we need to separate the $N\binom{N+r-2}{N-1}$ terms into $\binom{N+r-1}{N-1}$ groups and sum up the terms in each group. As a result, the total number of additions for a given r should be $N\binom{N+r-2}{N-1} - 1$ less $\binom{N+r-1}{N-1} - 1$, i.e.

$$N\binom{N+r-2}{N-1} - \binom{N+r-1}{N-1}$$

The total number of additions in step 2 is:

$$\sum_{r=2}^{R-1}[N\binom{N+r-2}{N-1}-\binom{N+r-1}{N-1}]$$

The total number of additions in step 3 is:

$$\binom{N-1+R-1}{N-1}-1$$

The total number of additions in step 4 is:

$$N\binom{N+R-2}{N-1}-\binom{N+R-1}{N-1}$$

The total number of additions in step 5 is:

$$\binom{N+R-1}{N-1}-1$$

Hence the total number of additions of the IDAC algorithm is:

$$\sum_{r=2}^{R}[N\binom{N+r-2}{N-1}-\binom{N+r-1}{N-1}]+\binom{N+R-1}{N-1}+\binom{N+R-2}{N-1}-2$$

$$= (N+\frac{N}{N+R-1}-1)\binom{N+R-1}{N}-1 \qquad (7.4)$$

This value is smaller than that of Equation 6.15. Table 7.2 compares the number of additions of the two algorithms.

Similar as in DAC, if only the joint queue length distributions at a subset of the nodes are required, further savings can be achieved.

7.4 Discussions

The computational savings of IDAC over DAC are achieved mainly by avoiding calculating the throughput and marginal queue lengths in every recursion. The space requirement of the IDAC is about the same as(if not less than) that of the DAC. In addition, IDAC has the benefit of reducing possible numerical errors in computation. Although the computational costs of the IDAC still grow exponentially with the number of nodes in the network. The IDAC algorithm is more efficient than the DAC. Since DAC has been shown to be more efficient than both RECAL and MVAC, IDAC is the choice where RECAL, MVAC or DAC had been the choice before.

	DAC	IDAC	DAC	IDAC
R \ N	2	2	4	4
10	145	64	4,702	2,364
20	590	229	60,187	28,104
40	2,380	859	851,482	381,709
60	5,370	1,889	4,129,877	1,824,814
80	9,560	3,319	12,771,372	5,601,419

Table 7.2: Number of Addition Requirements

Chapter 8

Analyzing Queueing Networks by Independent Decomposition And Aggregation

In this chapter, we apply the Decomposition and Aggregation techniques and develop an Independent Decomposition and Aggregation method for the analysis of BCMP queueing networks. According to this method, we derive a new algorithm called Adaptive Convolution by Chain algorithm(ACCAL).

First of all, we transform the non-IS type nodes in $B(N,\mathbf{K})$ into IS type nodes. Denote the transformed network by $B'(N,\mathbf{K})$. We derive the performance relationship between $B(N,\mathbf{K})$ and $B'(N,\mathbf{K})$. Then we apply the Decomposition and Aggregation technique to the analysis of $B'(N,\mathbf{K})$. The transformation of $B(N,\mathbf{K})$ into $B'(N,\mathbf{K})$ makes our constructive development of ACCAL different from Conway and Georganas' methodology for construction of exact computational algorithms[14]. Recall that they transformed a closed BCMP network into a network with PS nodes only.

A major concern in developing an algorithm is reducing the computational time. One way is to reduce the number of mathematic operations an algorithm needs. Another way is to take advantage of parallel processing, since nowadays multiprocessor systems are well developed. In other words, if an algorithm consists of smaller tasks that may

be executed in parallel, the total execution time of the algorithm will be reduced in a multiprocessor system. ACCAL is a result of the combined efforts from both ways. ACCAL is efficient in dealing with queueing networks with many chains and a few nodes. In addition, it has the following characteristics:

1. *Faster.* It has smaller computational requirements than IDAC does. Hence it is more efficient in dealing with networks that have many chains and a few nodes than any existing algorithms.
2. *Adaptive.* According to the number of nodes in the network and other information, the algorithm converts the network into an equivalent one that can be more efficiently analyzed by the algorithm.
3. *Parallel Processing.* The algorithm has three independent parts that may be executed in parallel in a multiprocessor system. In this way, the computational time is further reduced.

The adaptive nature and the parallel processing characteristic distinguish ACCAL from any of the existing algorithms.

8.1 Transformation of $B(N, \mathbf{K})$ into $B'(N, \mathbf{K})$

Following the assumption in Section 6.1 that nodes 1 to J are of either FCFS, PS or LCFSPR type and nodes $J+1$ to N are of IS type. Replace nodes 1 to J by Infinite Service(IS) type nodes. As a result, all the nodes in the network are of IS type. The routing and the service time distributions for different chains are unchanged. Denote the transformed network by $B'(N, \mathbf{K})$. In $B'(N, \mathbf{K})$, define:

- $g^r(\mathrm{k})$: unnormalized steady state probability that the joint queue length is $\mathrm{k} \in S^r$.
- $g^r(\underline{\mathrm{k}})$: unnormalized steady state probability that the network is at state $\underline{\mathrm{k}} \in \underline{S}^r$.

In $B(N,\mathbf{K})$, define:

$$\begin{aligned} b(\mathrm{k}(r)) &= \prod_{i=1}^{N} \frac{w_{ir}^{k_{ir}}}{k_{ir}!} & \mathrm{k}(r) \in L_r \\ c(\mathrm{k}) &= \prod_{i=1}^{J} k_i! & \mathrm{k} \in S^R \\ u(\mathrm{k}) &= \prod_{i=1}^{J} \prod_{x=1}^{k_i} \mu_i(x) & \mathrm{k} \in S^R \end{aligned}$$

where $L_r = \{\mathrm{k}(r) : \sum_{i=1}^{N} k_{ir} = K_r\}$.

The relationship between $B(N,\mathbf{K})$ and $B'(N,\mathbf{K})$ may be described by the following Theorem:

Theorem 4

$$P^R(\mathrm{k}) = \frac{1}{G^R} \times c(\mathrm{k}) \times g^R(\mathrm{k})/u(\mathrm{k}) \tag{8.1}$$

$$p^R(\mathrm{k}) = c(\mathrm{k}) \times g^R(\mathrm{k})/u(\mathrm{k}) \tag{8.2}$$

Proof:

$$\begin{aligned} P^R(\mathrm{k}) &= \sum_{\sum_{r=1}^{R} k_{ir} = k_i} P^R(\underline{\mathrm{k}}) \\ &= \frac{1}{G^R} \sum_{\sum_{r=1}^{R} k_{ir} = k_i} \left(\prod_{i=1}^{J} \frac{k_i!}{\prod_{x=1}^{k_i} \mu_i(x)} \prod_{r=1}^{R} \frac{w_{ir}^{k_{ir}}}{k_{ir}!}\right)\left(\prod_{j=J+1}^{N} \prod_{r=1}^{R} \frac{w_{jr}^{k_{jr}}}{k_{jr}!}\right) \\ &= \frac{1}{G^R} \prod_{i=1}^{J} \frac{k_i!}{\prod_{x=1}^{k_i} \mu_i(x)} \sum_{\sum_{r=1}^{R} k_{ir} = k_i} \prod_{i=1}^{N} \prod_{r=1}^{R} \frac{w_{ir}^{k_{ir}}}{k_{ir}!} \\ &= \frac{1}{G^R} \times c(\mathrm{k}) \times g^R(\mathrm{k})/u(\mathrm{k}) \end{aligned} \tag{8.3}$$

Multiply both sides by G^R, we have:

$$p^R(\mathrm{k}) = c(\mathrm{k}) \times g^R(\mathrm{k})/u(\mathrm{k})$$

Q.E.D

In the next Section, we will apply the Decomposition and Aggregation technique to analyze $B'(N,\mathbf{K})$.

8.2 Independent Decomposition and Aggregation

We choose to decompose $B'(N,\mathbf{K})$ by the Rth chain customers. Since all the nodes in $B'(N,\mathbf{K})$ are of IS type, each customer behaves independently of others. As a result, we call the Decomposition and Aggregation procedure as *Independent Decomposition and Aggregation.*

We partition $\underline{S}^R$ into M sets of communicating states $S(\alpha_1)$, $1 \leq \alpha_1 \leq M$, where

$$S(\alpha_1) = \{\underline{\mathrm{k}} : \underline{\mathrm{k}} \in \underline{S}^R, \mathrm{k}(R) = \mathrm{n}_R, \alpha_1 = \xi^R(\mathrm{k}(R))\}$$

$\mathrm{k}(R) = (k_{1R}, ..., k_{NR})$, $\mathbf{n}_R = (n_{1R}, ..., n_{NR})$ and $\xi^R(.)$ is a function which maps the set of vectors

$$\{\mathrm{k}(R) \ : \ k_{iR} \geq 0, \ 1 \leq i \leq N, \ \sum_{i=1}^{N} k_{iR} = K_R\}$$

into the set of integers

$$\{1, ..., \binom{K_R + N - 1}{N - 1}\}$$

The vector $\mathrm{k}(R)$ is the state of the customers of chain R in the network $B'(N,\mathbf{K})$.

