MODELS OF SYNCHRONOUS PRODUCTION LINES WITH NO INTERMEDIATE BUFFERS

A THESIS SUBMITTED TO
THE GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES
OF
MIDDLE EAST TECHNICAL UNIVERSITY

BY

HANDE ÇETİNAY

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR
THE DEGREE OF MASTER OF SCIENCE
IN
INDUSTRIAL ENGINEERING

JULY 2010

Approval of the thesis:

MODELS OF SYNCHRONOUS PRODUCTION LINES WITH NO INTERMEDIATE BUFFERS

submitted by **HANDE ÇETİNAY** in partial fulfillment of the requirements for the degree of **Master of Science in Industrial Engineering Department, Middle East Technical University** by,

Prof. Dr. Canan Özgen _____
Dean, Graduate School of **Natural and Applied Sciences**

Prof. Dr. Nur Evin Özdemirel _____
Head of Department, **Industrial Engineering**

Prof. Dr. Sinan Kayalıgil _____
Supervisor, **Industrial Engineering Dept., METU**

Prof. Dr. Sencer Yeralan _____
Co-Supervisor, **Agricultural and Biological Engineering Dept., University of Florida, USA**

Examining Committee Members:

Assist. Prof. Dr. Pelin Bayındır _____
Industrial Engineering Dept., METU

Prof. Dr. Sinan Kayalıgil _____
Industrial Engineering Dept., METU

Prof. Dr. Sencer Yeralan _____
Agricultural and Biological Engineering Dept., University of Florida, USA

Assist. Prof. Dr. İsmail Serdar Bakal _____
Industrial Engineering Dept., METU

Prof. Dr. Ülkü Gürler _____
Industrial Engineering Dept., Bilkent University

Date: 21/07/2010

I hereby declare that all information in this document has been obtained and presented in accordance with academic rules and ethical conduct. I also declare that, as required by these rules and conduct, I have fully cited and referenced all material and results that are not original to this work.

Name, Last name : Hande, Çetinay

Signature :

ABSTRACT

MODELS OF SYNCHRONOUS PRODUCTION LINES WITH NO INTERMEDIATE BUFFERS

Çetinay, Hande

M.Sc., Department of Industrial Engineering

Supervisor: Prof. Dr. Sinan Kayalıgil

Co-Supervisor: Prof. Dr. Sencer Yeralan

June 2010, 91 pages

Production lines with unreliable machines have received a great amount of attention in the literature. Especially, two-station systems have mostly been studied because such systems are easier to handle when compared to the longer lines. In literature, longer lines are usually evaluated by a decomposition algorithm, whereby the long line is partitioned into chunks of two-station lines. Decomposition algorithms require intermediate buffer storages of capacity at least two or three. The trends in modern manufacturing practices, on the other hand, such as the Toyota Production System, dictate that intermediate storages be eliminated. Our work studies multi-station lines with no intermediate storage. We develop software to automate the generation of transition probability matrices to allow the analysis of system behavior.

The algorithm allows the use of software packages to handle computations and to solve for exact solutions. Long-run behavior is obtained via the algorithm developed in the computational environment MATLAB. The purpose is to analyze the system performance measures such as starvation and blockage times of stations, production rate and work-in-process.

In addition, the production rate and the work-in-process measures over failure and repair probabilities are curve-fit to establish simple and useful empirical formulas for lines consisting three, four and five identical stations. Numerical analyses show that the proposed algorithm is effective for exact solutions and the suggested formulas are valid for approximate solutions.

Keywords: Multi-Station Production Lines, Discrete-Time Markov Chains, Unreliable Machines, Exact Solutions, No Intermediate Buffers

ÖZ

ARA STOKSUZ EŞZAMANLI ÜRETİM HATTI MODELLERİ

Çetinay, Hande

Yüksek Lisans, Endüstri Mühendisliği Bölümü
Tez Yöneticisi: Prof. Dr. Sinan Kayalıgil
Ortak Tez Yöneticisi: Prof. Dr. Sencer Yeralan

Temmuz 2010, 91 sayfa

Bozulabilir makinalardan oluşan üretim hatları literatürde geniş yer bulmuştur. Özellikle iki makinalı hatlar, model ve analiz kolaylığı nedeniyle çok makinalı hatlara kıyasla daha çok çalışılmıştır. Literatürde, uzun hatlar, ayrıştırma methodu ile, iki makinalı hatlara bölünerek incelenmiştir. Ayrıştırma methodu, makinalar arası en az iki veya üç ara stok koşulu getirmektedir. Fakat, Toyota üretim sistemi gibi günümüz modern üretim hatları, makinalar arası ara stokları kaldırmak istemektedir. Çalışmamız, çok makinalı ve ara stoksuz üretim sistemlerini konu almaktadır. Sistem davranış analizleri için, sistem karakteristiklerini belirleyen, sistem geçiş olasılıkları matrisini otomatik olarak yaratan algoritmalar geliştirilmiştir.

Geliştirilen algoritmalar, yazılım paketleri kullanıma uygun ve kesin sonuç veren çalışmalardır. MATLAB yazılım ortamında geliştirilen program sonucu durağan durum olasılıkları hesaplanmıştır. Burada amaç, makinaların boş kalma ve tıkanma oranları, sistemin üretim hızı ve hattaki parça sayısı gibi performans ölçütlerini analiz etmektir. Ek olarak, üç, dört ve beş özdeş makinalı sistemler için, durağan durum olasılıkları hesaplanarak sistem parametreleri ve sistem performansı arasındaki ilişki incelenmiştir.

Üretim hızı ile üretimdeki parça sayısı kriterleri icin data analizi yapılarak geçerli formüller elde edilmiştir. Analiz sonuçları, geliştirilen algoritmaların kesin sonuçlar için ve de önerilen formüllerin ise yaklaşık sonuçlar için geçerli ve kullanılabilir olduğunu göstermiştir.

Anahtar kelimeler: Çok İstasyonlu Üretim Hatları, Kesikli Zaman Markov Modeli, Bozulabilir Makinalar, Kesin Çözümler, Sıfır Ara Stok

To My Family, and

To my Robert…

ACKNOWLEDGEMENTS

First of all, I would like to express my deepest gratitude to all people at Industrial Engineering Department, METU, with whom I spent my years of undergraduate and graduate studies.

I would like to thank Prof. Dr. Sencer Yeralan, who I met in sports center, and thereafter with his great guidance and support both academically and personally, I completed my graduate studies.

I am grateful to Prof. Dr. Sinan Kayalıgil due to his guidance, time and advices for the research. Moreover, I would like to thank jury members for their valuable contributions.

Special thanks to Robert De Korte, with his day and night support in every part of my life including my master thesis.

In addition to my genius sister, Hale Çetinay, who initiated the logic with her brilliant advices, I would like to express my enduring love to my parents, Nurhan and İbrahim Çetinay, who are always supportive, loving and caring to me in every possible way in life.

Finally, I would like to thank TÜBİTAK for providing a qualified and reliable environment for my graduate study with their funding.

TABLE OF CONTENTS

LIST OF TABLES

TABLES

LIST OF FIGURES

FIGURES

CHAPTER 1

INTRODUCTION

1.1 System Description

The main reason for line inefficiency is the unreliable nature of stations. The starvation and blocking conditions on stations are idle times, which are defined and studied by many researched such as Muth and Yeralan (1981). When one machine in the flow fails then the rest of the line is prone to fail, especially if there is no intermediate buffer present. A failure may cause the preceding machine to be blocked, while at the same time, the downstream machines may be starved because there is no upstream input available. Analyzing the system behavior in manufacturing environments is crucial for production efficiency. Furthermore, the production rate and work-in-process are the main performance measures of the production lines.

A part of manufacturing systems that employ long lines typically contain several automated workstations tightly coupled without intermediate buffers. This is the case for many mass production plants like in the automobile manufacturing, where three to five assembly stations are combined into workstations.

The common approach to model tightly coupled lines is to consider each workstation as an individual machine. The characteristics of these composite machines need to be derived from the characteristics of the individual stations that comprise the workstation. This study evolves with the purpose of developing the composite workstation characteristics from the individual station characteristics. Note that the composite workstation will have the same processing time as the other stations in the line.

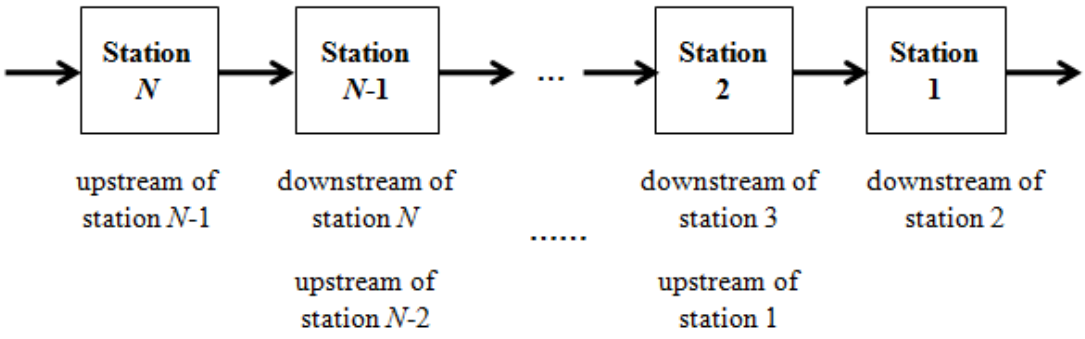

Figure 1.1 *N*-Station Production Line

The system under study includes a linear network of machines without intermediate buffer, as illustrated in Figure 1.1. Work pieces enter the system starting with station *N*. The system assumes infinite supply of raw materials; in other words, station *N* is never starved. An operation is performed on work piece by a single machine at each station. The system assumes infinite storage possibilities for the finished products; henceforth, station 1 is never blocked.

The aim is to foresee the system behavior of the production lines with no intermediate buffers. Regarding performance measures, the long run behavior of the system is important to describe the efficiency and effectiveness of the production lines. In addition, it is also important to gain insights about the production lines when the total number of stations in the line increased. This study enables developing transition probability matrices for multi-station production lines in order to describe the system behavior. Steady state probabilities are calculated from transition probability matrices are used to identify various performance measures. The main performance measures for the production lines are defined as; starvation and blockage probabilities, expected work-in-process *(WIP),* and expected production rate of the system. Particularly, it is essential to analyze the expected production rate and expected work-in-process as a function of line characteristics because these measures are the most valid and effective system behavior indicators.

1.2 Related Literature

Production lines and queuing systems have received much attention in the literature. To begin with; production rate of lines with and without intermediate buffer has been studied starting from the mid-fifties. The studies of Muth (1979) and Schick and Gershwin (1978) among many others are important studies conducted for modeling production lines.

In the literature, analytical models for multi-station production lines, which are the focus of this research, are classified in two main categories; modeling by continuous-time and discrete-time Markov models. Continuous-time models are favored in the process industry where no identification of the individual units exists. On the other hand, discrete models treat individual parts as successive states of the line, which is also the case in this study. Gershwin and Schick (1983) is an example of discrete modeling. Studies conducted by Yeralan *et al.* (1986), Yeralan and Tan (1997) provide examples of continuous-time models.

Moreover, operation-dependent failures have been studied in many studies, that is, machines in line can only fail when they are processing a work piece. Idle machines are not subject to failure as assumed by Gershwin (1987). Some studies also concentrated on time-dependent failures of machines that machines are subject to failure only due to aging such as Tan (1997).

A great deal of interest focused on the stochastic modeling of production systems. Most of the studies are intended to determine the performance measures, mainly the expected production rate and the expected buffer of the manufacturing lines. Recently, closed loop production lines with finite number of pallets and multiple failure modes for unreliable stations have received attention. Examples are Maggi *et al.* (2009) and Tolio *et al.* (2002). Table 1.1 summarizes the literature survey that was conducted with regard to the modeling of production lines. Mainly discrete-time models are the focus in the available literature.

Table 1.1 Literature Survey

	Model	**Number of Stations**	**Buffer Size**	**Line Property**	**Failure and Repair**	**Additional Property**	**Method of Analysis**
Maggi *et al.* (2009)	Discrete-time	3 station (2 station subsystems)	Fixed number of pallets	Synchronous	Geometric	Multiple Failure Modes, Closed Loop	Approximate Decomposition
Tolio *et al.* (2002)	Discrete-time	2 station	$N \geq 3$	Synchronous	Geometric	Multiple Failure Modes	Exact Solution
Tan (1997)	Discrete-time	Multi-station	0	Homogenous Heterogeneous	Exponential	Continuous-parameters	Closed form formula
Gershwin (1987)	Discrete-time	Multi-station (2 station subsystems)	Finite N	Synchronous	Geometric	2 states for stations	Approximate Decomposition
Gershwin and Schick (1983)	Discrete-time	2 station 3 station	$N \leq 15$	Synchronous	Geometric	2 states for stations	By solving linear equations
Muth and Yeralan (1981)	Discrete-time	2 station	Finite N	Homogenous Balanced line	Geometric	4 states for stations	Approximation

All the mentioned studies except Tan (1997) focused on operation-dependent failures, which is also the case in this study. To begin with, systems containing two stations have received most attention as they are easy to model and analyze compared to the longer lines. For longer production lines, the exact analytical evaluation brings conceptual and computational model complexity. As a result, Gershwin (1987) introduced the decomposition method to estimate the system behavior of multi-stations. Thereafter, many studies have concentrated on decomposition approximations for multi-station production lines. The decomposition method decomposes the system into several two-station representative parts that are examined independently and combined somehow to represent the whole system.

It is necessary to point out that the complexity is reduced by decreasing the number of states for stations in the model. Studies by Gershwin and Schick (1983) and Gershwin (1987) defined mainly two station states, namely operational and non-operational machines at stations. Yet, idle conditions of the machines, namely starvation and blockage, are also important to define and analyze the model.

Exact studies are relatively rare compared to the approximation studies in the literature. Exact studies are restricted with two-station or three-station production lines. Decomposition approximations, on the other hand, allow evaluation of longer lines, yet they typically require intermediate buffers of a certain capacity. In summary, the context of this study brings a novelty for modeling production lines with no intermediate buffers.

1.3 Contribution of This Study

Production lines with unreliable machines have received considerable attention in the literature. Especially two-station systems are most commonly studied, since these systems are easier to handle and analyze. The complexity with multi-station lines is that, with an increasing station number, the state space of the system has to be substantially expended. Hence, model handling and computation becomes difficult.

Recent research on multi-station lines with a finite buffer has focused on the decomposition approach to estimate the system behavior. In the approximation approaches, long line systems are partitioned into two-station sub-lines that when they are connected, represent the whole system. Such approaches require intermediate buffer storages of capacities at least two or three and the states of stations are classified into two, namely being operational and non-operational stations. Those restrictions prevent irregularities in the transition probability matrix and allow easy manipulation and solution of the steady-state solutions.

The trend in modern manufacturing practices, on the other hand, such as performed by the Toyota Production System (TPS), is to eliminate intermediate storages between stations as much as possible. Different from approximation methods, this study, based on the configuration of an automotive factory model, develops an efficient algorithm to analyze the exact system behavior of multi-station lines without intermediate buffer through generation of automated transition probability matrix and solution of linear steady-state equations.

With the proposed methodology, exact solutions are available for lines having more than two machines. Different from the past studies, our model defines five different station states considering the blockage and starvation of stations due to machine interference as a result of unreliable machines in the production line. In this research, discrete-time Markov chains are used to explore the system characteristics. Furthermore, an algorithm is introduced to efficiently generate the transition probability matrices for exact computations.

The algorithm allows the use of software packages to handle computations and compute exact solutions of such linear systems. The long-run behavior of the system is obtained via an algorithm, developed within this study, and implemented by the computational software package MATLAB. System performance measures, such as starvation and blockage probabilities of each station, as well as overall production rate and work-in-process have been the subject of analysis.

In addition, the production rate and the work-in-process measures over failure and repair probabilities are approximated with a polynomial curve-fit to establish simple and useful empirical formulas for lines consisting three, four and five stations. Numerical analyses show that the proposed methodology is effective for exact solutions and the suggested formulas are reasonably accurate for approximate solutions.

1.4 Outline

In Chapter 2, an introduction of the discrete-time Markov chain model and the system model assumptions are given. Chapter 3 presents the backward induction algorithm that generates the number of system states in multi-station lines. In Chapter 4 the principles of the backward algorithms are presented, namely the transition probability matrix generation algorithms. Thereafter the proposed algorithms are described. Chapter 5 explains the system performance measures and, followed by Chapter 6, which briefly exhibits the analysis of the sampled numerical results and qualitative observations in terms of system performance measures. Furthermore, Chapter 7 presents numerical analysis for developing simple and reasonably accurate production rate and work-in-process equations for approximation. Finally, Chapter 8 concludes the study.

CHAPTER 2

THE MARKOV CHAIN MODEL

Markov chains are by definition memory-less, which means that at any time, the transition probability between the system states depends only on the current state. It is independent of the past history of transitions; in other words, the transition from one system state to another is independent of how the system originally got to the past states. In this study, a discrete-time Markov chain model that represents the system behavior of the production lines is formulated. Herein, each period is represented by the constant duration of processing of work pieces by any station in the production line.

2.1 System Parameters

Machines are unreliable in nature. In the stochastic model under study, each machine has a unique probability to fail at the end of periods if it is operating during the current period. In addition, the machine failures occur at the end of periods, after the station completes its operation on the work piece. As Schick and Gershwin (1978) assumed, the repair of a failed machine starts at the beginning of the next period after the failure happened. Similarly, there is a constant probability of repair for failed machines during the periods. For machine *(i)*, where $1 \leq i \leq N$, the failure and repair parameters are geometrically distributed and defined as:

q[i]: The failure probability of a working machine *(i)*

r[i]: The repair probability of a failed machine *(i)*

(qb[i] = 1 – q[i] and *rb[i] = 1 – r[i])*

2.2 Station States

Each station in the production line can be represented by one of the five defined station states, each representing the condition of the station throughout a period from the start of the period. All the station states define the system state of the production line at the beginning of the current period and remain the same during the period. Station state transitions occur at the end of the periods hence, therefore the change in the system states also occurs at the end of the periods. A period is defined as the sequence that a unit is processed by the stations. Five possible station states are described as:

1) **Up (U)**

   If a machine is in up condition at the beginning of the period, it operates and processes the work piece during the period. The machine completes processing the piece at the end of the period. As machine breakdowns occur at the end of periods by definition, the machine can either fail or can be in good condition after it completes processing the work piece at the end of the period that is at the beginning of the next period.