Since all the nodes are of IS type, it may be seen that the set of states $S(\alpha_1)$ corresponds exactly with the state space of the queueing network $B'(N, \mathbf{K} - \mathrm{k}(R))$, where

$$g^{R-1}(\underline{\mathrm{k}}) = \prod_{i=1}^{N} \prod_{r=1}^{R-1} \frac{w_{ir}^{k_{ir}}}{k_{ir}!}$$

As a result, the number of equivalence classes at this level is one.

The next step is to establish a recursive relationship between $g^R(\mathrm{k})$ and $g^{R-1}(\mathrm{k})$. This relationship may be expressed by the following Theorem:

Theorem 5

$$g^R(\mathrm{k}) = \sum_{\mathrm{k}(R) \in \mathcal{L}_R} b(\mathrm{k}(R)) \times g^{R-1}(\mathrm{k} - \mathrm{k}(R)) \tag{8.4}$$

where $g^{R-1}(\mathrm{k} - \mathrm{k}(R)) = 0$ *if any term in* $\mathrm{k} - \mathrm{k}(R)$ *is negative.*

Proof:

$$
\begin{aligned}
g^R(\mathrm{k}) &= \sum_{\sum_{r=1}^{R} k_{ir}=k_i} \prod_{i=1}^{N} \prod_{r=1}^{R} \frac{w_{ir}^{k_{ir}}}{k_{ir}!} \\
&= \sum_{k_{iR}+\sum_{r=1}^{R-1} k_{ir}=k_i} \prod_{i=1}^{N} \frac{w_{iR}^{k_{iR}}}{k_{iR}!} \prod_{i=1}^{N} \prod_{r=1}^{R-1} \frac{w_{ir}^{k_{ir}}}{k_{ir}!} \\
&= \sum_{\mathrm{k}(R)\in L_R} \prod_{i=1}^{N} \frac{w_{iR}^{k_{iR}}}{k_{iR}!} \sum_{\sum_{r=1}^{R-1} k_{ir}=k_i-k_{iR}} \prod_{i=1}^{N} \prod_{r=1}^{R-1} \frac{w_{ir}^{k_{ir}}}{k_{ir}!} \\
&= \sum_{\mathrm{k}(R)\in L_R} b(\mathrm{k}(R)) \times g^{R-1}(\mathrm{k}-\mathrm{k}(R))
\end{aligned}
$$

where a term in the third line will be zero if any of the $k_i - k_{iR}$, $i = 1,....N$ in that term is negative, since the number of customers should never be negative.

Q.E.D

If we continue to apply the single level decomposition to $B'(N, \mathbf{K} - \mathrm{k}(R))$, then at the rth level in the resulting hierarchy, we have $B'(N, \mathbf{K} - \mathrm{k}(R) - ... - \mathrm{k}(R-r+1))$. Therefore, the number of levels in the hierarchy is R and the number of equivalent class at each level is one. The recursive relationship between level r and $r+1$ is:

Corollary 1

$$
g^r(\mathrm{k}) = \sum_{\mathrm{k}(r)\in L_r} b(\mathrm{k}(r)) \times g^{r-1}(\mathrm{k}-\mathrm{k}(r)) \tag{8.5}
$$

where $g^{r-1}(\mathrm{k}-\mathrm{k}(r)) = 0$ *if any term in* $\mathrm{k}-\mathrm{k}(r)$ *is negative.*

The above equation is the main recursion of our new algorithm. Since it is similar to a convolution operation, we name the algorithm the Convolution by Chain algorithm. In the next Section, we will present the new algorithm.

8.3 The Convolution By Chain Algorithm

This section presents the basics of the Adaptive Convolution By Chain Algorithm. Let $\mathbf{e}_j$ be a compatible vector whose jth element is one and other elements are zero. First of all, we need the following Corollaries:

Corollary 2 *If* $K_R = 1$

$$g^R(\mathbf{k}) = \sum_{i=1}^{N} w_{iR} \times g^{R-1}(\mathbf{k} - \mathbf{e}_i) \tag{8.6}$$

Proof. The corollary follows from Theorem 1 and the definition of $b(\mathbf{k}(r))$.

Q.E.D

Corollary 3

$$b(\mathbf{k}(r) + \mathbf{e}_i - \mathbf{e}_j) \;=\; b(\mathbf{k}(r)) \times \frac{w_{ir}}{k_{ir}+1} \times \frac{k_{jr}}{w_{jr}} \tag{8.7}$$

$$u(\mathbf{k}+\mathbf{e}_i-\mathbf{e}_j) = \begin{cases} u(\mathbf{k}) \times \frac{\mu_i(k_i+1)}{\mu_j(k_j)} & 1 \le i,j \le J \\ u(\mathbf{k}) \times \mu_i(k_i+1) & 1 \le i \le J,\ J+1 \le j \le N \\ \frac{u(\mathbf{k})}{\mu_j(k_j)} & 1 \le j \le J,\ J+1 \le i \le N \\ u(\mathbf{k}) & J+1 \le i,j \le N \end{cases} \tag{8.8}$$

$$u(\mathbf{k}) = \begin{cases} u(\mathbf{k}-\mathbf{e}_i) \times \mu_i(k_i) & \mathbf{k}-\mathbf{e}_i \in S^{R-1} \quad 1 \le i \le J \\ u(\mathbf{k}-\mathbf{e}_i) & \mathbf{k}-\mathbf{e}_i \in S^{R-1} \quad J+1 \le i \le N \end{cases} \tag{8.9}$$

$$c(\mathbf{k}+\mathbf{e}_i-\mathbf{e}_j) = \begin{cases} c(\mathbf{k}) \times \frac{k_i+1}{k_j} & 1 \le i,j \le J \\ c(\mathbf{k}) \times (k_i+1) & 1 \le i \le J,\ J+1 \le j \le N \\ \frac{c(\mathbf{k})}{k_j} & 1 \le j \le J,\ J+1 \le i \le N \\ c(\mathbf{k}) & J+1 \le i,j \le N \end{cases} \tag{8.10}$$

$$c(\mathbf{k}) = \begin{cases} c(\mathbf{k}-\mathbf{e}_i) \times k_i & \mathbf{k}-\mathbf{e}_i \in S^{R-1} \quad 1 \le i \le J \\ c(\mathbf{k}-\mathbf{e}_i) & \mathbf{k}-\mathbf{e}_i \in S^{R-1} \quad J+1 \le i \le N \end{cases} \tag{8.11}$$

These are easy to verify by their definitions.

The following is the Convolution By Chain Algorithm:

1. a, b and c may be executed in parallel.

(a) For $r = 1, ..., R$

Compute $b(\mathrm{k}(r))$ by Equation 8.7 with initial state $(K_r, 0, .., 0)$ and $b(K_r, 0..., 0) = w_{1r}^{K_r}/K_r!$ and compute $g^r(\mathrm{k})$ by Equation 8.5 with initial $g^1(\mathrm{k}) = b(\mathrm{k}(1))$ for $\mathrm{k} = \mathrm{k}(1)$.

(b) Compute $c(\mathrm{k})$ by Equation 8.10 with initial $c(K, 0..., 0) = K!$.

(c) Compute $u(\mathrm{k})$ by Equation 8.8 with initial $u(K, 0..., 0) = \prod_{i=1}^{K} \mu_1(i)$.

2. Compute $p^R(\mathrm{k})$ by Equation 8.2.

3. Compute $G^R = \sum_{\mathrm{k} \in S^R} p^R(\mathrm{k})$.

4. Compute $P^R(\mathrm{k}) = p^R(\mathrm{k})/G^R$

When mean values are required, the above algorithm is modified. Without loss of generality, assume that we are interested in the performance of the Rth chain. The Rth chain is converted into a chain with $K_R - 1$ customers and a $(R+1)th$ chain with one customer. Joint queue length distributions are not affected by this conversion.

When mean values are required, the Convolution by Chain Algorithm is modified as follows:

1. a, b and c are independent and may be executed in parallel.

(a) For $r = 1, ..., R$

Compute $b(\mathrm{k}(r))$ by Equation 8.7 with initial $b(K_r, 0..., 0) = w_{1r}^{K_r}/K_r!$ and compute $g^r(\mathrm{k})$ by Equation 8.5 with initial $g^1(\mathrm{k}) = b(\mathrm{k}(1))$ for $\mathrm{k} = \mathrm{k}(1)$.

(b) Compute $c(\mathrm{k})$, $\mathrm{k} \in S^R$ by Equation 8.10 with initial $c(K-1, 0..., 0) = (K-1)!$ and compute $c(\mathrm{k})$, $\mathrm{k} \in S^{R+1}$ by Equation 8.11.

(c) Compute $u(\mathrm{k})$, $\mathrm{k} \in S^R$ by Equation 8.8 with initial $u(K-1, 0..., 0) = \prod_{i=1}^{K-1} \mu_1(i)$ and compute $u(\mathrm{k})$, $\mathrm{k} \in S^{R+1}$ by Equation 8.9.