2) **Down (D)**

   If a machine is in down condition at the beginning of the period, then during the period the machine can either be repaired with probability $r[i]$ or cannot be repaired with probability $rb[i] = 1 - r[i]$.

3) **Blocked (B)**

   If a machine, with up condition in the beginning of the period, finishes processing a work piece and does not fail at the end of the period, while the downstream machine is full, then the machine cannot pass the item to the next station and the machine becomes blocked. If a machine is blocked at the beginning of the period, for the next period it can still be blocked or not depending on the downstream station state. Note that a blocked machine does not fail.

4) **Down-Blocked (DB)**

If a machine, with up condition in the beginning of the period, finishes processing a work piece and then fails at the end of the period, while the downstream machine is full, then the machine cannot pass the work piece that is completed and the machine becomes both down and blocked. It can therefore not deliver the work piece until the downstream station is available and the machine needs repair to operate again. A down-blocked machine at the beginning of the period can either be repaired or not during the period and it can either pass the finished item or not to the downstream, which demonstrates the possible station states of the machine at the beginning of the next period.

5) **Starved (S)**

If a machine with good condition is not fed by the upstream, then it is starved. Similarly, a starved machine does not fail.

The possible states for the stations are summarized in Figure 2.1**.** As Figure 2.1 demonstrates, all five station states are valid for stations in the line except the first and the last one. Recall that, station 1 can never be blocked or down-blocked due to the assumption of infinite storage. In addition, station *N* is never starved due to infinite supply.

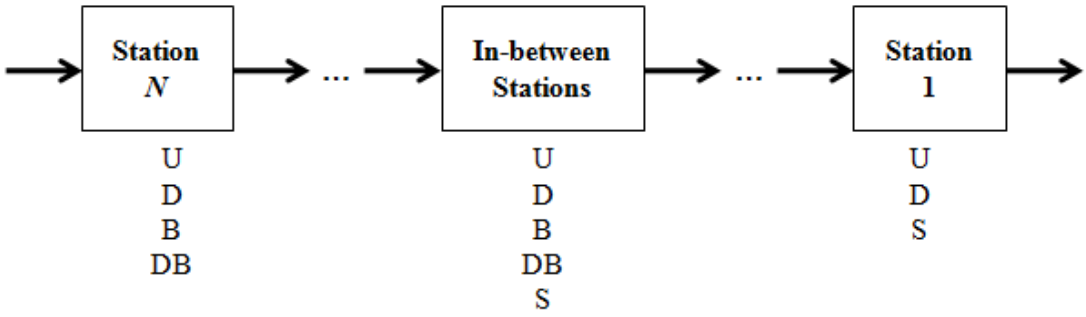

Figure 2.1 Station States

2.3 Model Assumptions

Schick and Gershwin (1978) is one of the pioneers who studied discrete-time Markov chains in modeling manufacturing lines. Regarding their model and assumptions, the relevant assumptions that define the system are given by:

1) The model considers homogenous lines where the processing time for stations is constant and equal for all stations.

2) Time is scaled so that the station periods take one time unit.

3) Transportation time between stations is negligible.

4) Machine capacities are limited to one; only one unit can be processed at a time.

5) An infinite supply of work pieces is available to the station *N*, and an unlimited storage area is considered for the last station, defined as station 1. As a result, station *N* is never starved, and station 1 is never blocked.

6) No intermediate buffer between machines is present.

7) Only operation dependent failures of machines are considered, which means that machines can only fail while they are processing a work piece. The failure probability of a starved or blocked machine is therefore zero.

8) Whenever a machine is processing a work piece, there is a unique probability *(q[i])* that the machine fails. By convention, machine breakdowns occur at the end of periods after machines complete their operations on the work piece.

9) There is a unique probability *(r[i])* that given a failed machine at the beginning of any period can be repaired during the period.

CHAPTER 3

THE CARDINALITY OF SYSTEM STATES

The cardinality of system states is crucial in order to analyze the system behavior. Transition probability matrix should be constructed to foresee the system performance. The size of the matrix and the complexity of the exact analysis directly depend on the cardinality of the system state set. Increasing the number of stations, increases the cardinality of the system states accordingly. To analyze the effect of increasing the number of stations in production lines on the total number of system states, backward induction is applied.

3.1 Backward Induction Algorithm

Notation:

$F(i, j)$ is the number of feasible downstream station states of station (i), being in state (j), where $1 \leq i \leq N$ and $j \in M$, $M = \{U, D, DB, B, S\}$.

$G(N)$ is the total number of system states in the production line containing (N) stations.

Initial Conditions for $i = 1$

$F(1, U) = F(1, D) = F(1, S) = 1,$
$F(1, B) = F(1, DB) = 0.$

Recursion Equations for $i = 2, 3, \ldots, N$

$$F(i, U) = F(i-1, U) + F(i-1, D) + F(i-1, S) + F(i-1, B) + F(i-1, DB)$$
$$F(i, D) = F(i, S) = F(i, U) \quad (1)$$

$$F(i, B) = F(i-1, D) + F(i-1, B) + F(i-1, DB)$$
$$F(i, DB) = F(i, B) \quad (2)$$

The total number of system states of a line having N stations:

$$G(N) = F(N, U) + F(N, D) + F(N, B) + F(N, DB) \quad (3)$$

Note that the station N is never starved.

3.2 Explanatory Computations

3.2.1 Two-Station Production Line

$$F(2, U) = F(1, D) + F(1, U) + F(1, S) + F(1, B) + F(1, DB)$$
$$= 1 + 1 + 1 + 0 + 0 = 3.$$
$$F(2, D) = F(2, S) = F(2, U) = 3.$$

$$F(2, B) = F(1, D) + F(1, B) + F(1, DB)$$
$$= 1 + 0 + 0 = 1.$$
$$F(2, DB) = F(2, B) = 1.$$

The cardinality of the system states of a two-station production line is:

$$G(2) = F(2, U) + F(2, D) + F(2, B) + F(2, DB) = 3 + 3 + 1 + 1 = 8.$$

3.2.2 Three-Station Production Line

$$F(3, U) = F(2, D) + F(2, U) + F(2, S) + F(2, B) + F(2, DB)$$
$$= 3 + 3 + 3 + 1 + 1 = 11.$$

$F(3, D) = F(3, S) = F(3, U) = 11.$

$F(3, B) = F(2, D) + F(2, B) + F(2, DB)$
$= 3 + 1 + 1 = 5.$
$F(3, DB) = F(3, B) = 5.$

The total number of the system states of a three-station production line is:
$G(3) = F(3, U) + F(3, D) + F(3, B) + F(3, DB) = 11 + 11 + 5 + 5 = 32.$

3.2.3 Four-Station Production Line

$F(4, U) = F(3, D) + F(3, U) + F(3, S) + F(3, B) + F(3, DB)$
$= 11 + 11 + 11 + 5 + 5 = 43.$
$F(4, D) = F(4, S) = F(4, U) = 43.$

$F(4, B) = F(3, D) + F(3, B) + F(3, DB)$
$= 1 + 5 + 5 = 21.$
$F(4, DB) = F(4, B) = 21.$

The total number of the system states of a four-station production line is:
$G(4) = F(4, U) + F(4, D) + F(4, B) + F(4, DB) = 43 + 43 + 21 + 21 = 128.$

3.2.4 Five-Station Production Line

$F(5, U) = F(4, D) + F(4, U) + F(4, S) + F(4, B) + F(4, DB)$
$= 43 + 43 + 43 + 21 + 21 = 171.$
$F(5, D) = F(5, S) = F(5, U) = 171.$

$F(5, B) = F(4, D) + F(4, B) + F(4, DB)$
$= 43 + 21 + 21 = 85.$
$F(5, DB) = F(5, B) = 85.$

The total number of the system states of a five-station production line is:
$G(5) = F(5, U) + F(5, D) + F(5, B) + F(5, DB) = 171 + 171 + 85 + 85 = 512.$

The algorithm iterates the variable $i = 2, 3, \ldots, N$ to recursively increase the number of stations, as a result of which, the system state set cardinality increases accordingly.

3.3 The Total Number of States

Theorem 1:

$$G(N+1) = 4 \cdot G(N) \quad for\ 2 \leq N \tag{4}$$

Proof:

Using Equation 3:

$$G(N+1) = F(N+1, U) + F(N+1, D) + F(N+1, B) + F(N+1, DB) \tag{5}$$

One can rewrite Equation 5 by using recursion equations as:

$$\begin{aligned} G(N+1) = {} & 2 \cdot [F(N, U) + F(N, D) + F(N, S) + F(N, B) + F(N, DB)] \\ & + 2 \cdot [F(N, D) + F(N, B) + F(N, DB)] \end{aligned} \tag{6}$$

After substituting (2) and (3) into (6), one finds:

$$\begin{aligned} G(N+1) = {} & 2 \cdot [F(N, U) + F(N, U) + F(N, U) + F(N, B) + F(N, B)] \\ & + 2 \cdot [F(N, U) + F(N, B) + F(N, B)] \end{aligned}$$

$$G(N+1) = 8 \cdot [F(N, U) + F(N, B)] \tag{7}$$

Finally, substituting (2) and (3) into (1), one can rewrite *G (N)* as:

$$G(N) = F(N, U) + F(N, U) + F(N, B) + F(N, B]$$
$$= 2 \cdot [F(N, U) + F(N, B)] \quad (8)$$

Hence; using (7) and (8), Equation 4 is satisfied.

The theorem shows that when the number of stations in a production line is increased by one, then the total number of system states increases by four times.

Theorem 2:

$G(N) = 2^{(2N-1)}$ *for* $2 \leq N$

Proof:

Considering the minimum number of stations in a production line, which is two, all possible system states are enumerated in Table 3.1. In addition, the line in consideration is shown in Figure 3.1.

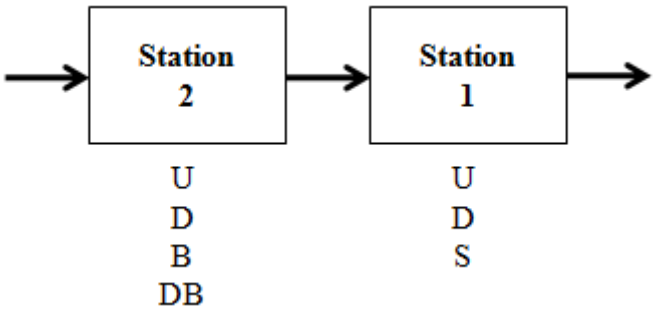

Figure 3.1 Two-Station Line

Table 3.1 System States of the Two-Station Production Line

Station 2	Station 1
Up	Up
Up	Down
Up	Starved
Down	Up
Down	Down
Down	Starved
Blocked	Down
Down-Blocked	Down

As shown in Table 3.1, by explicit enumeration: $G(2) = 8$. Considering $N = 2$ and using Equation 4 with $G(2) = 8$, the total number of system states can explicitly be written as:

$$G(N) = 8 = 2^{(N+1)}$$

$$G(N+1) = 4 \cdot G(N) = 2^{(N)} \cdot 2^{(N+1)} \tag{9}$$

After rewriting Equation 9, one can conclude that:

$$G(N) = 2^{(N-1)} \cdot 2^{(N)} = 2^{(2N-1)} \quad for\ 2 \leq N$$

Hence, given an N-station production line, the total number of system states is:

$$G(N) = 2^{2N-1}$$

CHAPTER 4

TRANSITION PROBABILITY MATRIX GENERATION

The generation of the transition probability matrix is essential to calculate the long-run system behavior through steady-state probabilities of the Markov chain model. However, due to the increasing size of the transition probability matrix, it is not possible to handle the model and the computations manually. The matrix generation should be automated in order to have the exact analysis for multi-station lines. Table 4.1 shows the dimensions of the transition probability matrices based on Theorem 2.

Table 4.1 Dimensions of the Transition Probability Matrices

Number of Stations	Total Number of System States	Dimensions of the Transition Probability Matrix
2	8	8 x 8
3	32	32 x 32
4	128	128 x 128
5	512	512 x 512
6	2048	2048 x 2048
…	…	…
N	2^{2N-1}	2^{2N-1} x 2^{2N-1}

The logic behind the automated transition matrix construction has two stages:

- Generation of single step transitions from current to the next system state.
- Calculation of transition probabilities.

4.1 Generation of Single Step Transitions

In the developed methodology, the backward induction was implemented to investigate one-step transitions between the system states. The procedure generates the station states starting at the final station towards the initial station, to determine every possible transition from a system state. After generating all possible transitions between the system states it is rather easy to find the transitions probabilities.

The forward induction was however found to be insufficient to model the system. This is due to the peculiar structure of the blocked state of the stations. Considering a starved machine in the current period, the state of the machine for the next period can be determined only by looking at the state of the preceding machine making a forward recursion feasible. Therefore, if the previous machine is able to supply a piece, then the downstream will be in up condition, if not, then the machine will be starved at the beginning of the next period. That is to say, the starvation condition of the stations enables a forward formulation; on the other hand, the blockage condition of the machines does not allow a forward formulation. To determine the condition of a blocked machine both forward and backward information are necessary. Whether a machine will be blocked or not at the next period depends on all other consequent machine states. A blocked machine at or near the end of the production line for example can cause all the upstream machines to be blocked suddenly. Henceforth, to obtain the state of a blocked machine at the next period, one needs to look at the arrangement not only the upstream station but also the following downstream station. This is basic principle triggered this study and formulates the reason why the backward methodology was implemented instead of a forward one.

In summary, the backward induction algorithm was developed in order to generate all one-step transition states from a priori known system state vector, which is applicable to any production line with N stations. The purpose of the algorithm is to generate the transition probability matrix of the system and then analyze the related performance measures to understand the system behavior.

4.2 The Principles

4.2.1 Preliminary Notation

N is an integer that indicates the number of stations in the production line. The minimal number of stations is two, hence $N \geq 2$.

$X_j(i)$ is the station state of station (i) being in the $(j)^{th}$ station state with $1 \leq i \leq N$. Notice that a backwards notation is used, where $X_j(N)$ is the state of the first, that is, the upstream station state, and $X_j(1)$, the state of the last station.

X_j is the system state indicator and a row vector with length N, *i.e.* $X_j \in M^{1xN}$ with $1 \leq j \leq 2^{2N-1}$ is the system state index. The state of an N-station production line is represented by $X_j = [X_j(N), X_j(N-1), \ldots, X_j(2), X_j(1)]$, containing all station states.

There are transitions among systems states. Those states that can be reached from system state X_j are denoted by $Y_{j,k}$. $Y_{j,k}$ is a row vector with length N, i.e. $Y_{j,k} \in M^{1xN}$. The first subscript of $Y_{j,k}$ denotes that the system state can be reached from state X_j, $1 \leq j \leq 2^{2N-1}$. Depending on the specific system state X_j, there are a number of alternate states that could be reached in a single step. The second subscript of $Y_{j,k}$ enumerates these alternatives. The elements of $Y_{j,k}$ are likewise defined. $Y_{j,k}(i)$ is the state of station (i) of the system state $Y_{j,k}$, $1 \leq i \leq N$.

To illustrate, given $X_j = [U, D, S]$, the final machine is starved, that is $X_j(1) = S$. The second machine is down, $X_j(2) = D$ and the initial machine is up, which is represented by $X_j(3) = U$. The one-step transitions from X_j are provided in Table 4.2.

Table 4.2 Transitions from $X_j = [U, D, S]$

$Y_{j,1}$ = [U, U, S]	$Y_{j,2}$ = [D, U, S]	$Y_{j,3}$ = [B, D, S]	$Y_{j,4}$ = [DB, D, S]

4.2.2 The Methodology

The basic principle behind the presented procedures is to find the system states ($Y_{j,k}$) that are reachable from X_j in a single step. It is possible to obtain the subsequent period's station states, *i.e.* $Y_{j,k}(i)$, in a recursive manner. The following information is necessary to find $Y_{j,k}(i)$, as demonstrated in Figure 4.1.

- The current state of the $(i)^{th}$ station, *i.e.* $X_j(i)$,
- The state of the upstream station in the current system state, namely $X_j(i+1)$,
- The state of the downstream station in the following step, namely $Y_{j,k}(i-1)$.

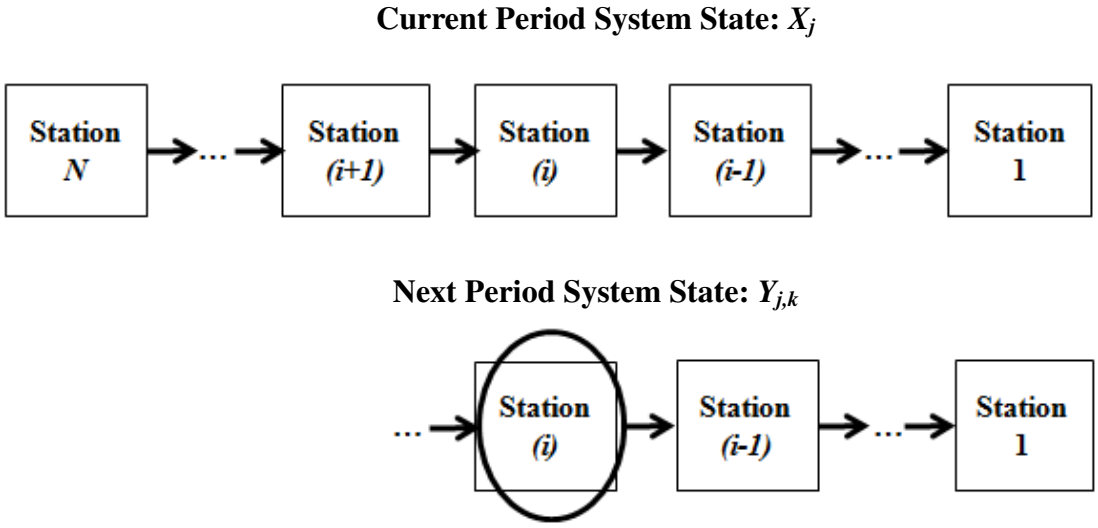

Figure 4.1 Recursive Manner of Generating Station States

Observation: There is at least one transition ($Y_{j,k}$) generated from every current state X_j. In addition, the total number of feasible solutions, *i.e.* the upper limit on k of $Y_{j,k}$ can easily be calculated from X_j and is defined by the parameter Z_j.