2. Compute $p^R(\mathrm{k})$ by Equation 8.2.

3. Compute $G^R = \sum_{\mathrm{k} \in S^R} p^R(\mathrm{k})$.

4. Compute $g^{R+1}(\mathrm{k})$ by Equation 8.6.

5. Compute $p^{R+1}(\mathrm{k})$ by Equation 8.2.

6. Compute $G^{R+1} = \sum_{\mathrm{k} \in S^{R+1}} p^{R+1}(\mathrm{k})$.

7. Compute $T^{*}_{R+1} = G^{R}/G^{R+1}$

8. Compute $P^{R+1}(\mathrm{k}) = p^{R+1}(\mathrm{k})/G^{R+1}$

where T^{*}_{R+1} is the throughput of the $(R+1)th$ chain of the converted network.

For the purpose of comparison with other algorithms, we did not include the calculation of the mean performance values for the original network. However, those values may be obtained as follows:

1. $T_R = K_R \times T^{*}_{R+1}$

2. $Q_j = \sum_{k=1}^{K} k \times \sum_{\mathrm{k} \in S^{R+1}, k_j = k} P^{R+1}(\mathrm{k})$

3. $Q_{jR} = T_R \times w_{jR} \times \sum_{k=1}^{K} k/\mu_j(k) \times \sum_{\mathrm{k} \in S^{R}, k_j = k-1} P^{R}(\mathrm{k})$

We will discuss the method for obtaining the performance measures of other chains in Section 8.5.

An interesting point of the new algorithm is that in step one of the algorithm, there are three independent parts that may be executed in parallel in a multiprocessor system. In the next Section, we will examine the time requirements of the algorithm.

8.4 Computational Requirements

Assume that each chain has m customers. Further assume that all nodes are of FCFS, PS or LCFSPR type and have state dependent servers. Savings may be achieved if there are IS nodes or state independent servers. We consider sequential processing and parallel processing separately.

8.4.1 Sequential Processing

When only joint distributions are required

1. Initialization of $b(\mathbf{k}(r))$ requires $2m-2$ multiplications. Calculation of $b(\mathbf{k}(r))$ by Equation 8.7 requires 4 multiplications each time and there are $\binom{m+N-1}{N-1}$ values of $\mathbf{k}(r)$ for a given r. When computing $g^r(\mathbf{k})$ by Equation 8.5 for a given r, every $b(\mathbf{k}_r)$ will be multiplied by every $g^{r-1}(\mathbf{k})$ once and only once. There are $\binom{(r-1)m+N-1}{N-1}$ values of $g^{r-1}(\mathbf{k})$. Therefore, the total number of multiplications in step 1(a) is:

$$R\{2m-2+4[\binom{m+N-1}{N-1}-1]\}+\sum_{k=1}^{R-1}\binom{m+N-1}{N-1}\binom{km+N-1}{N-1} \quad (8.12)$$

2. Initialization of $c(\mathbf{k})$ requires $Rm-2$ multiplications. Calculation of $c(\mathbf{k})$ by Equation 8.10 requires at most 2 multiplications each time and there are total $\binom{Rm+N-1}{N-1}$ values of k. Hence, the total number of multiplications in step 1(b) is:

$$Rm-2+2[\binom{Rm+N-1}{N-1}-1]$$

3. Initialization of $u(\mathbf{k})$ requires $Rm-1$ multiplications. Calculation of $u(\mathbf{k})$ by Equation 8.8 requires at most 2 multiplications each time and there are $\binom{Rm+N-1}{N-1}$ values of k. Hence, the total number of multiplications in step 1(c) is:

$$Rm-1+2[\binom{Rm+N-1}{N-1}-1]$$

4. Computation of $p(\mathbf{k})$ by Equation 8.2 requires 2 multiplications each time and there are $\binom{Rm+N-1}{N-1}$ values of k. As a result, the total number of multiplications in step 2 is:

$$2\binom{Rm+N-1}{N-1}$$

5. Step 4 requires number of multiplications:

$$\binom{Rm+N-1}{N-1}$$

The total number of multiplications of the algorithm is:

$$\binom{m+N-1}{N-1}\sum_{k=1}^{R-1}\binom{km+N-1}{N-1}+4R\binom{m+N-1}{N-1}$$
$$+\ 7\binom{Rm+N-1}{N-1}+4Rm-6R-7$$

When mean values are required

1. The computation of $b(\mathbf{k}(r))$ and $g^r(\mathbf{k})$ for the first $R-1$ chains requires $(R-1)\{2m-2+4[\binom{m+N-1}{N-1}-1]\}+\sum_{k=1}^{R-2}\binom{m+N-1}{N-1}\binom{km+N-1}{N-1}$ multiplications.The Rth chain has $m-1$ customers. Hence, the calculation of $b(\mathbf{k}(R))$ requires $4[\binom{m+N-2}{N-1}-1]+2(m-1)-2$ multiplications and the calculation of $g^R(\mathbf{k})$ requires $\binom{(R-1)m+N-1}{N-1}\binom{m+N-2}{N-1}$ multiplications. Therefore, the total number of multiplications in step 1(a) is:

$$(R-1)\{2m-2+4[\binom{m+N-1}{N-1}-1]\}+4[\binom{m+N-2}{N-1}-1]$$
$$+\ 2(m-1)-2++\sum_{k=1}^{R-2}\binom{m+N-1}{N-1}\binom{km+N-1}{N-1}$$
$$+\ \binom{(R-1)m+N-1}{N-1}\binom{m+N-2}{N-1}$$

2. Computing $c(\mathbf{k}-\mathbf{1}_i)$ by Equation 8.10 requires at most $Rm-3+2[\binom{Rm+N-2}{N-1}-1]$ multiplications. Computing $c(\mathbf{k})$ by Equation 8.11 requires at most $\binom{Rm+N-1}{N-1}$ multiplications. Hence, the total number of multiplications in step 1(b) is:

$$Rm-3+2[\binom{Rm+N-2}{N-1}-1]+\binom{Rm+N-1}{N-1} \qquad (8.13)$$

3. Computing $u(\mathbf{k}-\mathbf{1}_i)$ by Equation 8.8 requires at most $Rm-2+2[\binom{Rm+N-2}{N-1}-1]$ multiplications. Computing $u(\mathbf{k})$ by Equation 8.9 requires at most $\binom{Rm+N-1}{N-1}$. Therefore, the total number of multiplications in step 1(c) is:

$$Rm-2+2[\binom{Rm+N-2}{N-1}-1]+\binom{Rm+N-1}{N-1} \qquad (8.14)$$

4. Calculating $p^R(\mathrm{k})$ by Equation 8.2 requires

$$2\binom{Rm+N-2}{N-1} \tag{8.15}$$

multiplications.

5. Computing $g^{R+1}(\mathrm{k})$ by Equation 8.6 requires

$$N\binom{Rm+N-2}{N-1} \tag{8.16}$$

multiplications.

6. Computing $p^{R+1}(\mathrm{k})$ by Equation 8.2 requires

$$2\binom{Rm+N-1}{N-1} \tag{8.17}$$

7. Computing T requires 1 multiplication.

8. Computing $P^{R+1}(\mathrm{k})$ requires

$$\binom{Rm+N-1}{N-1} \tag{8.18}$$

multiplications.

The total number of multiplications is:

$$\begin{aligned} & \binom{m+N-1}{N-1}\sum_{k=1}^{R-2}\binom{km+N-1}{N-1} + \binom{(R-1)m+N-1}{N-1}\binom{m+N-2}{N-1} + \\ + \; & (N+6)\binom{Rm+N-2}{N-1} + 5\binom{Rm+N-1}{N-1} + 4(R-1)\binom{m+N-1}{N-1} + \\ + \; & 4\binom{m+N-2}{N-1} + 4mR - 6R - 10 \end{aligned} \tag{8.19}$$

8.4.2 Parallel Processing

In parallel processing, the time requirement for step 1 is equal to the maximum of that of 1(a), 1(b) and 1(c). Obviously, 1(a) requires a higher number of multiplications than either 1(b) or 1(c). Hence, the number of multiplications in parallel processing is:

N	R	m	SAVINGS (JOINT ONLY)	SAVINGS (MEAV VALU.)
2	4	8	1.89E+02	2.48E+02
2	8	8	3.81E+02	5.04E+02
6	8	4	1.743645E+06	2.379816E+06
6	12	4	1.1478829E+07	1.6135296E+07
10	12	4	3.5985854E+10	4.82968E+10
10	16	4	3.883281E+11	5.3461611E+11

Table 8.1: Examples of the Savings of the Parallel over the Sequential Processing

1. When only joint distributions are required.

$$\binom{m+N-1}{N-1}\sum_{k=1}^{R-1}\binom{km+N-1}{N-1}+4R\binom{m+N-1}{N-1}$$
$$+\ 3\binom{Rm+N-1}{N-1}+2Rm-6R$$

2. When mean values are required.

$$\binom{m+N-1}{N-1}\sum_{k=1}^{R-2}\binom{km+N-1}{N-1}+\binom{(R-1)m+N-1}{N-1}\binom{m+N-2}{N-1}$$
$$+\ (N+2)\binom{Rm+N-2}{N-1}+3\binom{Rm+N-1}{N-1}+4(R-1)\binom{m+N-1}{N-1}$$
$$+\ 4\binom{m+N-2}{N-1}+2mR-6R$$

The computations saved through parallel processing for cases one and two are respectively:

$$4\binom{Rm+N-1}{N-1}+2Rm-7 \text{ and } 4\binom{Rm+N-2}{N-1}+2\binom{Rm+N-1}{N-1}+2mR-10$$

Table 8.4.2 shows examples of the savings.

8.5 Adaptive Convolution by Chain Algorithm

The previous section discussed the computational requirements of the Convolution by Chain algorithm. As may be seen, the computational costs vary according to N, R and

m. We will show later that when $m = 1$, the new algorithm has smaller computational requirements than the IDAC algorithm does in dealing with networks that have a few nodes and many chains. In some cases, further improvement may be achieved with $m > 1$. On the other hand, a multiple chain closed product form queueing network that has R chains may be converted in the following ways. Every chain that has $m > 1$ customers may be converted into r chains $2 \leq r \leq m$. The number of customers in each converted chain is greater or equal to one. All the chains converted from the same chain have identical routing and service time distributions. The total population of the r chains is equal to m. The joint queue length distributions are not affected by this conversion. Hence, given a multiple chain product form queueing network, an interesting problem is how to convert it into another network that has a different number of chains so that the Convolution by Chain algorithm works more efficiently.

After a careful study of the multiplication requirements, we suggest the following rules to ensure the efficiency.

1. Re-enumerate the chains in such a way that $K_i \geq K_j$ if $i < j$, $i, j = 1, ..., R$, i.e. a chain with a larger population will have a smaller chain number index.
2. After the re-enumeration,
   (a) if $N = 2$, and
      i. if $4 \leq R \leq 7$ and $K_r \leq 2$ for all r, convert all the chains into single customer chains.
      ii. else, the network is unaltered.
   (b) if $N = 3$, and
      i. if $K_r \leq 4$ for $r = 1, 2, 3$, convert all the chains except the first three into single customer chains.
      ii. if $K_r \leq 6$ for $r = 1, 2$, convert all the chains except the first two into single customer chains.
      iii. if $R \geq 7$ and $K_r \leq 2$ for all r, the network is unaltered.

iv. else, convert all the chains except the first one into single customer chains.

(c) if $N \geq 4$, and

i. if $K_1 \geq 5$, convert all the chains except the first one into single customer chains.

ii. if $K_1 < 5$, convert all the chains into single customer chains.

The Convolution by Chain Algorithm together with the above rules is called the Adaptive Convolution by Chain Algorithm(ACCAL). A network being converted in this way may be analyzed more efficiently by ACCAL than either the original network or a converted single customer chain network. The above rules may be easily verified through a similar way as we calculated the computational costs.

When the first chain is unchanged, $g^1(\mathbf{k}) = b(\mathbf{k}(1))$. $b(\mathbf{k}(1))$ is computed by Equation 8.7 with initial condition $b(K_r, 0....0) = w_{1r}^{K_r}/K_r!$. When a chain has a single customer, Equation 8.6 is applied. As a result of these rules, ACCAL converts networks in an adaptive way and uses either Equation 8.5 or Equation 8.6 for the analysis.

When only the joint distributions are required, the above rules are applied directly. When mean values of each chain are required, every chain is first converted into a chain with one less customers and another chain with a single customer. Then the above rules are applied. If mean values of each chain are required, after the results for the Rth chain are obtained, we borrow the following equation from the DAC [70]:

$$P^R(\mathbf{k}) = T_s \sum_{j=1}^{N} w_{js} k_j P^{R-1,s}(\mathbf{k} - \mathbf{e}_j)/\mu_j(k_j) \tag{8.20}$$

where $P^{R-1,s}(\mathbf{k} - \mathbf{e}_j)$ denotes the joint queue length distribution of the network with chain s removed. Therefore, T_s is computed as follows:

1. Assume $T_s = 1$.

2. By choosing:

$$\mathbf{k} = (r, 0, ...0), (r-1, 1, 0, ...0), \ ..., \ (0, ...0, r)$$

$P^{R-1,s}(\mathbf{k}-\mathbf{e}_j)$ can be derived from Equation 8.20.

3. $T_s = \sum_{\mathbf{k}\in S^{R-1,s}} P^{R-1,s}(\mathbf{k})$

In summary, ACCAL first converts the network into an equivalent one so that it may be analyzed more efficiently.

8.6 Further Improvement

At this stage, further improvement is still possible. The computation of $c(\mathbf{k})$ may be further simplified. First we define:

1. (from [62]) A Bag: a collection of elements over some domain. Multiple occurrences of elements are allowed and the order of the elements is not important. A bag consisting of elements $i_1, i_2, ..., i_N$ is written as $\beta\{i_1, i_2, ..., i_N\}$.
2. A function γ: $\gamma(J;\mathbf{I}) = \beta\{i_1, i_2, ..., i_J\}$ where $\mathbf{I} = (i_1, i_2, ..., i_N)$, $N \geq J$. In other words, the function $\gamma(J;\mathbf{I})$ creates a bag consisting of the first J elements of I.

Proposition 1

$$c(\mathbf{k}) = c(\mathbf{k}') \quad if \quad \gamma(J;\mathbf{k}) = \gamma(J;\mathbf{k}') \tag{8.21}$$

Proof: This proposition comes directly from the definition of $c(\mathbf{k})$.

Q.E.D.

As a result, the number of $c(\mathbf{k})$s we need to compute is equal to the number of distinct $\gamma(J,\mathbf{k})s$, $\mathbf{k} \in S^R$. According to [49], this number is equal to the sum of the coefficients of $x, x^2, ..., x^K$ in the expansion of the following expression:

$$\frac{1}{(1-x)(1-x^2)...(1-x^N)}$$

The savings in operations are obvious.

8.7 Dynamic Scaling

Like RECAL, IDAC and the convolution algorithms, ACCAL may suffer from the potential floating point underflow or overflow problem. As a result, a simple dynamic scaling procedure is employed. When the underflow or overflow is about to occur, simply multiply the present $g^r(\mathbf{k})$ by a scaling factor α. Then continue the algorithm. Note that the two normalization constants for obtaining T^*_{R+1} should be on the same scale.

8.8 Comparisons with other Algorithms

We compare the computational costs of the new ACCAL with the IDAC algorithm. For simplicity, we take $m = 1$ for ACCAL without the improvements discussed in Section 8.5 and Section 8.6. Actual costs of the algorithm should be less than the ones computed below.

The computational cost of ACCAL when mean values are required is shown in Equation 8.19. When $m = 1$, $b(\mathbf{k}(r))$ is no longer computed. Hence, the number of multiplications in sequential processing is:

$$\begin{aligned} & N\sum_{k=1}^{K-1}\binom{k+N-1}{N-1} + 2\{K-3+2[\binom{K+N-2}{N-1}-1]+\binom{K+N-1}{N-1}\}+1 \\ + \; & 2\binom{K+N-2}{N-1}+3\binom{K+N-1}{N-1}+1 \\ = \; & N\binom{N+K-1}{N}+6\binom{K+N-2}{N-1}+5\binom{K+N-1}{N-1}+2K-N-8 \end{aligned} \tag{8.22}$$

The computational cost difference between the IDAC algorithm and ACCAL in sequential processing is:

$$\binom{K+N-2}{N-1}[(2-4/K)(N+K-1)-6]-2K-2N+9$$

This value is negative(i.e. IDAC has less number of multiplications) only when:

1. $N \leq 3$ and $K \leq 4$.
2. $4 \leq N \leq 7$ and $K \leq 3$.

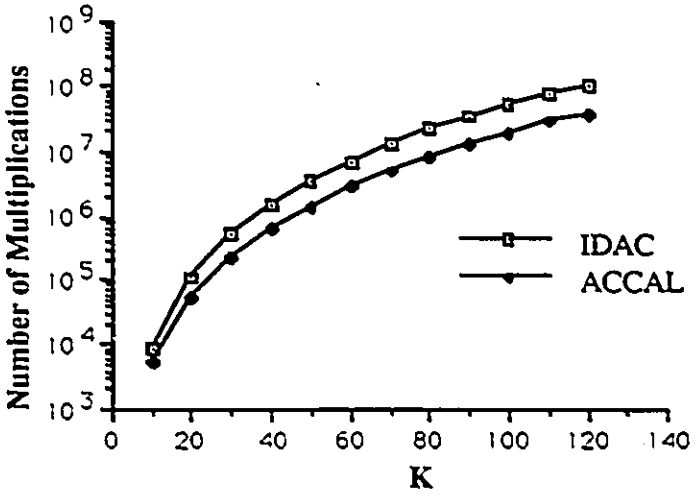

Figure 8.1: Time Requirements of ACCAL and the IDAC algorithm when N=4

3. $N \geq 8$ and $K \leq 2$.