Definition: Generating set $\Lambda_1 = \{U, D, DB\}$

The station states, included in set Λ_1, doubles the number of transitions $(Y_{j,k})$. This is because an up station can either be broken down or not, and a down station can either be repaired or not, therefore these conditions generate twice more possibilities for the one-step transition states. For example, given current system state as $X_j = [U, U, U]$, all of the three station states are included in Λ_1. Hence, there are a total of $Z_j = 2^3$ different one-step transitions $(Y_{j,k},\ 1 \leq k \leq Z_j)$ present from the current system state X_j.

The Table 4.3, 4.4, 4.5 and 4.6 were prepared in order to represent all one-step station state transitions for the in-between station. According to the information of the current station state $X_j(i)$, the algorithm selects the applicable table and then with the information of $X_j(i+1)$ and $Y_{j,k}(i-1)$, the new transition states ($Y_{j,k}(i)$ for $1 \leq k \leq Z_j$) are easily found from the related tables in a backward recursive manner.

Henceforth, considering an N-station production line, using the tables and the necessary downstream and upstream information, the $Y_{j,k}(2)$ to $Y_{j,k}(N-1)$ transition states can be created. The transitions state of the first and the last machine are defined as the boundary conditions below.

Table 4.3 Working States from $X_j(i) \in \{U, B, DB\}$

$X_j(i) \in \{U,B,DB\}$	$Y_{j,k}(i-1)$					
		U	**D**	**DB**	**B**	**S**
$X_j(i+1)$	**U**	U	B	B	B	B
	D	S	B	B	B	B
	DB	U	B	B	B	B
	B	U	B	B	B	B
	S	S	B	B	B	B

Table 4.4 Non-Working States from $X_j(i) \in \{U, DB\}$

$X_j(i) \in \{U,DB\}$	$Y_{j,k}$ (i-1)					
		U	**D**	**DB**	**B**	**S**
X_j (i+1)	**U**	D	DB	DB	DB	DB
	D	D	DB	DB	DB	DB
	DB	D	DB	DB	DB	DB
	B	D	DB	DB	DB	DB
	S	D	DB	DB	DB	DB

Table 4.5 Working States from $X_j(i) \in \{D, S\}$

$X_j(i) \in \{D,S\}$	$X_{j,k}$ (i-1)					
		U	**D**	**DB**	**B**	**S**
X_j (i+1)	**U**	-	U	U	U	U
	D	-	S	S	S	S
	DB	-	U	U	U	U
	B	-	U	U	U	U
	S	-	S	S	S	S

Table 4.6 Non-Working States from $X_j(i) \in \{D\}$

$X_j(i) \in \{D\}$	$X_{j,k}$ (i-1)					
		U	**D**	**DB**	**B**	**S**
X_j (i+1)	**U**	-	D	D	D	D
	D	-	D	D	D	D
	DB	-	D	D	D	D
	B	-	D	D	D	D
	S	-	D	D	D	D

Note that empty columns in Table 4.5 and 4.6 indicate that there is no transition between those states.

Boundary Conditions:

- ***Initialization***: Generation of $Y_{j,k}(1)$

Given the current station state of the last station, *i.e.* $X_j(1)$, the last station state of the transition system, $Y_{j,k}(1)$, can easily be calculated based on the information of $X_j(2)$ only. Table 4.7 and 4.8 summarize the transitions for the last station.

- ***Finalization*** : Generation of $Y_{j,k}(N)$

At the final step of the algorithm all transition states of $Y_{j,k}(N)$ are created. $Y_{j,k}(N)$ is literally the first station of the transition system. As by definition an infinite supply is assumed, the algorithm assumes a hypothetical station, which is always in up condition, in the upstream. Hence, from the developed four tables, $Y_{j,k}(N)$ can be easily found with the information of $X_j(N)$, $Y_{j,k}(N-1)$ and the hypothetical station given $X_j(N+1) = U$.

Table 4.7 From $X_j(2) \in \{U, B, DB\}$

$X_j(2)$	$Y_{j,k}(1)$
U	U
D	S
DB	U
B	U
S	S

Table 4.8 From $X_j(2) \in \{D, S\}$

$X_j(2)$	$Y_{j,k}(1)$
U	D
D	D
DB	D
B	D
S	D

4.3 The Algorithms

- **Tree Expansion Method**

The tree expansion algorithm is developed to generate all possible transition vectors from a system state vector. An approach like a branching tree is introduced to improve the computational effectiveness.

- **Tabular Method**

A tabular form based algorithm is introduced by logical shifting of the station states in order to generate all possible transitions from a system state vector. The main difference between the two algorithms is that the number of transitions from a system state is implicitly calculated within the tree expansion method; however, the tabular method is based on the direct enumeration of the transition states.

4.4 Tree Expansion Method

The tree expansion algorithm is developed to generate all possible transitions between system states. A methodology similar to branching is used to automatically generate the transition probability matrix in order to reduce the modeling and computational burden.

The algorithm includes the following routines:

1) **Initialization:** Starts the core-routine with a priori known system state.
2) **Core-Routine:** Defines the general structure of the algorithm and determines the terminating condition.
3) **Sub-Routine:** This is the branching procedure, which is called by the core-routine. Given a system state, it returns the system state transitions through tree expansion by recursion.

Notation:

M is the set of all possible station states: $M = \{U, D, DB, B, S\}$.
Λ_1 is the set from which two branches can be generated: $\Lambda_1 = \{U, D, DB\}$.
Λ_2 is the set from which a single branch can be generated: $\Lambda_2 = \{B, S\}$.
Ω_1 is the initial set of system states to initialize the core-routine.
Ω_2 is the set of system states that are updated after the sub-routine is run. The set includes the system states that have not entered the core-routine yet. Thus, it is the set of system states that need to be processed in the next loop by the algorithm.
Ω_3 is the set of all system states generated at the end of each run of the sub-routine.

Ω_1 and Ω_2 are updated with Ω_3. Ω_1 and Ω_2 are external sets that are generated and updated after each loop of the algorithm; whereas Ω_3 is a local set generated and erased at every loop of the sub-routine. Note that $\Omega_2 \subseteq \Omega_1$.

β_1 is the set of all y_1, which are the states of nodes at level 1, generated from $X_j(1)$.
β_{i+1} is the set of all y_{i+1}, which are the state of the nodes at level $(i+1)$ after set β_i, where $1 \leq i \leq N-1$, $N \geq 3$ and $y_i \in M$.

The tree expansion algorithm generates and fills $Y_{j,k}(i+1)$'s by using Table 4.3, 4.4, 4.5 and 4.6. Thus instead of $X_j(i+1)$ and $Y_{j,k}(i-1)$ in the tables to find $Y_{j,k}(i)$'s, $X_j(i+2)$ and y_i are used to refer to the predefined tables in section 4.4.2.

Initialization:

The main algorithm starts with an a priori known state. Notice that N is the total station number in the production line and is determined by the user at the initialization step by defining X_1.

Set Ω_1 and set Ω_2 are initialized with the system state vector X_1 as follows:
$\Omega_1 = \{X_1\}$ and $\Omega_2 = \{X_1\}$, where $X_1(i) = U$ for $1 \leq i \leq N$.

4.4.1 The Structure of the Tree Expansion Algorithm

The tree expansion algorithm starts branching from the final station state $X_1\ (1)$ creating $Y_{1,k}\ (1)$. Then the algorithm iterates with the logic of the Tables 4.3, 4.4, 4.5, 4.6, 4.7 and 4.8, and generates transitions by branching at levels and creating new nodes filled with the station states from tables up to the first station state $Y_{j,k}\ (N)$. In this manner the algorithm constructs the state transitions as a tree structure.

A tree graph is a connected graph without any cycles as shown in Figure 4.2. The tree has one unique root which is $X_j\ (1)$ at level 0. Thereafter, at each subsequent level, one or two branches are generated from each node. The number of levels is restricted to the number of stations in the line, namely N, which also represents the depth of the tree. Such a tree is referred to as a rooted tree. For more information on rooted trees the reader is referred to Lipschutz and Lipson (1997).

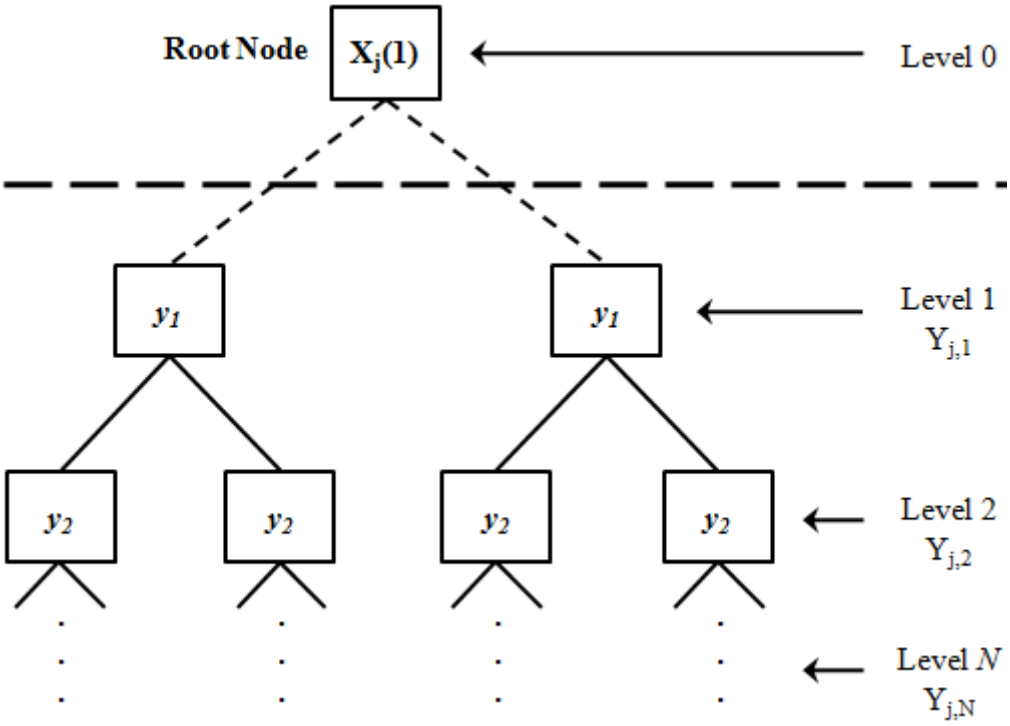

Figure 4.2 Rooted Tree Structure of the Tree Expansion Algorithm

A rooted tree is a tree graph where a vertex is designated as the root of the tree. There is a unique simple path from the root to any other vertex, that is, every vertex except the root has a unique parent in the structure. It is important to notice that the parent of a node is the node that is connected to it on the upward path to the root node as shown in Figure 4.3.

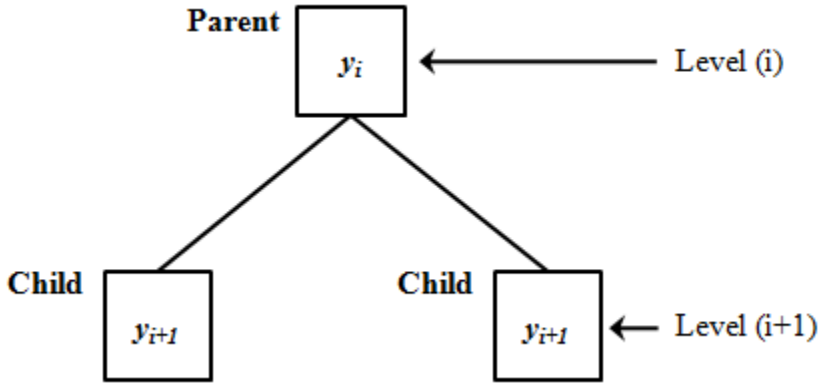

Figure 4.3 Parent-Child Relations in a Rooted Tree

Notation:

$g\ (y_i)$ is a function which finds the parent of the given node y_i and returns the station state of the parent node.

A rooted tree structure enables precedence relation between the vertices. Consequently, the rooted tree is a beneficial tool to enumerate all the logical possibilities of a sequence of events where each event can occur within a succession in a finite number of ways. This is exactly the case in the model under study.

4.4.2 The Pseudo Code of the Tree Expansion Algorithm

Initialization:

Input X_1 and define $\Omega_1 = \{X_1\}$, $\Omega_2 = \{X_1\}$.

Core-Routine:

Step 1 Stop, if $| \Omega_2 | = 0$,

Step 2 Get a system state X_j, $X_j \in \Omega_2$,

Step 3 Update Ω_2 by eliminating the selected X_j from set Ω_2

Step 4 Run the sub-routine, which updates set Ω_1 and set Ω_2,

Step 5 Go to *Step 1*.

Sub-Routine: Generation of the nodes from level 1 to level N

Phase 1: Generation of level 1

Step 1a If $X_j(1) \in \Lambda_1$, then:

Create two new nodes by generating two branches from node $X_j(1)$.

Check $X_j(2)$, and then generate one node using Table 4.7 and the other node using Table 4.8.

Step 1b If $X_j(1) \in \Lambda_2$, then:

Create one new node by generating a single branch from node $X_j(1)$.

Check $X_j(2)$, and then generate the child node using Table 4.7.

Step 2 Set $\beta_1 = \{y_1 \mid y_1$ is the state of the nodes at level 1, generated from $X_j(1)\}$.

Phase 2: Generation of level 2 to level $(N-1)$

Step 1 Iterate for every $i = 1, 2, \ldots, N-2, N-1$

Step 2 For all $y_i \in \beta_i$

Step 2.1a If $y_i \in \Lambda_1$, then:

Create two new nodes (y_{i+1}) by generating two branches from node y_i.

Step 2.1b If $y_i \in \Lambda_2$, then:

Create one new node (y_{i+1}) by generating a single branch from node y_i.

Step 3 Check $X_j(i+2)$ and y_i,

Step 3.1a If $X_j(i+1) \in \{U, DB\}$, then:

Generate one node y_{i+1} using Table 4.3 and the other using Table 4.4.

Step 3.1b If $X_j(i+1) \in \{B\}$, then:

Generate the child node y_{i+1} using Table 4.3.

Step 3.1c If $X_j(i+1) \in \{S\}$, then:

Generate the child node y_{i+1} using Table 4.5.

Step 3.1d If $X_j(i+1) \in \{D\}$, then:

Generate one node y_{i+1} using Table 4.5 and the other using Table 4.6.

Step 4 Set $\beta_{i+1} = \{y_{i+1} \mid y_{i+1}$ is the state of the nodes at level $(i+1)$ from $\beta_i\}$

Phase 3: Generation of level N

Step 1 For all $y_{N-1} \in \beta_{N-1}$

Step 1a If $y_{N-1} \in \Lambda_1$, then:

Create two new nodes y_N by generating two branches from node y_{N-1},

Step 1b If $y_{N-1} \in \Lambda_2$, then:

Create one new node y_N by generating a single branch from node y_{N-1}.

Step 2 Set $X_j(N+1) = U$ and check y_{N-1}, then:

Generate one node y_N using Table 4.3 and the other using Table 4.4.

Step 3 Set $\beta_N = \{y_N \mid y_N$ is the state of the nodes at level N from set $\beta_{N-1}\}$

Phase 4: Traversing the tree from level N to level 1

Step 1 Iterate for every $i = N, N-1, \ldots, 2$

Step 1.1 Stop if $i = 2$ and go to *Step 2*.

Step 1.2 For all $y_N \in \beta_N$, call function $g(y_i)$, and set: $X_j(i) = g(y_i)$

Step 2 Set $\Omega_3 = \{X_j \mid X_j$ is the system state generated after *Step 1*$\}$

After phase 3, the rooted tree structure is constructed, at phase 4 the tree is traversed from level N to level 1 to obtain the transition system states. When all transition states from system state X_j are generated, the information of the system state and the transition states are saved to find the transition probabilities.

Phase 5: Updating Ω_1 and Ω_2

Step 1 For every $X_j \in \Omega_3$, initiate *Step 2*.

Step 2 If $X_j \in \Omega_1$, include X_j into set Ω_1 and Ω_2.

4.4.3 Examples of the Tree Expansion Algorithm

Example 1

The algorithm starts with a priori known system state, *i.e.* $X_1 = [U, U, U]$. The number of stations in the line, *i.e.* the length of the row vector (three in this case) is given to the algorithm by defining X_1. Recall that, it is the depth of the rooted tree. Figure 4.4 demonstrates the algorithm. The core-routine supplied level 0 information, and then at phase 1 of the sub-routine, level 1 is generated. Phase 2 generates level 2 and finally phase 3 generates the last level, which is level 3 here.