However, the above cases can be analyzed more efficiently by the MVA and the convolution algorithms. Hence, When $m = 1$, ACCAL has fewer computational requirements than the IDAC does in dealing with networks that have a few nodes and many chains. Figures 8.8 and 8.8 compare the time requirements of ACCAL and the IDAC algorithm.

8.9 Discussions

Compared with the DAC and the IDAC algorithms, ACCAL avoids computing the joint queue length distributions, the marginal queue length distributions and the throughput every time the population is increased by one. Instead, it calculates the joint queue length distributions and the throughput only when the full population less one and the full population are presented. This makes ACCAL more efficient than the others.

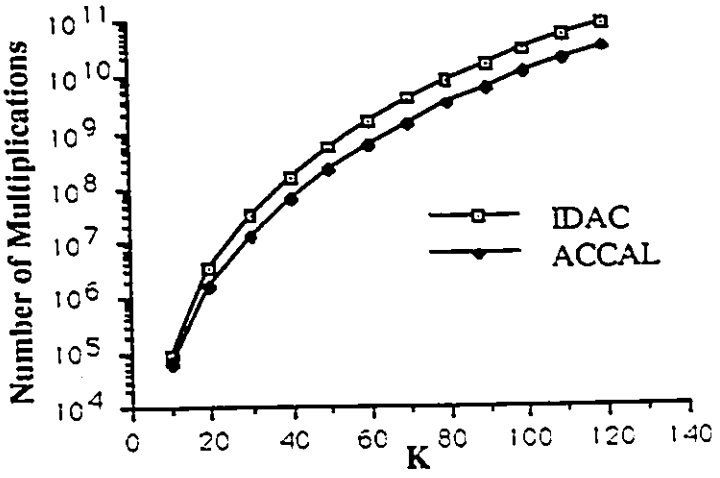

Figure 8.2: Time Requirements of the ACCAL and the IDAC algorithm when N=6

Chapter 9

Conclusions

9.1 Summary

The contributions of this thesis may be summarized as follows:

1. In Chapter 3, we have extended the boundary for the class of Local Balance Stochastic Petri Nets, proved that the Stochastic Petri Nets that falling into the new boundary have product form solutions and proposed a systematic way as well as a C language program for identifying this class of Stochastic Petri Nets. The program has been tested with many Stochastic Petri Net examples.
2. In Chapter 4, we have presented the Decomposition by Subnet method. The method shows how to decompose the analysis of Local Balance Stochastic Petri Nets into the analysis of subnets, the results of which are combined to obtain the results of the original system.
3. Also in Chapter 4, we have derived the Norton's theorem for Local Balance Stochastic Petri Nets by applying the Decomposition by Subnet method. We have shown how a subnet with marking dependent firing rates may be used to concisely, yet exactly, represent the original system. We also have shown that the Norton's theorem may be applied to the parametric analysis of Local Balance Stochastic Petri Nets.

4. In Chapter 5, we have developed an efficient algorithm for the parametric analysis of Stochastic Petri Nets in general by applying the concept of "Ideal Aggregates". It is much more efficient than analyzing the original system repeatedly every time the parameter of interest takes a new value.

5. In Chapter 7, we have presented the IDAC algorithm, which improves the efficiency of the DAC algorithm. IDAC also has the benefit with regard to reducing possible numerical errors.

6. In Chapter 8, we have proposed an Independent Decomposition and Aggregation method for the analysis of BCMP queueing networks. According to it, we have derived ACCAL, which is more efficient than the IDAC algorithm. Hence, it is the most efficient algorithm in dealing with networks consisting of many chains and a few nodes. The parallel processing and adaptive characteristics of ACCAL distinguish it from any existing algorithms and further improve its efficiency.

9.2 Suggestions for Further Research

Before we conclude this thesis, here are a few words about possible further research.

In Chapter 3, we have extended the boundary for Local Balance Stochastic Petri Nets. However, the new boundary is by no means a final boundary. There are many Stochastic Petri Nets that have product form solutions but may not pass the test procedure described in Chapter 3. For example, all the Local Balance Stochastic Petri Nets considered in Chapters 3 and 4 are marking independent (though the aggregated net is marking dependent). However, it is reasonable to conjecture that some marking dependent Stochastic Petri Nets may also have product form solutions. What kind of marking dependence is allowed? What are the effects on the product forms and Norton's theorem? These are certainly interesting problems and further exploration is desirable, because the existence of product form solutions simplifies the analysis significantly.

Our intention in Chapter 3 was to show that there is a systematic way of identifying

Local Balance Stochastic Petri Nets. The test procedure as well as the software program is by no means the most efficient one. More work may be done in improving the efficiency of the test procedure.

In Chapter 4, We have applied the decomposition by subnet method to derive the Norton's theory. By following a similar way of analysis, e.g. decomposition by CSS, a convolution algorithm may be derived for the efficient analysis of the Local Balance Stochastic Petri Nets.

The algorithm presented in Chapter 5 reduced the time requirement for parametric analysis. Time and space requirements may be further reduced by taking advantage of the sparsity of the associated Markov Chains.

For queueing networks, each of the existing algorithms is efficient in some domain. For instance, some are efficient in analyzing networks with many nodes and a few chains whereas others are efficient in analyzing networks with many chains and a few nodes. When both the number of nodes and the number of chains are large, none of the existing algorithms are efficient. However, the recursive nature of algorithms suggests that hybrid computational algorithms may be a good solution in this case. The possibility of the existence of hybrid algorithms was also pointed out in [14]. However, up to now, no hybrid algorithms of practical use have been derived.

Because of time limits, we are unable to do the work listed above. Nevertheless, they are certainly interesting topics for further study.

Appendix A

Local Balance Stochastic Petri Net Test Program List

```
/*
    1. The purpose of this program is to test if a safe SPN is a LBSPN.
    2. The SPN has marking independent firing rates only.
    3. A "-1" in an array signifies the end of the array.
*/

#include <stdio.h>
#include <stdlib.h>


#define NARC  1
#define MAX  100
#define MAXTRANS  100
#define MAXSUB  100
#define MAXMARK  1000
#define MAXPLACE  100
#define MAXCYCLE  20
#define TRUE  1
#define FAULT  0

struct transition
        {
        int input[MAXPLACE];
        int output[MAXPLACE];
        };

/*
&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&
&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&
*/
/* The following variables have to be specified according to the model */


#define NPLACE 10           /* Total number of places */
#define NTRANS 11           /* Total number of transitions */


    static int res[]={7,-1};      /* resource places that have an initial token */
    static int nres[]={1,8,-1};   /* non-resource palces that have an initial token */

    static struct transition transit[]=
        {
        {0},{0},          /* dummy */
        {1,7,-1},{2,7,-1},     /* { input places to transtion 1}, {output places to transition 1} */
        {2,7,-1},{3,-1},       /* { input places to transtion 2}, {output places to transition 2} */
        {3,-1},{4,-1},
        {4,-1},{1,7,-1},
        {1,-1},{5,-1},
        {5,-1},{6,-1},
        {6,-1},{1,-1},
        {1,-1},{6,-1},
        {7,8,-1},{7,9,-1},
```

```
        {7,9,-1},{10,-1},
        {10,-1},{7,8,-1}
        };

/* End of specification */
/*
&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&
&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&
*/


    struct tnode
        {
        int *mar;
        struct tnode *left;
        struct tnode *right;
        int numbmark;
        };
    struct tnode *root;
    struct tnode *talloc(void);
    struct tnode *addtree(struct tnode *p, int marking[]);

    int rg(int *m0, int *setrans, int *back, int *show, int sub, int test);
    int  findcycle(struct transition *trans, int *m0, int *cycle[]);
    int *nextmark(struct transition tname, int *oldmark);
    int enable(struct transition tname, int marking[]);
    int group( int *s[], int n, int *g[]);
    int *sqarraycat(int *s, int *t);
    int compare(int *m1, int *m2);
    int *arraycat(int *s, int *t);
    int setcomp(int *s, int *t);
    int insubnet(int i, int *s);
    int incomp(int *t, int *s);
    int *generatmark(int *s);
    int sortj(int k, int *m);
    int arraysize(int *s);
    int *markcopy(int *s);
    int *vecopy(int *s);
    int findcss();

    void treeprint(struct tnode *);
    void setzero(int *s, int j);
    void priv(int vextor[]);
    void putdorm(int *net);
    void prim(int mark[]);
    void migenerate();

    int  dormtrans[NTRANS+1];
    int  dormtoken[NPLACE+1];
    int  totmark=0;                /* total number of markings */
```

```
    int *m;
    int totnet= -1;                 /* total number  of subnets */
    int *mark[MAXMARK];
    int *cycles[MAXCYCLE];          /* array of transition cycles */

    int nsub;                   /* number of subnets-1 */
    int ncss;                   /* number of CSS -1 */
    int *css[MAX];                  /* list of CSSs */
    int *tsubnets[MAXSUB];          /* list of subnets */
    int *inisubm[MAX];          /* initial subnet markings */

    int  ifnew = FAULT;


main()
{
    int i,j,k,l,totrans=0,ncyc,*subnets[MAXSUB];
    int set[MAX],back,show,*m0;
    int n_mark, n_css, n_sub;

    setzero(dormtrans, NTRANS+1);
    for (i=0; i<=arraysize(nres)-1; i++)
        {
        m=generatmark(res);
        m[nres[i]]=1;

        /* generate cycles */

        ncyc=findcycle(transit,m,cycles);
        if(ncyc == -1)
            {
            printf("failure due to dead lock subnet");
            printf(" markings\n");
            return;
            }

        /* generate subnets */

        nsub=group(cycles, ncyc, subnets);
        for(k=0;k<=nsub;k++)
            {
            tsubnets[totnet+k+1]=vecopy(subnets[k]);
            putdorm(subnets[k]);
            }
        totnet += (nsub+1);
        }
    /* when there is only a single subnet */

    if(totnet<1)
        {
```

```
        printf("failure1\n");
        return;
        }

/* else */

printf("There are\t%d\tsubnets:\n",totnet+1);
for(i=0;i<=totnet;i++)
        priv(tsubnets[i]);
printf("\n");

/* when some transitions can not be included into any subnets */

for(k=0; k<=totnet; k++)
        totrans += arraysize(tsubnets[k]);
if(totrans != NTRANS)
        {
        printf("failure2\n");
        return;
        }

/* else, generate CSSs */

ncss=findcss();

/* generate initial subnet markings */

migenerate();

/* generate initial markingks */

m0=arraycat(res,nres);
mark[0]=generatmark(m0);
for(i=0; i<=NTRANS-1; i++)
set[i]=i+1;
set[NTRANS] = -1;

/* generate Reachability Graph  */

totmark=rg(mark[0],set,&back,&show, 3,0);

/* verify test conditions */

for(n_mark=0; n_mark<=totmark; n_mark++)
        {
        for(n_css=0; n_css<=ncss; n_css++)
                {
                if(arraysize(css[n_css])>1)
                        {
                        l=0;
                        while(*(css[n_css]+l) != -1)
                                {
```

```
                        n_sub= *(css[n_css]+l);
                        k=rg(mark[n_mark], tsubnets[n_sub], &back, &show, n_sub, 1);
                        if(back && show == 0)
                                {
                                printf("failure");
                                return;
                                }
                        l++;
                        }
                }
        }
    }
    printf("success\n");

}


int rg(int *m0, int *setrans, int *back, int *show, int sub, int test)
{

/* when test=0, generate Reachability Graph from m0. When test=1, do the test procedure for
marking m0 with subnet sub */

    int n, i, l, x, w, tot=0, live, *new, *m[MAXMARK];

    root=NULL;
    *back = *show = FAULT;
    m[0]=m0;
    root=addtree(root,m[0]);
    for(n=0;n<=tot;n++)
        {
        live=FAULT;
        for(l=0; setrans[l] != -1; l++)
            {
            x=setrans[l];
            if(enable(transit[x], m[n]))
                {
                live=TRUE;
                new=nextmark(transit[x],m[n]);
                root=addtree(root,new);
                if(ifnew)
                    {
                    tot++;
                    m[tot]=markcopy(new);
                    }
                if(!test)
                    mark[tot]=markcopy(new);
                if(test)
                    {
                    if(*back=FAULT && compare(new,m0)==0)
                        *back=TRUE;
                    if(*show=FAULT && incomp(inisubm[sub],new))
```

```
                                *show=TRUE;
                            }
                        }
                    }
                if(!test )
                    {
                    if(live==FAULT)
                        {
                        printf("deadlock marking\t");
                        prim(m[n]);
                        return FAULT;
                        }
                    }
                }
        return tot;
}


void migenerate()
{

/* generate initial subnet markings */

    int i,l,w;

    for(i=0;i<=totnet;i++)
        {
        inisubm[i]= (int *) calloc(MAX, sizeof(int));
        w=0;
        for(l=0; res[l] != -1; l++)
            {
            if(insubnet(res[l],tsubnets[i]))
            *(inisubm[i]+w++)=res[l];
            *(inisubm[i]+w) = -1;
            }
        for(l=0; nres[l] != -1; l++)
            {
            if(insubnet(nres[l],tsubnets[i]))
            *(inisubm[i]+w++)=nres[l];
            *(inisubm[i]+w) = -1;
            }
        }
}


int findcss()
{

/* generate CSSs */

    int i,j,l, icss, respl, nnet, *resnet[MAXSUB];
```

```
    for(respl=0; res[respl] != -1; respl++)
        {
        resnet[respl]=(int *) calloc(MAXSUB, sizeof(int));
        l=0;
        for(nnet=0; nnet<=totnet; nnet++)
            {
            if( insubnet(res[respl], tsubnets[nnet]))
                {
                *(resnet[respl]+l)=nnet;
                *(resnet[respl]+(++l)) = -1;
                }
            }
        }
    icss=group(resnet,arraysize(res)-1,css);
    return icss;
}


int *vecopy(int *s)
{

/* copy s to p, the last element in s must be -1 */

    int i, *p;

    p=(int *) calloc(arraysize(s)+1,sizeof(int));
    if(p!=NULL)
        {
        for(i=0;i<=arraysize(s);i++)
            p[i]=s[i];
        }
    return p;
}


void setzero(int *s, int j)
{

/* set  elements 0 to j  into zero */

    int i;

    for(i=0; i<=j; i++)
        s[i] = 0;
        s[j+1] = -1;
}


int insubnet(int i, int *s)
{

/* return 1, if place i is in subnet s, else 0 */
```

```
    int j;

    struct transition *p;
    for(j=0; s[j] != -1; j++)
          {
          p = &transit[s[j]];
          if(sortj(i,p->input) || sortj(i,p->output))
                return 1;
          }
     return 0;
}


int  findcycle(struct transition *trans, int *m0, int *cycle[MAXCYCLE])
{

/* find out cycles starting from marking m0. The index of the last cycle is returned */

     int isub, cset[MAXTRANS],itrans,icyc;
     int i,j,k,l,live,*new;


     icyc = -1;
     for(itrans=1;itrans<=NTRANS ;itrans++)
          {
          if(!dormtrans[itrans] && enable(trans[itrans],m0))
                {
                icyc++;
                l=0;
                cycle[icyc]=(int *) calloc(NPLACE+1, sizeof(int));
                *(cycle[icyc]+l++)=itrans;
                *(cycle[icyc]+l) = -1;
                mark[0]=nextmark(trans[itrans],m0);
                totmark=0;
                j=0;
                while(j<=totmark)
                      {
                      live=FAULT;
                      for(k=1;k<=NTRANS;k++)
                            {
                            if(!dormtrans[k] &&
                            enable(trans[k],mark[j]))
                                  {
                                  live=TRUE;
                                  if(!sortj(k,cycle[icyc]))
                                        {
                                        *(cycle[icyc]+l) = k;
                                        *(cycle[icyc]+(++l))= -1;
                                        }
                                  new=nextmark(trans[k],
                                  mark[j]);
```

```
                                    if(compare(new,m0)!=0)
                                        {
                                        root=addtree(root,
                                        new);
                                        if(ifnew)
                                            {
                                            totmark++;
                                            mark[totmark]=
                                            markcopy(new);
                                            }
                                        }
                                    }
                                }
                            if(live == FAULT)
                                return 0;
                            j++;
                            }
                        }
                    }
                return icyc;
}


int group( int *s[], int n, int *g[])
{

/* divide the n vectors (last index is n-1) of s into groups, a vector is in the
 group, if and only if, it has at least one common elements with one of the other
 vectors in the same group. The index of the last group of g is retuned */

int i,j,ig,cset[MAX], *mid;

setzero(cset,n);
ig = -1;
for (i=0; i<=n; i++)
    {
    if(cset[i]==0)
        {
        ig++;
        cset[i]=1;
        g[ig]=vecopy(s[i]);
search:     for(j=1; j<=n; j++)
            {
            if(cset[j]==0 && setcomp(g[ig],s[j]))
                {
                cset[j]=1;
                mid=sqarraycat(g[ig],s[j]);
                g[ig]=vecopy(mid);
                goto search;
                }
            }
        }
```

```
    }
return ig;
}


int *markcopy(int *s)
{

/* copy marking s to p and return p */

    int i, *p;

    p = (int *) calloc(NPLACE+1, sizeof(int) );
    if (p != NULL)
          {
          for(i = 0; i<=NPLACE; i++)
          p[i]=s[i];
          }
    return p;
}


int *nextmark(struct transition trname, int *oldmark)
{

/* generate the next marking from "oldmark" by firing transition "trname" */

     int i,*p;
     struct transition *pt;

     pt = &trname;
     p=markcopy(oldmark);
     for(i=0; i<=NPLACE; i++)
          {
          if(pt -> input[i]!= -1) p[pt->input[i]]-=1;
          if(pt -> output[i]!= -1) p[pt -> output[i]]+=1;
          }
     return p;
     }


struct tnode *addtree(struct tnode *p, int marking[])
{

/* generate binary tree, the root of which is p */

     int com;

     ifnew = FAULT;
     if(p == NULL)
          {
          ifnew = TRUE;
```

```
            p = talloc();
            p->mar = markcopy(marking);
            p->numbmark=totmark+1;
            p->left = p->right = NULL;
            }
    else if((com=compare(marking, p->mar)) < 0)
            p->left=addtree(p->left, marking);
    else if(com>0)
            p->right = addtree(p->right, marking);
    else
            ;
    return p;
}


struct tnode *talloc(void)
{

/* space allocation */

     return(struct tnode *) malloc(sizeof(struct tnode));
}


int enable(struct transition tname, int marking[])
{

/* return 1, if transition "tname" is enabled in the marking,  else return 0 */

      int i;
      struct transition *pt;

      i=0;
      pt = &tname;
      if(pt -> input[i]== -1) printf( "error, no inputs to trans");
      else
            {
            while(pt -> input[i]!= -1)
                  {
                  if(marking[pt -> input[i]]<NARC) return 0;
                  else i++;
                  }
            return 1;
            }
       }


  void prim(int mark[])
  {

  /* print markings*/
```

```
    int i;

    for(i=1; i<=NPLACE; i++)
    if(mark[i]!=0)
    printf("%d ",i);
    printf("\n");
    }


int compare(int *m1, int *m2)
{

/* m1 > m2 , return -1;
   m1 < m2 , return  1;
   m1 = m2 , return  0; */

     int i;

     for(i=0; i<=NPLACE; i++)
           {
           switch(m1[i]-m2[i])
                 {
                 case  1: return -1;
                 case  -1: return 1;
                 case  0: ;
                 }
           }
      return 0;
  }


  void treeprint(struct tnode *p)
  {

  /* print binary tree, whose root is p */

      if(p!=NULL)
            {
            treeprint(p->left);
            prim(p->mar);
            treeprint(p->right);
            }
   }


   int sortj(int k, int *m)
   {

   /* Return 1 if k is in m, else return 0. The last element in m must be "-1". */

        int j;
```

```
    for(j=0;j<=arraysize(m)-1;j++)
        if(m[j]==k) return 1;
    return 0;
}

int *arraycat(int *s, int *t)
{

/* append t after s */

    int i,j,*m;

    m=(int *) calloc(arraysize(s)+arraysize(t)+2, sizeof(int));
    j=0;
    for(i=0;s[i] != -1;i++)
        m[j++]=s[i];
    for(i=0;t[i] != -1;i++)
        m[j++]=t[i];
    m[i]= -1;
    return m;
}

int arraysize(int *s)
{

/* compute the number of elements in an array */

    int n;

    for (n=0; !(s[n]<0); n++)
    ;
    return n;
}

int incomp(int *t, int *s)
{

/* return 1 if all the elements in t are also in s, else 0 */

    int i;

    for(i=0; t[i] != -1; i++)
        {
        if(!sortj(t[i],s))
            return 0;
        }
    return 1;
}
```

```
int setcomp(int *s, int *t)
{

/* return 1 if s and t has at least one element in common, 0 else */

    int i;

    for(i=0; s[i]!= -1; i++)
          {
          if(sortj(s[i],t))
          return 1;
          }
    return 0;
}


int *generatmark(int *s)
{

/* generate marking p specified by s, i.e. p[s[i]]=1, and return p */

     int i, *p;

     p=(int *) calloc(NPLACE+1, sizeof(int));
     for(i=0;i<=arraysize(s);i++)
          p[i]=0;
     for(i=0;i<=arraysize(s)-1;i++)
          p[s[i]]=1;
     return p;
}


void priv(int vector[])
{

/* print out array "vector" */

     int j;

     for(j=0; vector[j] != -1; j++)
          printf("%d\t",vector[j]);
     printf("\n");
}


int  *sqarraycat(int *s, int *t)
{

/* append the elements in t, which is not in s, after s. The value returned
should not be assigned to a static vector */
```

```
    int i,j,*m;

    m=(int *) calloc(arraysize(s)+arraysize(t)+2, sizeof(int));
    j=0;
    for(i=0;s[i] != -1;i++)
          m[j++]=s[i];
    for(i=0;t[i] != -1;i++)
          {
          if(!sortj(t[i],s))
                m[j++]=t[i];
          }
    m[j]= -1;
    return m;
}


void putdorm(int *net)
{

/* put trnsitions specified by net into dormant */

     int j;

     j=0;
     while(net[j]!= -1)
          {
          dormtrans[net[j]]=1;
          j++;
          }
 }
```