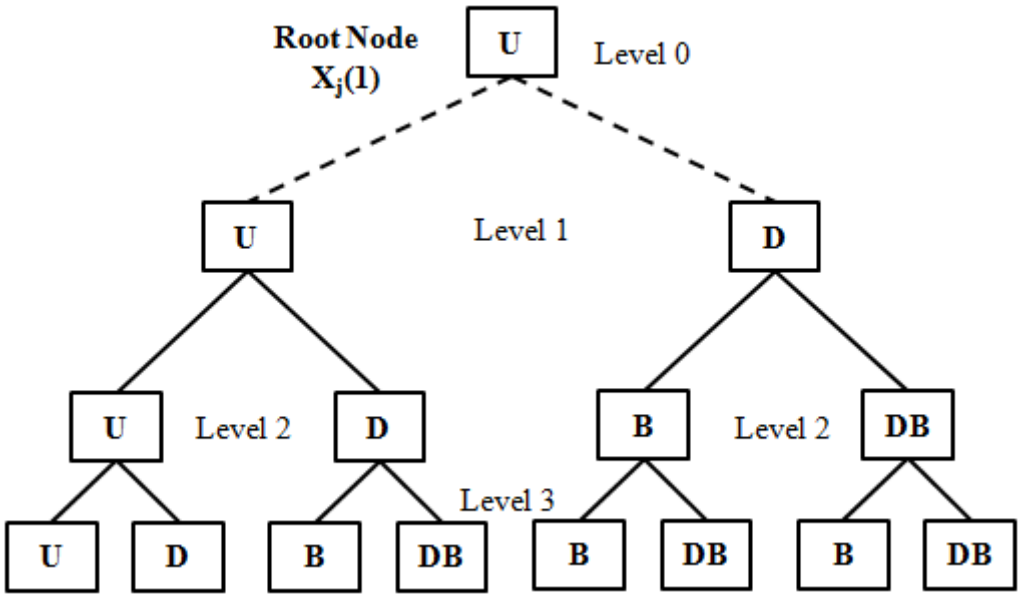

Figure 4.4 Generation from $X_1 = [U, U, U]$

At phase 4, the tree is traversed from level 3 to level 1 and the $Y_{1,k}$ *'s* are generated as presented in Table 4.9. The number of transition states *(*$Y_{1,k}$*)* are implicitly calculated within the algorithm. It is the total number of nodes at level *N*.

Table 4.9 Transitions from X_1

$Y_{1,1}$ = [U, U, U]	$Y_{1,5}$ = [B, B, D]
$Y_{1,2}$ = [D, U, U]	$Y_{1,6}$ = [DB, B, D]
$Y_{1,3}$ = [B, D, U]	$Y_{1,7}$ = [B, DB, D]
$Y_{1,4}$ = [DB, D, U]	$Y_{1,8}$ = [DB, DB, D]

Finally at phase 5, the algorithm updates sets Ω_1 and Ω_2. Set Ω_2 includes the new system state that was generated but whose transition states are still missing, and set Ω_1 keeping track of all the system states that have been generated.

Example 2

Given X_j *= [B, B, D],* the constructed rooted tree is in Figure 4.5. In addition, the transition vectors that are generated from the rooted tree in Figure 4.5 are stated in Table 4.10.

Table 4.10 Transitions from X_j *= [B, B, D]*

$Y_{j,1}$ = [U, U, U]	$Y_{j,2}$ = [B, B, D]

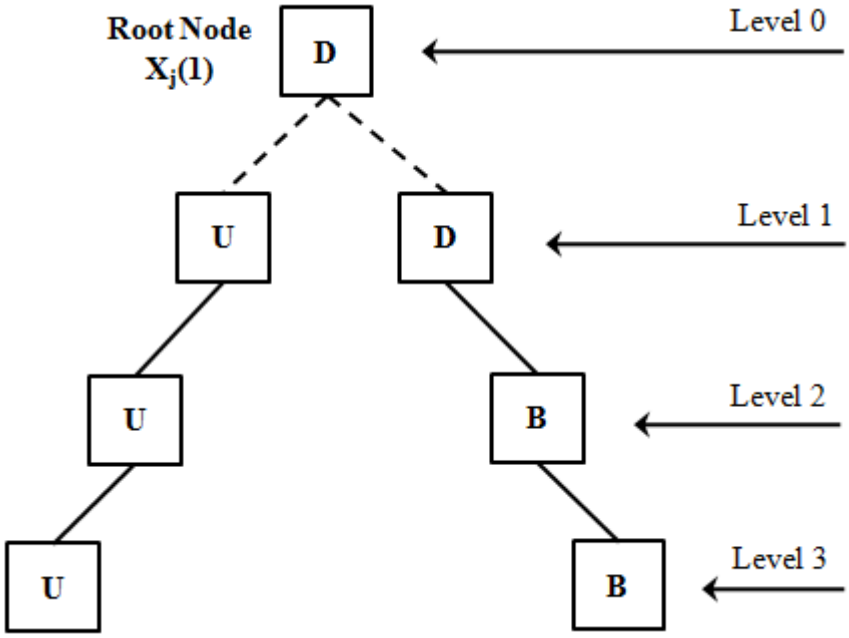

Figure 4.5 Generation from X_j = *[B, B, D]*

Example 3

Given X_j = *[D, B, D],* the rooted tree is constructed according to Figure 4.6. In addition, Table 4.11 shows the transition vectors found that are by backward traversing of the tree.

Table 4.11 Transitions from X_j = *[D, B, D]*

$Y_{j,1}$ = [U, S, U]	$Y_{j,3}$ = [U, B, U]
$Y_{j,2}$ = [D, S, U]	$Y_{j,4}$ = [D, B, D]

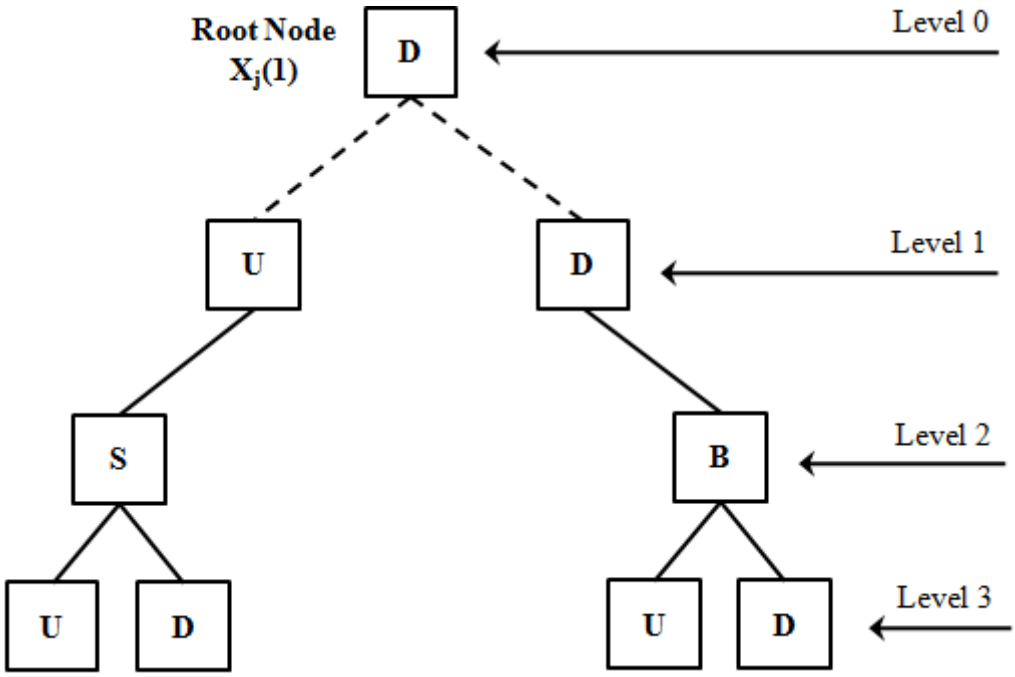

Figure 4.6 Generation from $X_j = [D, B, D]$

4.5 Tabular Method

The tabular algorithm uses logical shifting of the station states combined with the logic developed in the transition surveillance tables, namely Table 4.3, 4.4, 4.5, 4.6, 4.7 and 4.8, to generate all possible transitions between the system states. The algorithm yields the same results with the tree expansion method. They enable automated generation of transition probability matrices for multi-station lines.

The algorithm includes the following routines:

1) **Initialization:** Generates the system state set for *N* stations.
2) **Core-Routine:** Defines the transition matrix from a given system state, then the transition matrix is filled in by the sub-routine.
3) **Sub-Routine:** Constructs and fills in the transition matrix. The sub-routine generates a matrix, every row of which is a one-step transition.

Notation: Only additional notation is presented.

V_j is the total number of stations of X_j, where $X_j(i) \in \Lambda_1$, $1 \leq i \leq N$ and $V_j \leq N$.
Z_j is the number of transition vectors $Y_{j,k}$ generated from a system state vector X_j, where $1 \leq k \leq Z_j$. Note that $Z_j = 2^{V_j}$.

A^* is a $5^N \times N$ matrix, with every row a permutation of the station states over N stations. With the five possible station states this gives 5^N possible combinations.
A is a $2^{2N-1} \times N$ matrix, every row of which includes a feasible system state X_j.

Q_j^* is initially a zero matrix with dimensions $Z_j \times N$. Every row of Q_j^* is subsequently filled in within the sub-routine with the transition vector $Y_{j,k}$ that is generated from the system state X_j. Subsequently filling in Q_j^* per row results in Q_j
Q_j is a $Z_j \times N$ matrix, every row of which includes a transition ($Y_{j,k}$) from X_j.

$R(i) = Count(X_j(i))$ is the function defined to find the total number of station states in the range from $X_j(1)$ to $X_j(i)$ that belong to the set: $\Lambda_1 = \{U, D, DB\}$.

The variable $H(i)$ is a count of the number of these station states in the given range. The function returns a value $R(i)$, which is the number of consecutive rows in a column to be filled in within the algorithm.

The value $R(i)$ is associated to $H(i)$ and is calculated as:
$R(i) = Z_j / 2^{H(i)}$.

As an example consider $X_j = [U, D, B, D]$. Here, the total number of stations being in a state which is included in the set Λ_1 is: $V_j = 3$. Hence, the solutions of $R(i)$:

$Count(X_j(1)) = 2^{3-1} = 4$ $\quad$ $Count(X_j(3)) = 2^{3-2} = 2$
$Count(X_j(2)) = 2^{3-1} = 4$ $\quad$ $Count(X_j(4)) = 2^{3-3} = 1$

4.5.1 The Pseudo Code of the Tabular Algorithm

Initialization:

There are N machines in a line. Without considering the special requirements for the first and the last station, every machine can be at most in one of the five states ($M = \{U, B, S, D, DB\}$). There are 5^N possible permutations of N. Yet, some permutations are not feasible. These infeasible states are eliminated in the initialization step.

Phase 1: Generation of A^*

Step 1 Input N,

Step 2 Generate all permutations of the system states for N stations, and construct a $5^N \times N$ matrix, every row of which includes a permutation.

Phase 2: Generation of A

Step 1 Eliminate the system state vectors where station N is starved, hence discard the rows of A^* where $X_j(N) \in \{S\}$ due to infinite supply assumption.

Step 2 Eliminate the system state vectors where station 1 is either blocked or down-blocked, hence discard the permutations where $X_j(1) \in \{B, DB\}$ due to the infinite storage assumption.

Step 4 Eliminate the system state vectors that have a blocked or a down-blocked machine as the upstream of a starved machine in the line. Henceforth, the rows of A^* that satisfy the condition are discarded from A^*:

$X_j(i) \in \{S\}$ and $X_j(i+1) \in \{B, DB\}$ for $1 \leq i \leq N-1$

Clearly, a machine can never be starved if its upstream machine is blocked.

Step 5 Eliminate the system state vectors that have a blocked or a down-blocked machine as the upstream of an up machine in the line. The permutations that contain the condition are discarded from A^*:

$X_j(i) \in \{U\}$ and $X_j(i+1) \in \{B, DB\}$ for $1 \leq i \leq N-1$

By definition, a machine's capacity is one, thus a machine cannot process two units at the same time. At the end of each period the machine can therefore not pass one unit to the downstream while remaining blocked.

After phase 2, the remaining rows of the A^* matrix give the matrix A, which has a dimension of $2^{2N-1} \, x \, N$.

Core-Routine: Generation of Q_j^*

Step 1 For all X_j, contained at every $(j)^{th}$ row of the A matrix do the following:

Step 2 Given X_j, find V_j,

Step 3 Calculate Z_j,

Step 4 Generate a zero matrix Q_j^* of dimensions $Z_j \, x \, N$,

Step 5 Run the sub-routine.

Sub-Routine: Generation of Q_j

Phase 1: Filling the last column of the Q_j^* matrix

Step 1a If $X_j(1) \in \{U, D\}$,

Check $X_j(2)$ and fill the first half rows of the last column of Q_j^* from Table 4.7 and fill the rest of the rows of the last column of Q_j^* from Table 4.8.

Step 1b If $X_j(1) \in \{S\}$,

Check $X_j(2)$ and fill the last column of matrix Q_j^* from Table 4.7.

Phase 2: Column-wise filling in the rest of Q_j^* up to the first column

Step 1 Iterate for every $i = N-1, \ldots, 2$;

Step 2 If $X_j(i) \in \{U, DB\}$,

Step2.1 Find $R(i)$ using the function: $R(i) = Count\,(X_j(i))$,

Step 2.2 For all $k = 2 \cdot R(i) \cdot (c-1) + 1$, $c = 1, 2, \ldots, (Z_j / (2 \cdot R(i)))$, do:

Step 2.2a Define $k1 = k, (k+1), \ldots, (k + R(i) - 1)$, and

$k2 = (k + R(i)), (k + R(i) + 1), \ldots, (k + 2 \cdot R(i) - 1)$

Step 2.2b If $Y_{j,k}(i+1) \in \{D, B, DB\}$,

then $Y_{j,k1}(i) = B$ and $Y_{j,k2}(i) = DB$.

Step 2.2c If $Y_{j,k}(i+1) \in \{U\}$ and $X_j(i-1) \in \{U, B, DB\}$,

then $Y_{j,k1}(i) = U$ and $Y_{j,k2}(i) = D$.

Step 2.2d If $Y_{j,k}(i+1) \in \{U\}$ and $X_j(i-1) \in \{D, S\}$,

then $Y_{j,k1}(i) = S$ and $Y_{j,k2}(i) = D$.

Step 3 If $X_j(i) \in \{D\}$,

Step3.1 Find $R(i)$ using the function: $R(i) = Count(X_j(i))$,

Step 3.2 For all $k = 2 \cdot R(i) \cdot (c-1) + 1$, $c = 1, 2, \ldots, (Z_j / (2 \cdot R(i)))$, do:

Step 3.2a Define $k1 = k, (k + 1), \ldots, (k + R(i) - 1)$, and

$k2 = (k + R(i)), (k + R(i) + 1), \ldots, (k+2 \cdot R(i) - 1)$

Step 3.2b If $Y_{j,k}(i+1) \in \{D, B, DB, S\}$ and $X_j(i-1) \in \{U, B, DB\}$,

then $Y_{j,k1}(i) = U$ and $Y_{j,k2}(i) = D$.

Step 3.2c If $Y_{j,k}(i+1) \in \{D, B, DB, S\}$ and $X_j(i-1) \in \{D, S\}$,

then $Y_{j,k1}(i) = S$ and $Y_{j,k2}(i) = D$.

Step 4 If $X_j(i) \in \{S\}$,

Step 4.1 For all $k = 1, 2, \ldots, Z_j$, do:

Step 4.1a If $X_j(i-1) \in \{U, B, DB\}$, then $Y_{j,k}(i) = U$.

Step 4.1b If $X_j(i-1) \in \{D, S\}$, then $Y_{j,k}(i) = S$.

Step 5 If $X_j(i) \in \{B\}$,

Step 5.1 For all $k = 1, 2, \ldots, Z_j$, do:

Step 5.1a If $Y_{j,k}(i+1) \in \{D, B, DB\}$, then $Y_{j,k}(i) = B$.

Step 5.1b If $Y_{j,k}(i+1) \in \{U\}$ and $X_j(i-1) \in \{U, B, DB\}$, then $Y_{j,k}(i) = U$.

Step 5.1c If $Y_{j,k}(i+1) \in \{U\}$ and $X_j(i-1) \in \{D, S\}$, then $Y_{j,k}(i) = S$.

Phase 3: Filling the first column of Q_j*

Step 1 For all $k = 1, 3, 5, \ldots, Z_j$, do:

Step 1a If $X_j(1) \in \{U, DB\}$ and $Y_{j,k}(2) \in \{D, B, DB\}$, then $Y_{j,k}(1) = B$ and $Y_{j,k+1}(1) = DB$.

Step 1b If $X_j(1) \in \{U, DB\}$ and $Y_{j,k}(2) \in \{U\}$, then $Y_{j,k}(1) = U$ and $Y_{j,k+1}(1) = D$.

Step 1c If $X_j(1) \in \{D\}$ and $Y_{j,k}(2) \in \{D, B, DB, S\}$, then $Y_{j,k}(1) = U$ and $Y_{j,k+1}(1) = D$.

Step 2 For all $k = 1, 2, 3, \ldots, Z_j$, do:

Step 2a If $X_j(1) \in \{B\}$ and $Y_{j,k}(2) \in \{D, B, DB\}$, then $Y_{j,k}(1) = B$.

Step 2b If $X_j(1) \in \{B\}$ and $Y_{j,k}(2) \in \{U\}$, then $Y_{j,k}(1) = U$.

After phase 3, the transition matrix Q_j is formed, which includes a transition vector $Y_{j,k}$ from system state X_j at every row.

4.5.2 Examples of the Tabular Algorithm

Example 1

After defining $N = 3$, an A matrix with the dimensions of $2^{2N-1} \times N$ is constructed by the initialization routine of the algorithm. Every row of matrix A is a feasible system state X_j that is to be processed by the algorithm.

For instance, when the core-routine is run with a given $X_j = [U, U, U]$, the algorithm defines Q_j, an $8x3$ zero matrix. The sub-routine fills in the zero matrix starting from the last column (phase 1) and the middle columns (phase 2) as demonstrated in Figure 4.7.

In phase 3 of the sub-routine, the first column is filled in. The resulting matrix is defined as the transition matrix, Q_j of $X_j = [U, U, U]$, as presented in Figure 4.8.