Bibliography

[1] H. H. Ammar, "Time Scale Decomposition of a Class of Generalized Stochastic Petri Net Models" *IEEE Transactions on Software Engineering*, vol. 15, no. 6, June 1989.

[2] H. H. Ammar, Y.F. Huang, R.W. Liu "Hierarchical Models for Systems Reliability Maintainability and Availability" *IEEE Transactions on Circuits System*, vol. 34, no. 6, June 1987.

[3] P. Azema, G. Juanole, E. Sanchis, M. Montbernard, "Specification and Verification of Distributed Systems Using PROLOG Interpreted Petri Nets" *Proceedings of the 7th International Conference on Software Engineering*, Orlando, USA, 1984.

[4] F. Baskett, K.M. Chandy, R.R Muntz, F. Palacios, "Open, Closed and Mixed Networks of Queues with Different Classes of Customers" *Journal of the ACM*, 22, 1975.

[5] G. Berthelot, R. Terrat, "Petri Nets Theory for the Correctness of Protocols" *IEEE Transactions on Communications*, vol. 30, no. 12, Dec. 1982.

[6] G. Bochmann, J. Gecsel, "A Unified Method for the Specification and Verification of Protocols" *Information Processing 77*, North Holland, Amsterdam, 1977.

[7] S.C. Bruell, G. Balbo, *Computational Algorithms for Closed Queueing Networks*, North Holland, New York, 1980.

[8] P. Buchholz, "Numerical Solution Methods Based on Structured Descriptions of Markovian Models" *Proceedings of the 5th International Conference on Modelling Techniques and Tools for Computer Performance Evaluation*, Torino, Italy, Feb. 1991.

[9] J.P. Buzen, "Computational Algorithms for Closed Queueing Networks with Exponential Servers" *Communications of the ACM*, 16, 1973.

[10] J.P. Buzen, "Queueing Network Models of Multiprogramming" Ph.D Dissertation, Div. Eng. Appl. Phys., Harvard University, Cambridge, Massachusetts, 1971.

[11] W. L. Cao, W. J. Stewart, "Iterative Aggregation-Disaggregation Techniques for Nearly Uncoupled Markov Chains" *Journal of ACM*, 32, 1985.

[12] G. Ciardo, K. S. Trivedi, "Solution of Large GSPN Models" *Proceedings of the 1st. International Conference on the Numerical Solution of Markov Chains*, Raleigh, North Carolina, Jan. 1990.

[13] G. Chiola, C. Dutheillet, G. Franceschinis, S. Haddad, "On Well-formed Colored Nets and Their Symbolic Reachability Graph" *Proceedings of the 11th International Conference on Application and Theory of Petri Nets*, Paris, France, June, 1990.

[14] A.E. Conway, N.D. Georganas, *Queueing Networks: Exact Computational Algorithms - a Unified Theory Based on Decomposition and Aggregation*, MIT Press, Cambridge,1989.

[15] A.E. Conway, N.D. Georganas "RECAL: A New Efficient Algorithm for the Exact Analysis of Multiple Chain Closed Queueing Networks" *Journal of the ACM*, 33, 1986.

[16] A.E. Conway, N.D. Georganas "A New Method for Computing the Normalization Constant of Multiple Chain Queueing Networks, *INFOR*, 24, 3, 1986.

[17] A.E. Conway, E. de Souza e Silva, S.S. Lavenberg "Mean Value Analysis by Chain of Product Form Queueing Networks" *IEEE Transactions on Computers*, vol. 38, no. 3, March, 1989.

[18] P.J. Courtois, "On the Near-Complete Decomposability of Networks of Queues and of Stochastic Models of Multiprogramming Computing Systems" Sci. Rep., CMU-CS-72, 111, Carnegie-Mellon University, Pittsburgh, Pennsylvania, 1972.

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

[20] P.J. Courtois, "Exact Aggregation in Queueing Networks" *Proc. First Meeting AFCET-SMF on Applied Mathematics*, 1, Ecole Polytechnique, Palaiseau, France, 1978.

[21] P.J. Courtois, "Error Minimization in Decomposable Stochastic Models" *Applied Probability - Computer Science: the Interface*,1, 1982.

[22] P.J. Courtois, "On time and Space Decomposition of Complex Structures", *Communication of the ACM*. June, 1985.

[23] M. Diaz, "Modelling and Analysis of Communication and Cooperation Protocols using Petri Net Based Models" *Computer Networks*, vol. 6, Dec. 1982.

[24] M. Diaz, P. Azema, "Petri Net Based Models for the Specification and Validation of Protocols" *Lecture Notes in Computer Science*, vol. 188. Springer Verlag, New York, 1985.

[25] C. Dutheillet, S. Haddad, "Aggregation and Disaggregation of States in Colored Stochastic Petri Nets: Application to a Multiprocessor Architecture" *Proceedings of the 3rd International Workshop on Petri Nets and Performance Models*, IEEE-CS Press, Kyoto, Japan, Dec. 1989.

[26] G. Florin, S. Natkin, "One Place Unbounded Stochastic Petri Nets: Ergodicity Criteria and Steady State Solution" *Journal of System Software*, vol. 1, no. 2, 1986.

[27] G. Florin, S. Natkin, "Generalization of Queueing Network Product Form Solutions to Stochastic Petri Nets" *IEEE Transactions on Software Engineering*, vol. 17, no. 2, Feb. 1991.

[28] E. Gelenbe, I. Mitrani, *Analysis and Synthesis of Computer Systems*, Academic Press, 1980.

[29] J. Giglmayr, "Analysis of Stochastic Petri Nets by the Concept of Near-Complete Decomposability", *Transactions of the Tenth Prague Conference on Information Theory, Statistical Decision Functions, Random Processes*, Prague,Czechoslovakia, 1988.

[30] J. Giglmayr, "Analysis of Stochastic Petri Nets by the Decomposition of the Transition Rate Matrix" *ntzArchiv*, May, 1987.

[31] E. Gressier, "A Stochastic Petri Net Model for Ethernet" *International Workshop on Timed Petri Nets*, Torino, Italy, July, 1985.

[32] D.L. Isaacson, R.W. Madsen, *Markov Chains: Theory and Applications*, New York, Wiley, 1976.

[33] J.R. Jackson, "Networks of Waiting Lines" *Operations Research*, 5, 1957.

[34] K. Jensen, "Colored Petri Nets and the Invariant Method" *Theoretical Computer Science*, 14, 1981.

[35] F.P. Kelly, *Reversibility and Stochastic Networks*, Wiley, New York, 1980

[36] S.S Lam, Y.L. Lien "A Tree Convolution Algorithm for the solution of Queueing Networks" *Communication of the ACM*, 26, 1983.

[37] S.S. Lavenberg, M. Reiser "Stationary State Probabilities at Arrival Instants for Closed Queueing Networks with Multiple Types of Customers" *Journal of Applied Probability*, 17, 1980.

[38] S.S Lavenberg (ED) *Computer Performance Modeling Handbook*, Academic Press, New York, 1983.

[39] A.A. Lazar, T.G. Robertazzi, "The Algebraic and Geometric Structure of Markovian Petri Network Lattices", *Proceedings of the Twenty-fourth Annual Allerton Conference on Communication, Control and Computing*, University of Illinois, Urbana-Champaign ILL., 1986.

[40] A.A. Lazar, T.G. Robertazzi, "Markovian Petri Net Protocols with Product Form Solution" *IEEE INFOCOM'87*, San Francisco, CA, 1987.

[41] A.A. Lazar, T.G. Robertazzi, "Markovian Petri Net Protocols with Product Form Solution" *Performance Evaluation*. vol. 12, no. 1, January, 1991.

[42] M. Li, N.D. Georganas, "Colored Generalized Stochastic Petri Nets for Integrated System Protocol Performance Modelling" *Computer Communications*, vol. 13, no.7, Sept. 1990.

[43] M. Li, N.D. Georganas, "Exact Parametric Analysis of Stochastic Petri Nets " *IEEE Transactions on Computers*,1992(in press).

[44] M. Li, N.D. Georganas, "Norton' Theorem in the Analysis of a Class of Stochastic Petri Nets" Submitted for Publication, 1990.

[45] M. Li, N.D. Georganas, "Parametric Analysis of Stochastic Petri Nets " *Fifth International Conference on Modelling and Tools for Computer Performance Evaluation*, Torino, Italy, Feb. 1991.

[46] M. Li, N.D. Georganas, "IDAC-Improved Distribution Analysis by Chain Algorithm" submitted for publication, 1991.

[47] M. Li, N.D. Georganas, "ACCAL- an Efficient Adaptive Convolution by Chain Algorithm" Submitted for Publication, 1991.

[48] C. Lin, C. Marinescu, "Stochastic High Level Petri Nets and Applications" *IEEE Transactions on Computers*, no. 7, July, 1988.

[49] C. L. Liu, *Introduction to Combinatorial Mathematics*, Computer Science Series, McGraw-Hill Book Company, 1968.

[50] M.A. Marsan, G.Balbo, G. Conte, F. Gregoretti "Modelling Bus Contention and Memory Interference in a Multiprocessor System" *IEEE Transactions on Computers*, vol. 32, no. 1, 1983.

[51] M.A. Marsan, G.Balbo, G. Conte,S. Donatelli, "On the Product Form Solution of a class of Multiple-Bus Multiprocessor System Models" *Journal of Systems and Software*, vol. 1, no. 2, 1986.

[52] M.A. Marsan, G.Balbo, G. Conte, "A Class of Generalized Stochastic Petri Nets for the Performance Evaluation of Multiprocessor Systems" *ACM Transactions on Computer Systems*, May 1984.

[53] M.A. Marsan, G.Balbo, G. Conte, *Performance Models of Multiprocessor Systems*, The MIT Press, 1986.

[54] J. Martinez, H. Alla, M. Silva, "Petri Nets for the Specification of Flexible Manufacturing Systems" *Modelling and Design of Flexible Manufacturing Systems. Elsevier Science*, New York, 1986.

[55] M.K. Molloy, "Performance Analysis Using Stochastic Petri Nets" *IEEE Transactions on Computers*, vol. 31, no. 9, September 1982.

[56] M.K. Molloy, *Fundamentals of Performance Modelling*, Macmillan Publishing Co. New York, 1989.

[57] T. Murata, B. Shenker, S. M. Shatz, "Detection of Ada Static Deadlocks Using Petri Net Invariants" *IEEE Transactions on Software Engineering*, vol. 15, no. 3, Mar. 1989.

[58] T. Murata, "Petri Nets: Properties, Analysis and Applications" *Proceedings of the IEEE*, vol. 77, no. 4, Apr. 1989.

[59] S. Natkin, "Les Reseaux de Petri Stochastiques et leur Application a l'Evaluation des Systems Informatiques" These de Docteur, CNAM, Paris, France, 1980.

[60] M. F. Neut, *Matrix-Geometric Solutions in Stochastic Models*, John Hopkins University Press, 1981.

[61] B. Noble, J.W. Daniel, *Applied Linear Algebra*, Prentice-Hall Inc,1984.

[62] J. Peterson *Petri Net theory and the Modelling of Systems*, Prentice Hall, Inc. 1981.

[63] C.A. Petri, "Communication with Automata" Ph. D thesis, Technical Report RADC-TR, New York, 1966.

[64] M. Reiser, "Queueing Network Analysis of Computer Communication Networks with Window Flow Control" *IEEE Transactions on Communications*, vol. 27, 1979

[65] M. Reiser, H. Kobayashi, "Queueing Networks with Multiple Closed Chains: Theory and Computational Algorithms" *IBM Journal on Resource and Development*, 19, 1975.

[66] M. Reiser, S.S.Lavenberg, "Mean Value Analysis of Closed Multichain Queueing Networks" *Journal of the ACM*, 27, 1980.

[67] M. Reiser, H. Kobayashi, "Recursive Algorithms for General Queueing Networks with Exponential Srvers", IBM Research Report, RC 4254, Yorktown Heights, New York, 1973.

[68] T. G. Robertazzi, *Computer Networks and Systems: Queueing Theory and Performance Evaluation*, Springer Verlag, New York, 1990.

[69] H.A. Simon, A. Ando, "Aggregation of Variables in Dynamic Systems" *Econometrica*, 29. 1961.

[70] E. de Souza e Silva, S.S. Lavenberg, "Calculating Joint Queue Length Distributions in Product Form Queueing Networks" *Journal of the ACM*, 36, 1989.

[71] J. Spragins, "Analytical Queueing Models: Guest Editor's Introduction" *IEEE Computer*, 13, 4, 1980.

[72] H. Vantilborgh, "Exact Aggregation in Exponential Queueing Networks" *Journal of the ACM*, 25, 1978.

[73] I. Y. Wang, T. G. Robertazzi, "Service Stage Petri Net Protocols with Product Form Solution" *Proceedings of the 1989 ACM Sigmetrics and Performance'89 International Conference on Measurement and Modeling of Computer Systems*, Berkeley, CA, May, 1989.

[74] R.L Zarling, "Numerical Solution of Nearly Decomposable Queueing Networks" Ph. D. Dissertation, Department of Computer Science, North Caroüna University, Chapel Hill, 1979.

[75] W. M. Zuberek, "Timed Petri Nets and Preliminary Performance Evaluation" *Proceedings of the 7th IEEE Annual Symposium on Computer Architecture*, 1980.