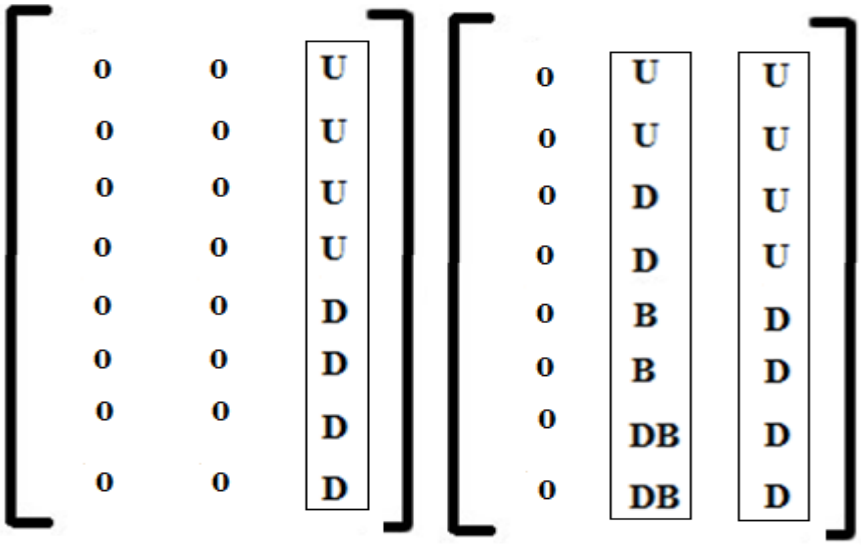

Figure 4.7 Column-wise Generations by the Tabular Method

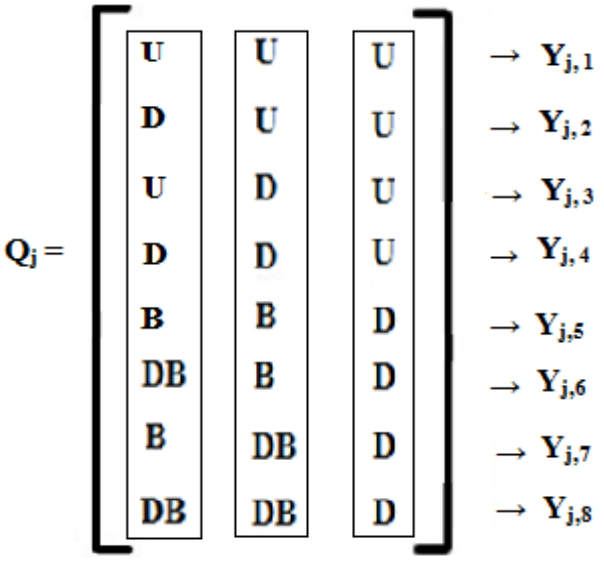

Figure 4.8 Transition matrix from $X_j = [U, U, U]$

Example 2

Given system vector $X_j = [D, D, S]$, the transition matrix generated by the algorithm is presented in Figure 4.9.

$$Q_j = \begin{bmatrix} U & S & S \\ D & S & S \\ U & D & S \\ D & D & S \end{bmatrix} \begin{matrix} \rightarrow Y_{j,1} \\ \rightarrow Y_{j,2} \\ \rightarrow Y_{j,3} \\ \rightarrow Y_{j,4} \end{matrix}$$

Figure 4.9 Transition matrix from $X_j = [D, D, S]$

As seen, there are a *4x3* matrix is constructed, rows of which correspond to different transitions from $X_j = [D, D, S]$ and columns include the station states.

4.6 Generation of the Transition Probabilities

After generating one-step transitions in a recursive manner by the proposed algorithms, it is easy to calculate of the transition probability between system states. Table 4.12 summarizes the applicable transition probabilities between station states between each consecutive period.

For example, given $X_j = [U, S, D, S]$, the constructed rooted tree is shown in Figure 4.10 and the resulting transition probabilities are presented in Table 4.13.

Table 4.12 Transition probabilities

	$Y_{j,k}$ (i) / X_j (i)	At the beginning of the next period				
		U	**B**	**S**	**D**	**DB**
At the beginning of the current period	**U**	qb[i]	qb[i]	q[i]	q[i]	q[i]
	B	1	1	1	0	0
	S	1	0	1	0	0
	D	r[i]	0	r[i]	rb[i]	0
	DB	r[i]	r[i]	r[i]	rb[i]	rb[i]

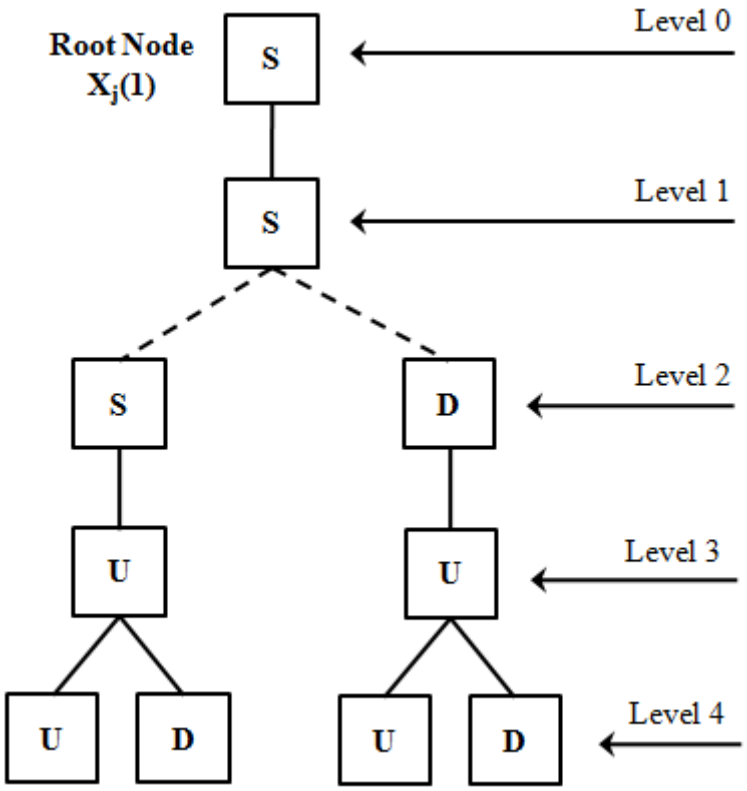

Figure 4.10 Generation from X_j = *[U, S, D, S]*

Table 4.13 Transition Probabilities from X_j = *[U, S, D, S]*

From	To	Transition Probability
X_j = [U, S, D, S]	$Y_{j,1}$ = [U, U, S, S]	$P_{j,1}$ = qb[4] · 1 · r[2] · 1
X_j = [U, S, D, S]	$Y_{j,2}$ = [D, U, S, S]	$P_{j,2}$ = q[4] · 1 · r[2] · 1
X_j = [U, S, D, S]	$Y_{j,3}$ = [U, U, D, S]	$P_{j,3}$ = qb[4] · 1 · rb[2] · 1
X_j = [U, S, D, S]	$Y_{j,4}$ = [D, U, D, S]	$P_{j,4}$ = q[4] · 1 · rb[2] · 1

Notation:

$P_{j,k}$ is the one-step transition probability from the current period system state to the next period system state.

4.7 Verification of the Results

In this study, all the computations based on the tabular method were developed in MATLAB. The algorithm generates the system state transitions and transition probabilities, and then steady-state probabilities were calculated. In addition, the tree expansion algorithm was also coded in Linux to verify the exact solutions offered by the backward formulation. It is important to indicate that the results of the two algorithms verify each other. The same transition probability matrices were constructed; hence identical steady-state probabilities were obtained from both algorithms. This shows that the computational effort devoted to this project were verified.

CHAPTER 5

SYSTEM PERFORMANCE MEASURES

5.1 Starvation and Blockage Probability

The starvation and blockage probabilities are important because they indicate the situations where machines are idle, which is an undesirable situation for the production lines.

Notation:

$\pi\ (X_j)$ is the steady-state probability of the $(i)^{th}$ station being in one of the station state conditions at the $(j)^{th}$ system state.

The starvation probability of machine *(i)* in the long-run is computed as follows:

$$\sum_{j=1}^{2^{2N-1}} \pi\left(X_j \backslash X_j\,(i) = S\right) \quad for \quad i = 1, 2, \ldots, N$$

Furthermore, the blockage probability of machine *(i)* in the long-run is given by:

$$\sum_{j=1}^{2^{2N-1}} \pi\left(X_j \backslash \left(X_j(i) \in \{B, DB\}\right)\right) \quad for \quad i = 1, 2, \ldots, N$$

The results for the starvation and blockage probabilities for stations were gathered while analyzing exact results of production lines as data sample in Appendix B displays.

5.2 Production Rate

The most essential performance measure is the production rate. The system under study, with serial-connected stations without intermediate buffer is efficient only when all stations are up and operating. These types of systems provide a lower bound on the production rate of systems with buffer as discussed by Tan (1997).

The expected production rate can be calculated by the sum of steady-state probabilities of all system states where the final station X_j *(1)* is up and operating. The production rate of an *N*-station line is given by:

$$\mathrm{PR}(N) = \sum_{j=1}^{2^{2N-1}} \pi\left(X_j \backslash X_j\,(1) = U\right)$$

Remark: Since there are no buffers involved, all the stations in the line have the same expected production rate which defines the expected production rate, *i.e.:*

$$\mathrm{PR}(N) = \sum_{j=1}^{2^{2N-1}} \pi\left(X_j \backslash X_j\,(i) = U\right) \quad for \;\; i = 1, 2, \ldots, N$$

5.3 Work-In-Process

Another important performance measure is the work-in-process, which is important especially for production lines with discrete items like automobile, fridge, and others. For an *N*-station line without buffer, the upper bound on the work-in-process is *N*, which is the situation where all machines are occupied with one work piece.

Notation:

θ_j is the total number of stations of X_j, where $X_j(i) \in \{U, B, DB\}$, and $1 \leq i \leq N$.

The work-in-process of an *N*-station line is computed as:

$$\mathrm{WIP}(N) = \sum_{j=1}^{2^{2N-1}} \theta_j \cdot \pi\left(X_j\right)$$

In order to see the occupancy of the stations in a production line, work-in-process of every station was analyzed. The work-in-process for station *(i)* for an *N*-station production line is calculated as:

$$\mathrm{WIP}_i(N) = \sum_{j=1}^{2^{2N-1}} \pi\Big(X_j \backslash \big(X_j(i) \in \{U, B, DB\}\big)\Big) \quad for \;\; i = 1 \leq i \leq N$$

Another measure for work-in-process that averages the expected work-in-process for *N*-station production lines are defined as:

Expected Occupancy = [*WIP*(*N*)] / *N*.

CHAPTER 6

ANALYSIS OF THE SYSTEM BEHAVIOR

6.1 Computing the Steady-State Probabilities

Having obtained the transition probability matrix, one can solve a set of linear equations to obtain the steady-state probabilities of the system states. More information on how the steady-state equations were solved can be found in the MATLAB code giving in Appendix A. The transition probability matrix generation was coded by tabular algorithm code and the long-run probabilities were achieved.

The transition probability matrix dimensions grow substantially for longer lines. Yet, for single solutions, the algorithm computational times are in the order of minutes for lines up to six stations. For more than six stations, MATLAB is out of memory for a computer with 3GB of physical memory. In automotive production typically lines consist of five-station zero buffer sub-lines connected with a buffer to another sub-line, thus, the results are useful for car body shop sub-lines.

6.2 Analysis of the Results

The performance measures were collected for specific failure and repair probabilities, a sample of the collected data is provided in Appendix B. For the failure probability, the interval of [0.001, 0.1] and for the repair probability, the interval of [0.01, 0.95] were taken as the interval of interest. The study includes three-station, four-station and five-station lines with identical machines as those represent the sub-lines in automotive production lines.

Notation:

q is the failure probability of every station *1, 2, .. , N.*

r is the repair probability of every station *1, 2, .. , N.*

The performance analysis was realized based on exact results collected from the results of the algorithm. A sample of the collected exact solutions for three-station lines is given in Appendix B.

6.2.1 Analysis of Production Rate

Production rate analysis is the most critical in the analysis as a measure for efficiency. Relationship between the production rate and the repair probability and the failure probability is visualized in Figure 6.1 for different failure probabilities.

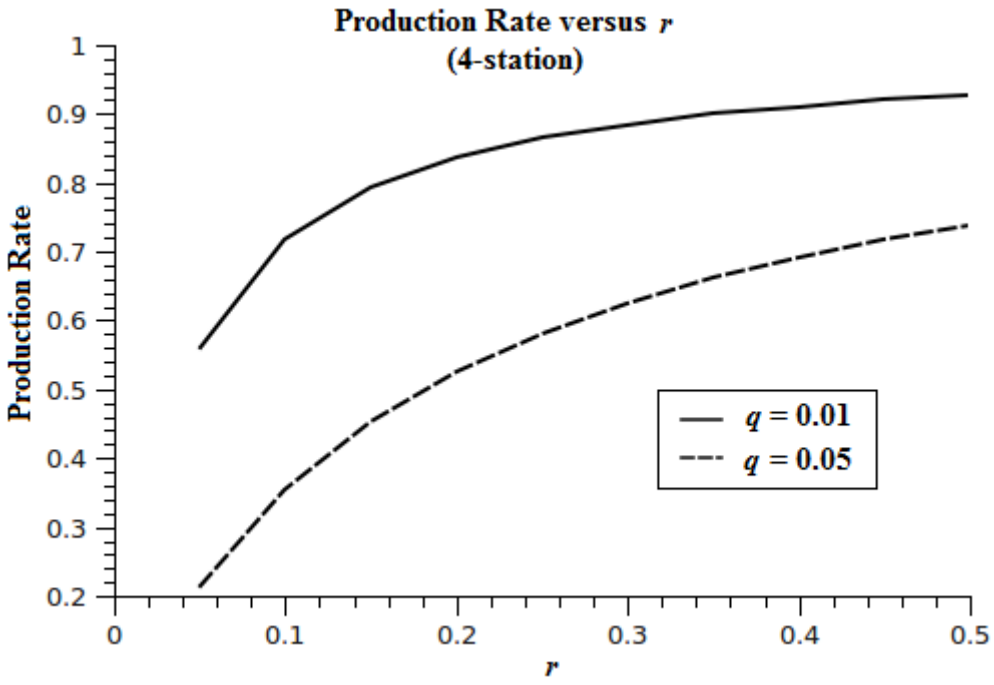

Figure 6.1 Production Rate versus Repair Probability

It is clear from Figure 6.1 that, as the repair probability increases, the production rate also increases, however, as the failure probability increases the production rate decreases as well. Moreover it was shown that as the number of stations increases in a production line, the production rate decreases relatively as depicted in Figure 6.2.

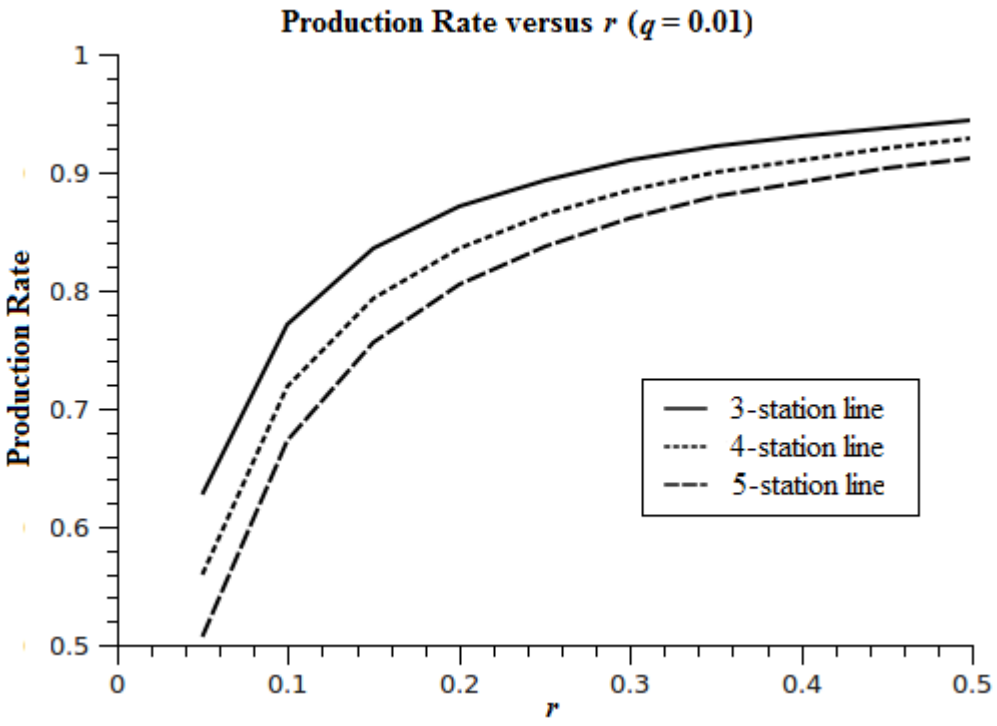

Figure 6.2 Expected Production Rates

6.2.2 Analysis of Work-In-Process

Based on the exact results, for the work-in-process measure:

- For 3-station lines, mean is 2.434 and variance is 0.193.
- For 4-station lines, mean is 3.199 and variance is 0.322.
- For 5-station lines, mean is 3.952 and variance is 0.480.

There is a negative slope relation between the failure probability and the work-in-process, whereas a positive relation exists with the repair probability. The situation is demonstrated in Figure 6.3.

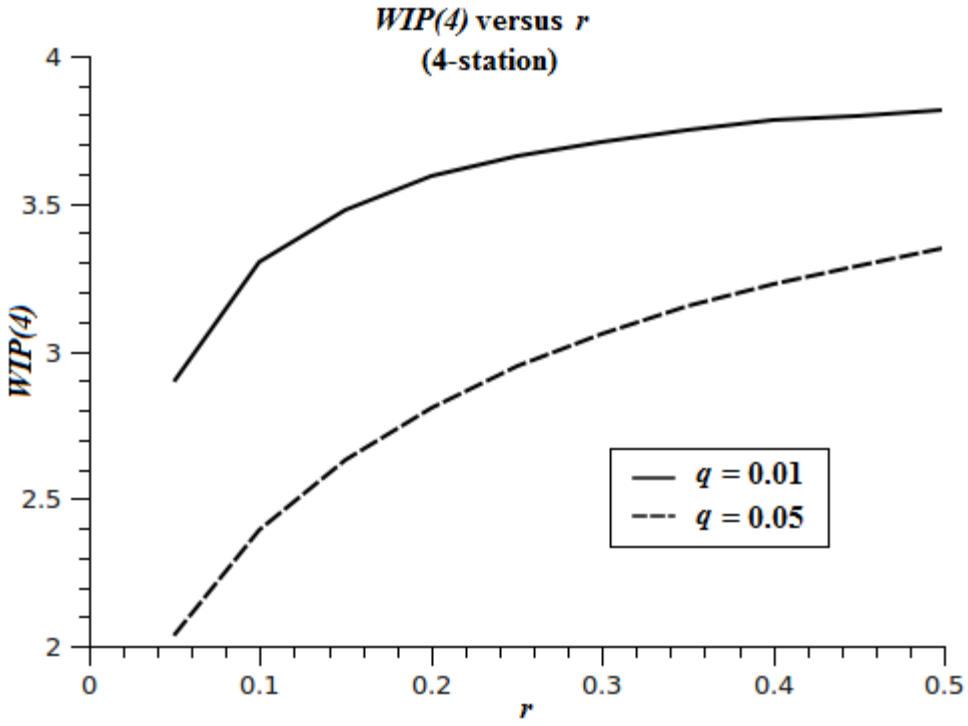

Figure 6.3 Work-In-Process versus Repair Probability

As the number of stations increases in a production line, the number of work pieces in the line increases relatively. The situation is depicted in Figure 6.4.

In addition, for a four-station line, work-in-process of every station, *i.e.* $WIP_i\,(N)$ for $1 \leq i \leq 4$, is demonstrated in Figure 6.5. As Figure 6.5 suggests, the stand-alone occupancy of the stations increase while going towards the beginning of the line, this is basically due to the infinite supply assumption.

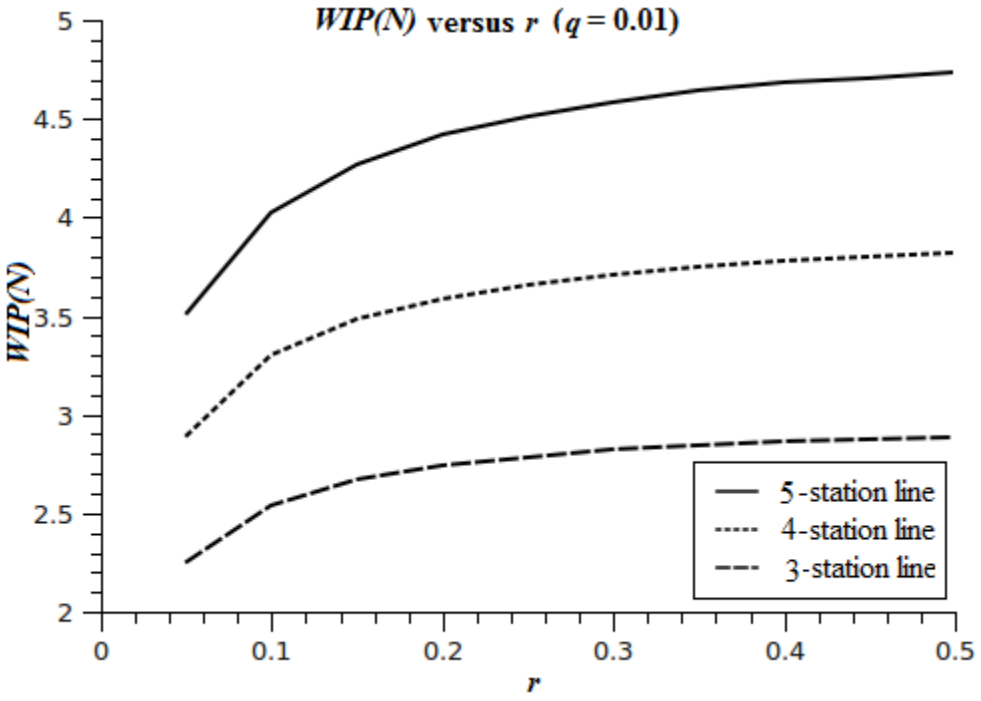

Figure 6.4 Expected Work-In-Process

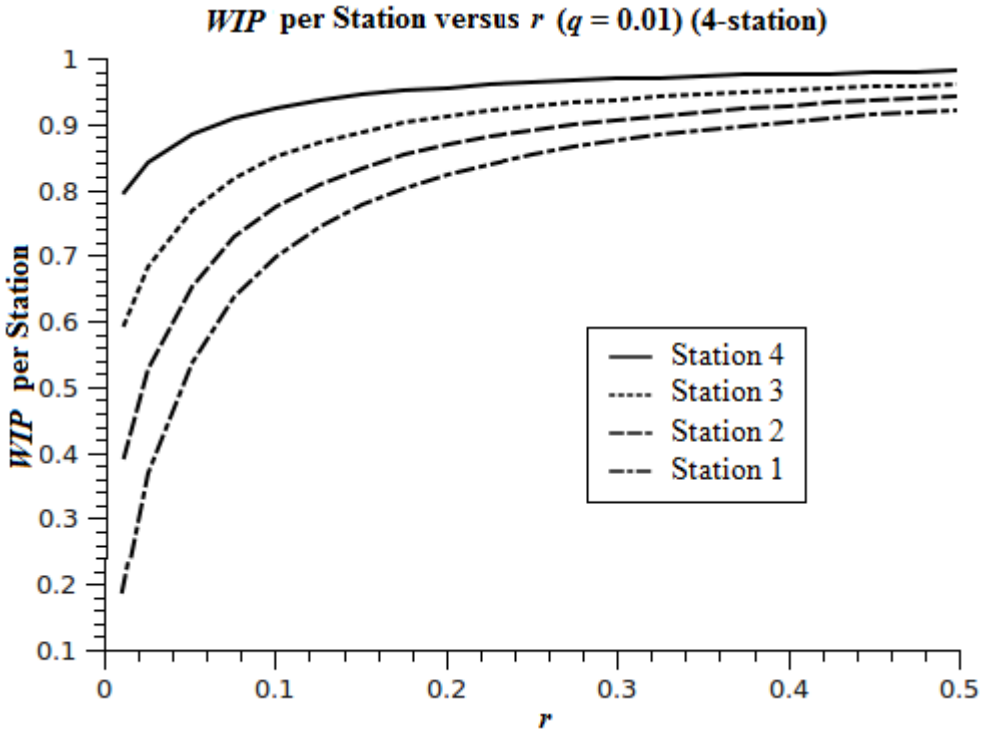

Figure 6.5 Work-In-Process per Station

As Figure 6.5 suggests, the work-in-process for the station 4 is the highest. This is reasonable as it is the first station and an infinite supply was assumed for the system. Respectively, the work-in-process of each station decreases gradually towards the end of the production line. The last station, station 1, has the lowest work-in-process.

Note that for an *N*-station line, the total of the work-in-process per station, $WIP_i\,(N)$, gives the expected work-in-process value for the entire production line. For the stations demonstrated in Figure 6.5, the total was illustrated in Figure 6.4.

Moreover, to indicate the expected occupancy and vacancy of the stations, Figure 6.6 was prepared. It is shown that the expected occupancy of the stations enlarges with an increase in the repair probability. Furthermore, the expected occupancy is slightly higher for shorter lines compared to the longer lines.

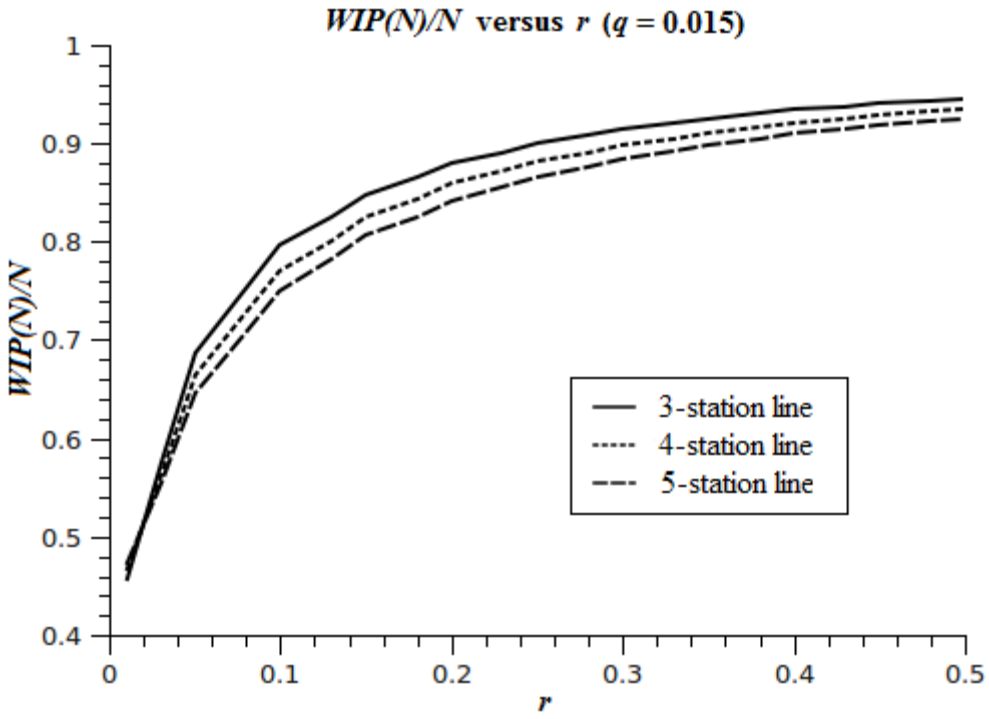

Figure 6.6 Expected Occupancy versus Repair Probability

CHAPTER 7

NUMERICAL ANALYSIS

7.1 Analysis of Production Lines with Identical Machines

In order to develop simple and useful formulas to find the production rate and the work-in-process measures for multi-station production lines, a curve-fit analysis was conducted. The aim of this analysis is to formulate simple functions that provide forecasts of the important performance measures with a reasonable accuracy. Only production lines with identical machines were considered in this numerical analysis.

A rational function fit was realized for the production rates and the work-in-process values based on the exact solutions gathered for the performance analysis in Chapter 6 for production lines with three, four and five stations. The approximate formulas were then analyzed in terms of the forecast error. Furthermore, the mean absolute error statistics (*i.e.* mean and variance) were collected and the approximate solutions were shown to be effective. The curve-fit and the error analysis were conducted in MATLAB and the code is included in Appendix C.

7.2 Rational Equation Parameterization for the Production Rate

The production rate is the most crucial production performance measure. To depict the relation between the expected production rates of production lines and the failure and repair probabilities, three-dimensional plots were generated using MATLAB (Appendix C). The relation between the expected production rate and the failure and repair probabilities for three-station lines is illustrated in Figure 7.1.

Based on the results of the exact analysis providing a smooth surface as shown by the three-dimensional plot in Figure 7.1, curve-fitting was realized to develop a reasonably accurate equation in order to establish reliable forecasts with simple formulas. As a result; the relation of the production rate of an *N*-station production line with the failure and the repair probabilities was found to be adequately represented by a simple rational function defined as:

$$PR(N) = (a0 + a1 \cdot q + a2 \cdot r + a3 \cdot q \cdot r) / (1 + a4 \cdot q + a5 \cdot r + a6 \cdot q \cdot r) \qquad (10)$$

where *a0, a1, a2, a3, a4, a5,* and *a6* are the parameters that need to be solved based on the exact solutions.

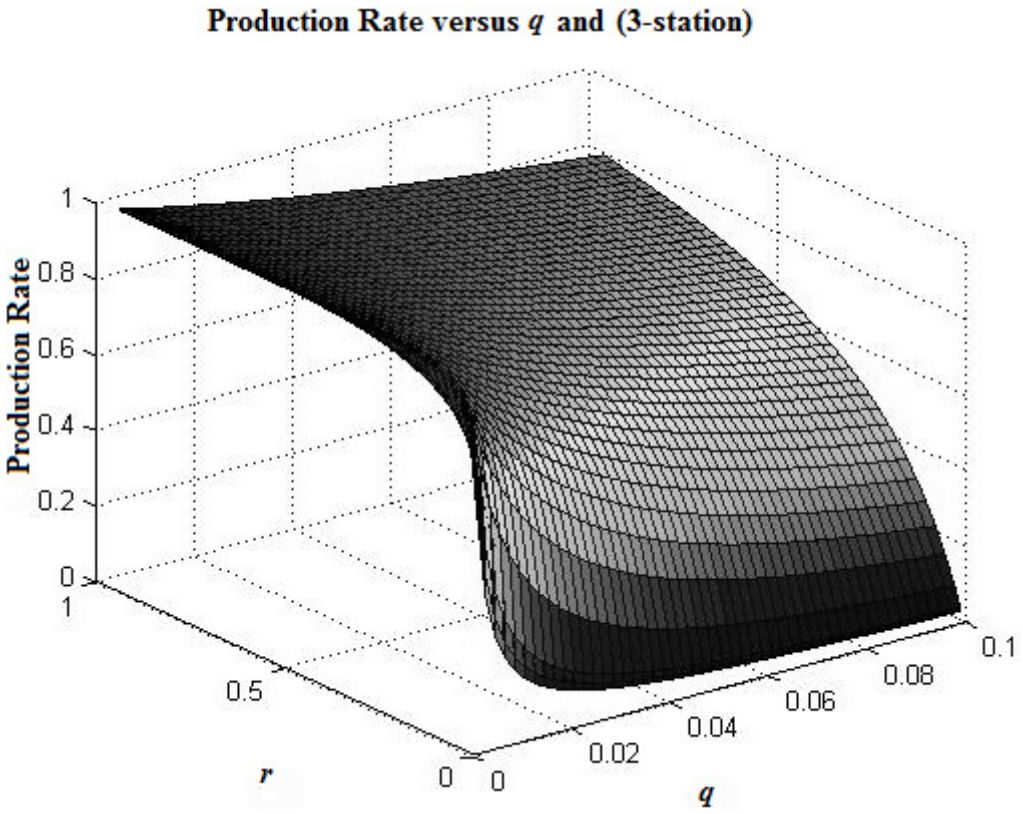

Figure 7.1 Production Rate versus *q* and *r*

Using non-linear least squares optimization routine in MATLAB, the rational function in Equation 10 was parameterized for different production lines. The found parameters were rounded and concluded to keep the acquired precision. The determined parameters are presented in the Table 7.1. Based on the acquired forecast equations, in order to analyze the accuracy of the forecast, an error analysis was realized and the results of the analysis are given in the following section.

Table 7.1 Constants for the Production Rate Function

Constants	3-Station Line	4-Station Line	5-Station Line
a0	1.9382	2.5989	2.1335
a1	-13.5482	-14.2351	-11.918
a2	10068.0462	8250.5246	4583.4131
a3	13340.8192	18240.4894	14615.8399
a4	30676.5171	33796.6908	23606.7762
a5	10082.9442	8272.8800	4601.3483
a6	11231.9814	14755.4887	11447.0396

7.2.1 Rational Function Fit Optimization Procedure

To find an approximate equation to represent adequate production rate and work-in-process forecasts for the lines, a rational function was considered. A three-dimensional function of *PR(N)* and *WIP(N)* versus *q* and *r* were constructed. With the exact results, $PR = f(q, r)$ and $WIP = f(q, r)$, obtained from the tabular method, the rational function was parameterized and an optimization procedure was executed to minimize the error between the data points and the parameterized function. The non-linear least squares optimization procedure was used as the rational function is non-linear in the parameters.

Least squares optimization uses a prior data inputs (q and r) and the related data outputs (production rate and work-in-process) to minimize the error with respect to a parameterized approximate function. The residual is: $e_{PR} = PR\ (q,\ r) - f\ (q,\ r,\ \theta)$ where $f\ (q,\ r,\ \theta)$ is the rational function with the known applicable data of q and r and the parameter vector θ. The parameter vector is computed by the non-linear least squares routine available within the MATLAB optimization toolbox in an iterative procedure. It finds optimal solution by minimizing the mean and variance of the residual error sequence in an iterative procedure.

7.2.2 Error Analysis for the Production Rate Function

In order to measure the accuracy of the obtained forecast equations for the production rate and work-in-process the error statistics have been analyzed. The used non-linear least squares optimization procedure finds an optimal solution of the parameters by minimizing the variance and the mean of the residual, *i.e.* the error was statistically minimized. Analyzing these statistics, the effectiveness of the rational functional fits for production rate and work-in-process forecast was determined.

In addition to minimum and maximum absolute error statistics, mean absolute error, the mean of error, and variance of error were collected. Furthermore, the mean absolute percent error (MAPE) and mean standard error percent (MSEP) were calculated for further insight in terms of accuracy and precision of the fit. More information about the calculations of MAPE and MSEP were given in Appendix D.

Table 7.2 summarizes the error statistics collected for the production rate forecasts based on the rational function approximations in Table 7.1. The statistics, collected in Table 7.2, show that the forecasts perform well when compared to the exact solutions in terms of mean and variance of error in addition to the absolute and percent error indicators. Figure 7.2 depicts the errors over forecasted production rate values.

Table 7.2 Production Rate Error Statistics

	3-Station Line	4-Station Line	5-Station Line
Min. Absolute Error	$2.4 \cdot 10^{-6}$	$1.3 \cdot 10^{-6}$	$2.2 \cdot 10^{-7}$
Max. Absolute Error	0.0053	0.0100	0.0127
Mean Absolute Error	$6.3 \cdot 10^{-4}$	0.0010	0.0013
Mean of Error	$-3.4 \cdot 10^{-6}$	$3.3 \cdot 10^{-6}$	$-8.0 \cdot 10^{-8}$
Variance of Error	$6.5 \cdot 10^{-7}$	$1.7 \cdot 10^{-6}$	$2.9 \cdot 10^{-6}$
Min. Abs. Percent Error	$3.1 \cdot 10^{-4}$	$1.6 \cdot 10^{-4}$	$3.4 \cdot 10^{-5}$
Max. Abs. Percent Error	0.6829	1.3935	1.9027
Mean Percent Error	-0.0073	-0.0179	-0.0279
MAPE (%)	0.1011	0.1767	0.2498
MSEP (%)	0.1583	0.2869	0.4190

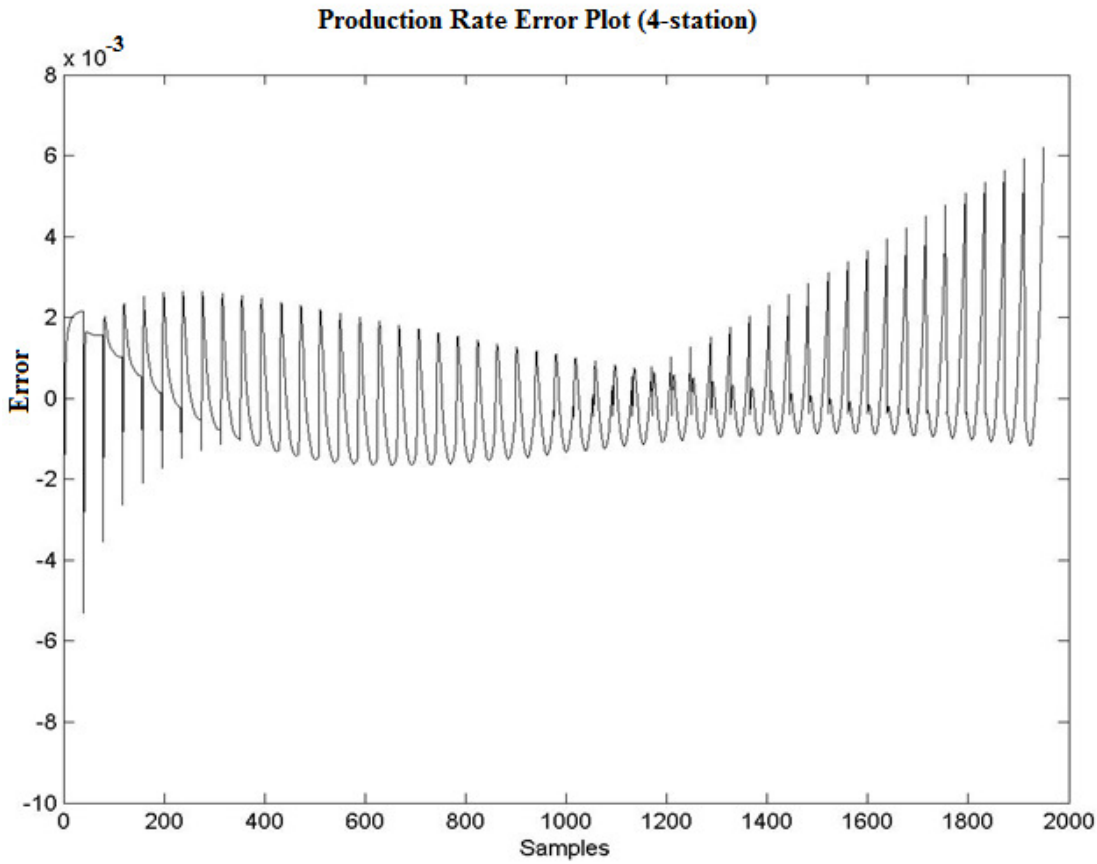

Figure 7.2 Error Plot for the Production Rate Function

As demonstrated, Figure 7.2 gives the forecast error sequence for a four-station production line versus sample number. Every sample relates to a specific combination of *q* and *r* values, as depicted in Figure 7.3. Figure 7.3 indicates that the failure probability is varied over the whole sample range, whereas the repair probability is variable over every value of *q*.

The peaks that are evident in the error plot in Figure 7.2 are due to the definition of the gathered data samples with respect to *q* and *r* as shown in Figure 7.3. Therefore, as a result of the data generation, the peaks are present in the error plots.

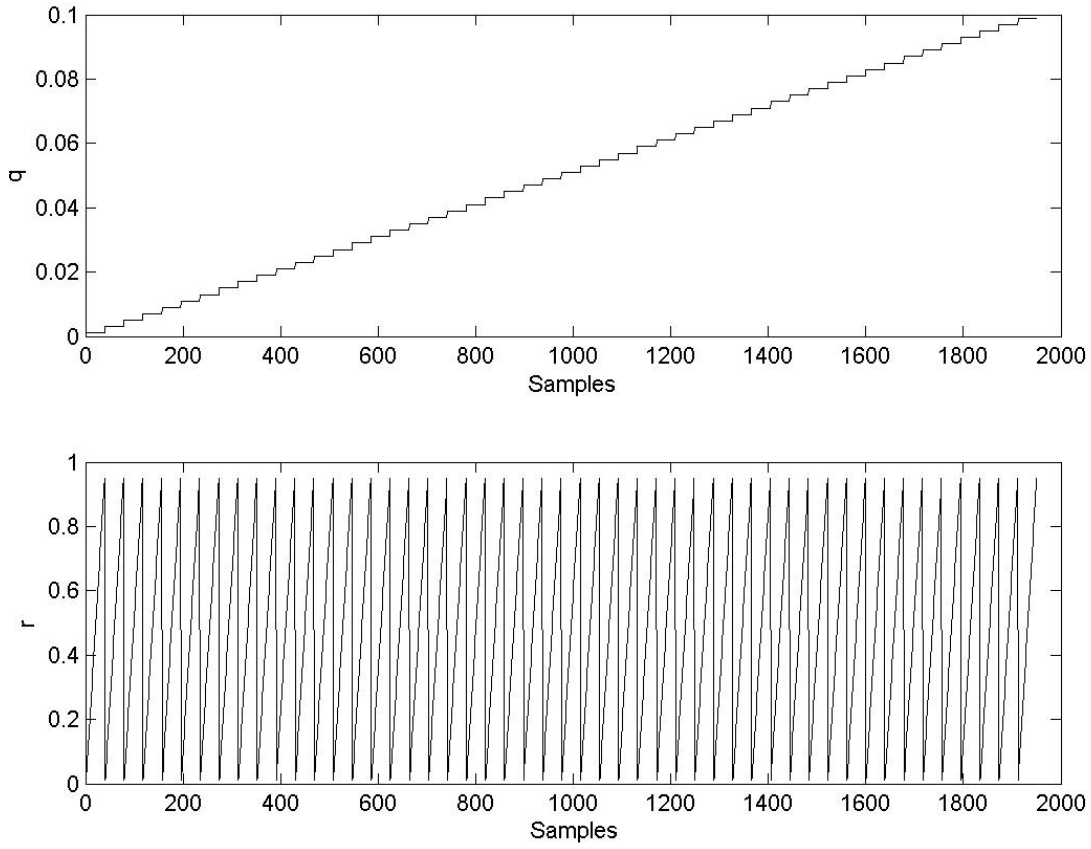

Figure 7.3 Generation of Data Sample

Note that the non-linear least squares optimization procedure minimizes the error (mean and variance) over the whole sample range using a uniform weight so every data sample has the same weight in the optimization routine. The accuracy of the obtained rational fit is considered more than adequate. As a result, the error analyses showed that the developed formulas yield very effective approximations.

7.2.3 Rational Equation Parameterization for the Work-In-Process

It is also critical to find representative forecast equations for the average work-in-process of the system in terms of the failure and repair probabilities. The three-dimensional plot that illustrates the average number of items for three-station, four-station, and five-station production lines is given in Figure 7.4.

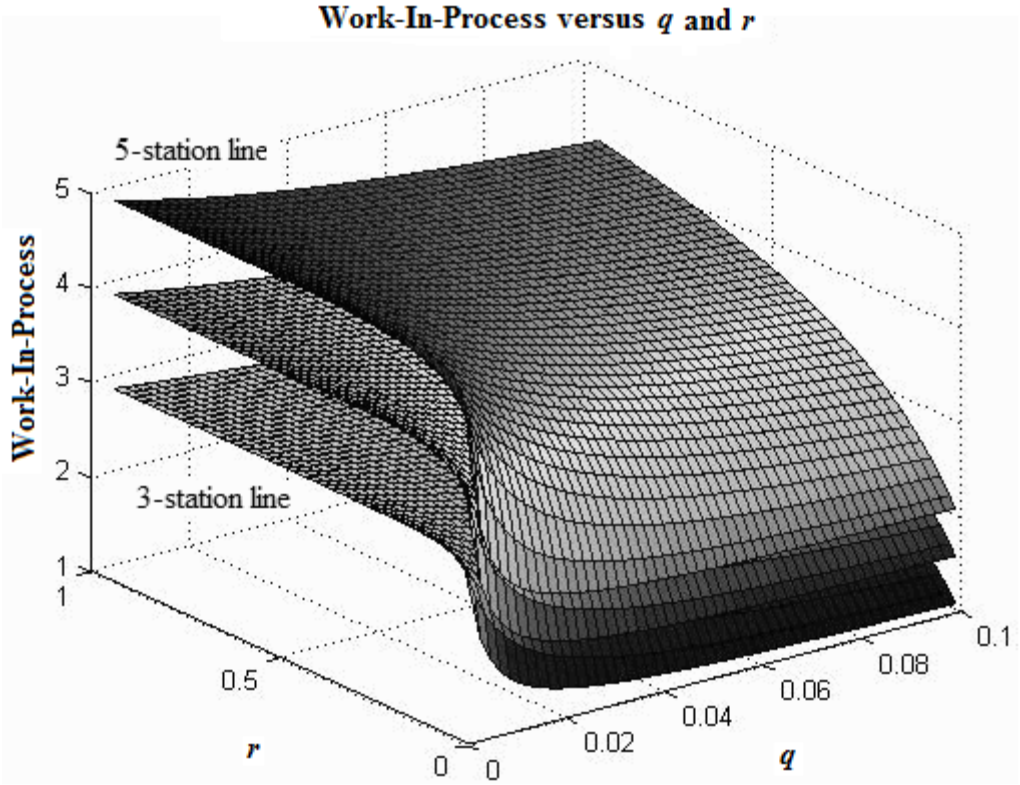

Figure 7.4 Work-In-Process versus q and r

It is clear from Figure 7.2 and 7.4 that that the production rate and the work-in-process plots suggest the use of the same rational function for curve-fitting. Hence, similar to the production rate function (10), the same parameterization was also realized for the work-in-process function.

$$WIP(N) = (b0 + b1 \cdot q + b2 \cdot r + b3 \cdot q \cdot r) / (1 + b4 \cdot q + b5 \cdot r + b6 \cdot q \cdot r) \quad (11)$$

where *b0, b1, b2, b3, b4, b5,* and *b6* are again parameters to be fitted by use of the non-linear least squares optimization routine.

Performing the same curve-fitting methodology, , the parameters in Equation 11 for different production lines were obtained as summarized in Table 7.3.

Table 7.3 Constants for the Work-In-Process Function

Constants	3-Station Line	4-Station Line	5-Station Line
b0	5.1715	14.1778	21.0819
b1	33126.9234	116504.2231	177702.1167
b2	32658.3901	75938.8111	86391.1440
b3	40948.2095	155263.8189	250044.0336
b4	33153.8861	77696.4095	88863.9427
b5	10897.0572	19017.5184	17320.6444
b6	12128.7591	33787.0170	42810.9255

7.2.4 Error Analysis for the Work-In-Process Function

Table 7.4 presents the error statistics collected for the work-in-process forecasts based on the rational function approximation in Table 7.3. Table 7.4 shows that the forecast function provides reliable estimates.

Table 7.4 Work-In-Process Error Statistics

	3-Station Line	4-Station Line	5-Station Line
Min. Absolute Error	$1.5 \cdot 10^{-6}$	$3.8 \cdot 10^{-7}$	$5.2 \cdot 10^{-6}$
Max. Absolute Error	0.0112	0.0253	0.0405
Mean Absolute Error	0.0013	0.0025	0.0040
Mean of Error	$7.1 \cdot 10^{-7}$	$6.6 \cdot 10^{-6}$	$2.4 \cdot 10^{-6}$
Variance of Error	$2.6 \cdot 10^{-6}$	$1.0 \cdot 10^{-5}$	$2.6 \cdot 10^{-5}$
Min. Abs. Percent Error	$5.4 \cdot 10^{-5}$	$1.1 \cdot 10^{-5}$	$2.3 \cdot 10^{-4}$
Max. Abs. Percent Error	0.4423	0.7687	1.0122
Mean Percent Error	$-1.6 \cdot 10^{-4}$	$1.7 \cdot 10^{-5}$	$-3.1 \cdot 10^{-4}$
MAPE (%)	0.0521	0.0790	0.1014
MSEP (%)	0.0696	0.1056	0.1354

Furthermore, error plot for the forecasts was analyzed. Figure 7.5 shows the forecast error sequence versus the sample of data points for four-station production lines.

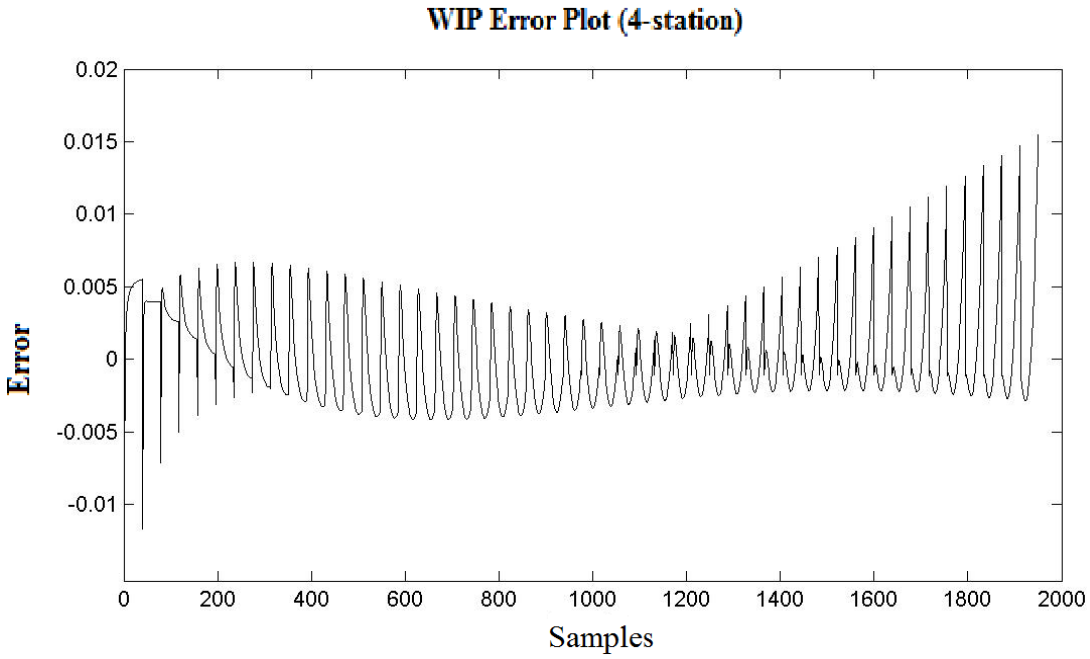

Figure 7.5 Error Plot for the Work-In-Process Function

7.2.5 Summary of the Rational-Fit Forecasting

So far the exact analysis results were gathered and it is shown that the important performance measures can also be estimated by relatively easy rational equations as a function of particular line characteristics, namely the failure and repair probabilities. Only lines with identical machines are considered for the approximated fits; hence all stations have the same failure and repair probability.

Consequently, it is shown that the obtaining rational function to forecast the expected work-in-process is effective. As the error analysis shows, the approximate parameterizations can be used most adequately and computationally more effective instead of the exact solutions for lines with identical machines.

7.3 Analysis of Balanced Lines

After the analysis of production lines with identical machines, the subsequent step is to analyze the behavior of balanced lines, where all stations have the same stand-alone availability. The stand-alone availability of a station *(i),* that is *SAA(i),* is defined by Muth and Yeralan (1981):

$$\mathrm{SAA}(i) = \frac{r[i]}{q[i] + r[i]} \quad for \quad i = 1 \leq i \leq N$$

This means that there may be different failure and repair probabilities for each machine, however that each machine still has an identical ratio. As an example, given 90% availability for a line, then the repair probability is nine times larger than the failure probability for every station.

The aim of this section is to forecast the production rate and work-in-process of the balanced lines using the numerical results of the developed methodology and MATLAB code. Based on the analysis, an upper bound and a lower bound to estimate the production rate and work-in-process of the balanced lines are defined.

- **Lower bound:** The solution of a line consisting of identical machines is used as a lower bound for the production rate and work-in-process estimates. The identical machines are chosen to have the same characteristics as the machine having the lowest failure and repair probability of the balanced line.

- **Upper bound:** Similarly, a production line with identical machines is used as an upper bound for the production rate and work-in-process estimates. In this case the identical machines are chosen to have the same characteristics as the machine having the highest failure and repair probability of the balanced line.

Regarding the above mentioned interval for the balanced lines, after several runs of the results, it is found out that the width of the interval lies within a 5% range provided that $0.001 \leq q \leq 0.1$ and $0.01 \leq r \leq 0.975$ for balanced lines with 85 to 95% availability.

Example

Given a four-station balanced line with 90% availability, the stations and the exact solutions are summarized in Table 7.5.

Table 7.5 Four-Station Balanced Line

q[4]	**q[3]**	**q[2]**	**q[1]**	**r[4]**	**r[3]**	**r[2]**	**r[1]**	***PR(4)***	***WIP(4)***
0.05	0.07	0.009	0.02	0.45	0.63	0.081	0.18	0.705	3.266

The lower and upper bound for the balanced line are calculated using the rational-fit equations developed for identical machines. Table 7.6 demonstrates the lower and upper bound for the balanced line.

Table 7.6 Lower and Upper Bounds for the Balanced Line

i = 1, 2, 3, 4	**q[i]**	**r[i]**	***PR(4)***	***WIP(4)***
Lower Bound	0.009	0.0081	0.696	3.240
Upper Bound	0.07	0.63	0.728	3.322

As suggested, the exact solution of the balanced line yields results that are within the interval defined by the identical machines as the lower and upper bounds:

- The production rate (0.705) of the balanced line lies within [0.696, 0.728].
- The work-in-process (3.266) of the balanced line lies within [3.240, 3.322].

The analysis showed that the intervals suggest a reliable range for the estimates and the wideness of the interval is small enough to give beneficial information.

7.4 Analysis of Production Lines with Different Machines

For production lines that consist of different machines, the useful failure and repair probability interval for estimating system behavior are defined based on the findings of the numerical analysis. For the production rate of line with the different machines the following lower and upper limits were observed, moreover, a point estimate within the limits was suggested for the production rate approximation:

- **Lower bound:** The solution of a balanced line is used as a lower bound for the production rate of the lines. The considered balanced line has availability, equal to the lowest availability of the stations in the line.

- **Upper bound:** Similarly, a balanced line is used to find an upper bound for the production rate estimates. The considered balanced line has availability, equal to the highest availability of the stations in the line.

- **Average Availability:** Based on the average availability of the stand-alone availabilities of the stations, an average availability approximation yields useful insights for the original line.

The balanced line approximations were performed by changing failure probability over stations while keeping the repair probability of the stations constant. This is because the production rate of the systems is more sensitive to changes in the repair probability. Henceforth, to equate the availabilities of the stations for lower and upper bounds, the failures probabilities of the stations were adjusted.

The wideness of the interval came out to be relatively larger than the ranges defined for the balanced lines in the previous section. Yet the point estimation by average availability approximation was realized to provide reliable estimates. The failure and the repair probability of the machines are in consistence with the system characteristics of the automotive plants, that is the stand-alone availabilities of the stations were chosen within the interval of 85% to 95%.

Example

For four-station lines with different machines, the system parameters and behavior were obtained using the tabular method and presented in Table 7.7.

Table 7.7 Four-Station Lines with Different Machines

Line	q[4]	q[3]	q[2]	q[1]	r[4]	r[3]	r[2]	r[1]	*PR(4)*
A	0.008	0.050	0.010	0.070	0.045	0.575	0.190	0.511	0.698
B	0.100	0.080	0.050	0.020	0.566	0.720	0.575	0.313	0.723
C	0.007	0.002	0.070	0.030	0.093	0.013	0.396	0.570	0.690

There are four different failure and repair probabilities that give way to four different stand-alone availabilities *(SAA(i))* of the stations in the line. Table 7.8 shows the stand-alone availabilities and the average availability of the lines.

Table 7.8 Stand-Alone Availability of the Stations (%)

Line	Station 4	Station 3	Station 2	Station 1	Average
A	85	92	95	88	90
B	85	90	92	94	90
C	93	87	85	95	90

The upper and lower bound of the balanced lines are calculated using the balanced lines with the lowest and highest stand-alone availability. Table 7.9 demonstrates the characteristics of the balanced lines that are used as the upper and the lower bounds for the lines that consist of different machines.

Table 7.9 Lower (LB) and Upper (UB) Bounds

Line		**q[4]**	**q[3]**	**q[2]**	**q[1]**	**r[4]**	**r[3]**	**r[2]**	**r[1]**	**LB**
A	**85%**	0.008	0.101	0.034	0.090	0.045	0.575	0.190	0.511	0.605
B	**85%**	0.099	0.127	0.101	0.055	0.566	0.720	0.575	0.313	0.636
C	**85%**	0.016	0.002	0.069	0.101	0.093	0.013	0.396	0.570	0.609
Line		**q[4]**	**q[3]**	**q[2]**	**q[1]**	**r[4]**	**r[3]**	**r[2]**	**r[1]**	**UB**
A	**95%**	0.002	0.030	0.01	0.027	0.045	0.575	0.190	0.511	0.835
B	**95%**	0.029	0.038	0.03	0.016	0.566	0.720	0.575	0.313	0.837
C	**95%**	0.005	0.001	0.021	0.030	0.093	0.013	0.3966	0.57	0.811

Remark: The exact production rate solution of the balanced lines with the applicable availabilities is then defined the lower and upper bound limits on the production rate of the line with different machines as indicated in Table 7.9.

The balanced line approximations were performed by keeping the repair probability constant while changing the failure probabilities of the lines to obtain the desired stand-alone availability of the stations, namely the lowest and highest availabilities in Table 7.8.

A useful estimate was found by assuming a balanced line with the average availability of the stand-alone availabilities of the stations as given in Table 7.8. Table 7.10 shows the balanced line approximations with the average availability. These are obtained by adjusting the failure probabilities of the stations to acquire the target availability while keeping the repair probabilities constant.

Table 7.10 Balanced Lines with Average Availability

Line		q[4]	q[3]	q[2]	q[1]	r[4]	r[3]	r[2]	r[1]	Average Availability
A	**90%**	0.005	0.064	0.021	0.057	0.045	0.575	0.190	0.511	0.704
B	**90%**	0.062	0.080	0.064	0.035	0.566	0.720	0.575	0.313	0.721
C	**90%**	0.010	0.001	0.044	0.063	0.093	0.013	0.396	0.570	0.718

In addition, Table 7.11 presents the solutions for the production rate of the balanced line approximations in Table 7.9 and Table 7.10, defining upper and lower bounds with average line approximations. The results of the balanced line assumption with the average availability yield reliable estimates for the production lines with different machines as shown in Table 7.11.

Table 7.11 Limits and Average Approximation

Line	Real *PR(4)*	Lower Bound	Upper Bound	Average Approximation
A	0.698	0.605	0.835	0.704
B	0.723	0.636	0.837	0.721
C	0.690	0.609	0.811	0.718

As suggested, the exact solutions of the lines yield results that are situated within the intervals that are defined by the balanced lines. It is clear that as the stand-alone availabilities of the stations come closer to each other, the interval for production rate estimate becomes less wide. In addition, the average availability approximation provides reliable point estimate for the exact solutions.

7.5 Further Considerations

7.5.1 Order of Stations

To analyze the relation between the system behavior and the order of the machines in a production line, necessary experiments were conducted. Table 7.12 presents example cases for different order of stations for a four-station production line.

Table 7.12 Different Order of Machines

q[4]	q[3]	q[2]	q[1]	r[4]	r[3]	r[2]	r[1]	*PR(4)*	*WIP(4)*
0.008	0.050	0.010	0.070	0.051	0.453	0.115	0.511	0.679715	3.163381
0.050	0.010	0.070	0.008	0.453	0.115	0.511	0.051	0.679534	3.269736
0.010	0.070	0.008	0.050	0.115	0.511	0.051	0.453	0.679618	3.227851
0.070	0.008	0.050	0.010	0.511	0.051	0.453	0.115	0.679833	3.135742

Table 7.12 shows that the order of the machines in the line does not have any effect on the performance measures of the system. The main reason is the availabilities of the stations are close to each other, namely within 85% to 95% availability.

7.5.2 Mean First Passage Times

It is also essential to figure out the first passage times between system states. This gives beneficial insights of how systems behavior evolves. The expected number of transitions before the system reaches the state *(j),* given that the system is currently in state *(i)* is defined as the mean passage time from *(i)* to state *(j).*

Notation:
$m_{i,j}$ is the mean first passage time from system state X_i to state X_j.

The mean first passage time equations are formulated as:

$$m_{i,j} = 1 + \sum_{k \neq j} P_{i,k} \cdot m_{k,j}$$

$$m_{i,i} = \frac{1}{\pi_i}$$

where $P_{i,k}$ is the transition probability from system state X_i to $X_{k.}$ and π_i is the steady-state probability of the system state X_i.

The set of linear equations were generated and then solved within MATLAB. The code generates all relevant mean first passage times between system states and is included in Appendix E.

To exemplify, the simplest case of two-station line was considered. All applicable system states *(2^3 = 8)* are summarized in Table 7.13.

Table 7.13 System States

X_1 = [D,D]	X_5 = [U,U]
X_2 = [D,U]	X_6 = [U,S]
X_3 = [D,S]	X_7 = [B,D]
X_4 = [U,D]	X_8 = [DB,D]

The corresponding *8x8* transition probability matrix generated by the tabular algorithm is given in Table 7.14 given *q[2] = 0.009, r[2]=0.4* and *q[2] = 0.009, r[2]=0.4*. The steady-state probabilities calculated for the system is in Table 7.15.

Table 7.14 The Transition Probability Matrix

	X_1	**X_2**	**X_3**	**X_4**	**X_5**	**X_6**	**X_7**	**X_8**
X_1	0.3	0	0.3	0.2	0	0.2	0	0
X_2	0.03	0	0.57	0.02	0	0.38	0	0
X_3	0	0	0.6	0	0	0.4	0	0
X_4	0	0.0045	0	0	0.4955	0	0.4955	0.0045
X_5	0	0.00855	0	0	0.94145	0	0.04955	0.00045
X_6	0	0.009	0	0	0.991	0	0	0
X_7	0	0	0	0	0.5	0	0.5	0
X_8	0	0.3	0	0	0.2	0	0.2	0.3

Table 7.15 Steady State Probabilities

X_1	**X_2**	**X_3**	**X_4**	**X_5**	**X_6**	**X_7**	**X_8**
0.00033	0.0078	0.01136	0.00022	0.88407	0.00758	0.08806	0.00057

In addition, Figure 7.16 presents the *8x8* matrix representing the mean first passage times between system states that is generated by the developed program.

Table 7.16 Mean First Passage Times

	$\mathbf{X_1}$	$\mathbf{X_2}$	$\mathbf{X_3}$	$\mathbf{X_4}$	$\mathbf{X_5}$	$\mathbf{X_6}$	$\mathbf{X_7}$	$\mathbf{X_8}$
$\mathbf{X_1}$	2991.86	128.56	126.08	3207.12	3.81	40.59	21.30	2504.50
$\mathbf{X_2}$	4145.52	128.22	91.80	4360.78	3.56	6.31	23.94	2507.15
$\mathbf{X_3}$	4273.70	128.18	87.99	4488.96	3.53	2.50	24.24	2507.44
$\mathbf{X_4}$	4272.53	127.01	218.81	4487.79	2.02	133.31	11.44	2494.65
$\mathbf{X_5}$	4271.34	125.81	217.62	4486.59	1.13	132.12	20.71	2503.91
$\mathbf{X_6}$	4271.20	125.68	217.49	4486.46	1.03	131.99	21.74	2504.94
$\mathbf{X_7}$	4273.34	127.82	219.62	4488.59	2.00	134.12	11.35	2505.91
$\mathbf{X_8}$	4219.41	73.89	165.70	4434.67	3.52	80.21	17.61	1755.11

As seen from Table 7.15, the minimum mean first passage time between system states is $m_{6,5} = 1.03$ and the maximum is $m_{3,4} = 4488.96$.

As a result, by the help of the transition matrix generation steady-state solutions and, mean first passage times can be analyzed for multi-station production lines with no intermediate buffers.

CHAPTER 8

CONCLUSIONS

8.1 Concluding Remarks

This study is focused on the exact analysis of the system behavior of multi-station production lines without intermediate buffer. Production lines that are operated under a fixed-cycle-time policy, with stations subject to breakdown are modeled by discrete-time Markov chains.

Based on the methods proposed, the tabular algorithm was programmed in MATLAB, which provided exact solutions of the transition probability matrix by generating transitions with transition probabilities a given number of stations in a multi-station production line. Moreover, steady-state solutions were found and important performance measures were analyzed in order to foresee the system behavior. The results for three, four and five-station production lines were analyzed in detail.

Critical performance measures such as production rate and work-in-process measures were further analyzed to establish valid formulas in accordance with the system parameters. A curve-fit analysis was performed and valid equations were found to forecast the production rate and the work-in-process with a reasonable accuracy. Furthermore, for lines with non-identical machines, approximations were suggested and numerical analysis was realized.

To verify the results, the tree expansion and tabular algorithms were developed in different programming environments, namely Linux and MATLAB, respectively. The tree expansion algorithm was developed for an automotive body shop production system. The constructed transition probability matrices and the calculated steady-state solutions of the two algorithms yield the same results. This verifies the computational efforts devoted to this research.

8.2 Future Work

This study provided exact solutions of the steady-state probabilities by constructing the transition probability matrices of multi-station production lines without intermediate storage. In addition, lines with identical machines are curve-fitted to obtain useful approximate equations for production rate and work-in process measures.

Furthermore, the analyses of the balanced lines, where all stations have the same availability were also realized based on the exact solutions of the algorithms. Effective lower and upper bounds for the production rate and the work-in-process were suggested with acceptable range and accuracy. In addition, the cases where all machines have different failure and repair probabilities were also considered. Approximations with upper and lower bounds were suggested.

Further research can be devoted to include intermediate storage between the machines. Furthermore, the code can be expanded to allow for intermediate buffer with one unit capacity easily. That is; a machine with zero failure probability and one repair probability can act like an intermediate storage of capacity one between two stations. Consequently, a next step after this research can be the including of buffers between stations based on the developed algorithm.

REFERENCES

Gershwin, S.B. (1987), *"An efficient decomposition method for the approximate evaluation of tandem queues with finite storage space and blocking"*, Operations Research 35(2), pp. 291-305.

Gershwin, S. B. and Schick, I.C. (1983), *"Modeling and analysis of three-stage transfer lines with unreliable machines and finite buffers"*, Operations Research, 31(2), pp. 354-380.

Lipschutz, S. and Lipson, M. (1997), *"Schaum's Outline of Discrete Mathematics",* 2nd Edition, ISBN: 0-07-038045-7, McGraw Hill.

Maggio, N., Matta, A., Gershwin, S. B. and Tolio, T. (2009), *"A decomposition approximation for three-machine closed-loop production systems with unreliable machines, finite buffers and a fixed population",* IIE Transactions 41(6), pp. 562-574.

Muth, E. J. (1979), *"Models for production lines in which stations are subject to breakdown"*, Methods of Operations Research 35, pp. 345-346.

Muth, E. J. and Yeralan, S. (1981), *"Effect of buffer size on productivity of work stations that are subject to breakdown"*, Proceedings of the 20th IEEE Conference on Decision and Control.

Schick, I.C. and Gershwin, S.B. (1978), "*Modeling and analysis of unreliable transfer lines with finite interstage buffers",* Complex Material Handling and Assembly Systems, Volume IV, Report ESLFR-834-6, Electronic Systems Laboratory, Massachusetts Institute of Technology, Cambridge, MA.

Tan, B. (1997), "*Variance of the throughput of an N-station production line with no intermediate buffers and time dependent failures*", European Journal of Operational Research 101(3), pp. 560-576.

Tolio, T., Matta, A. and Gershwin, S.B. (2002), *"Analysis of two-machine lines with multiple failure modes"*, IIE Transactions 34(1), pp. 51-62.

Yeralan, S., Franck, W. and Quasem, M.A. (1986), "*A continuous materials flow production line model with station breakdown*", European Journal of Operational Research 27, pp. 289-300.

Yeralan, S. and Tan, B. (1997), *"A station model for continuous materials flow production",* International Journal of Production Research 35(9), pp. 2525–2541.

APPENDIX A

MATLAB CODE FOR THE TABULAR METHOD

% Sub-Function: tara2.h

```
function sayi=tara(array,num)

 n=length(array);

sayi = length(find(array(num:n)~=2 & array(num:n)~=3 ));
```

% Main Script:

```
close all; clear all; clc;

% Number of machines
k = 2:3;

QplusR = 0;              % if 1 then fixed 'q+r' else it uses fixed q

%Interval
r_varr = [0.01 0.025:0.025:0.95]';
q_varr = (0.001:0.002:0.099)';

Nq = size(q_varr,1);
Nr = size(r_varr,1);

% NOTE: running for variable q and r over the number of machines is
% possible by defining the columns of the q_var and r_var matrix. The rows
% presents the sequence of variables (per machine) for which the algorithm
% is run, whereas the columns give the probabilities for every machine. If
% the user wants to allow for these variable probabilities k must be scalar.

display(['counting length = ',num2str(Nq)]);

for k=k(1):k(end)
display(['Now running for ',num2str(k),' Machines']);
```

```
q_var = kron(q_varr,ones(1,k));      % makes matrix; every column a machine
r_var = kron(r_varr,ones(1,k));      % makes matrix; every column a machine

%-------------------------------------------------------------------%
%               Building the Permutation Matrix
%-------------------------------------------------------------------%

elem = [0,1,2,3,4];   % 0=D, 1=U, 2=S, 3=B, 4=DB

x = npermutek(elem,k);

rr = find(x(:,1)~=2 & x(:,k)<3);
x = x(rr,:);

n = size(x,1);
j = 1;
t = 0;
for i = 1:1:n
   for l=2:k
      if x(i,l) == 2 || x(i,l) == 1
        if x(i,l-1) ~= 3 && x(i,l-1) ~= 4
         t = t+1;
        end
      else
         t = t+1;
      end
   end

   if t == k-1
      X(j,1:k) = x(i,:);
      n1 = size(X);
      j  = n1(1)+1;
   end
   t = 0;
end

%-------------------------------------------------------------------%
%               Generating the Transition Matrix
%-------------------------------------------------------------------%

N = size(X,1);

% Obtaining 'a'; checking each row: how many of '0 or 1 or 4' there are...
for i1 = 1:N
   a(i1,:) = length(find(X(i1,:) ~= 2 & X(i1,:) ~= 3 ));
   Y(i1).Y = zeros(2^a(i1,:),k);
end
```

```
%-------------------------------------%
%         Last row, for n = k
%-------------------------------------%

for i1=1:size(X,1)     % Loop iterating possible sequences (rows of X)
  rowYi = 2^a(i1);
  if X(i1,k) == 1 ||  X(i1,k) == 0
      % ---- for only first half of the rows ----
      switch X(i1,k-1)
      case {1,4,3}
         Y(i1).Y(1:rowYi/2,k) = ones(rowYi/2,1);
      case {0,2}
         Y(i1).Y(1:rowYi/2,k) = 2*ones(rowYi/2,1);
      end
   elseif X(i1,k) == 2
      switch (X(i1,k-1))
      case {1, 4, 3}
         Y(i1).Y(1:rowYi,k) = ones(rowYi,1);

      case {0 , 2}
         Y(i1).Y(1:rowYi,k) = 2*ones(rowYi,1);
      end
  end

%-------------------------------------------------%
%         Middle rows, for n = k-1 to 2
%-------------------------------------------------%

for n = k-1:-1:2

%FIRST INNER-LOOP
if X(i1,n) == 1 || X(i1,n) == 4
 C = tara2(X(i1,:),n);
 M = 2^(a(i1)-C);
 for w=1:2*M:2^a(i1)
   if Y(i1).Y(w,n+1) == 0 || Y(i1).Y(w,n+1) == 3 || Y(i1).Y(w,n+1)== 4
     % count not starving and not blocked 'n' to 'k'
     Y(i1).Y(w:w+M-1,n) = 3*ones(M,1);
     Y(i1).Y(M+w:w+2*M-1,n) = 4*ones(M,1);
   end
   if Y(i1).Y(w,n+1) == 1
     switch X(i1,n-1)
     case {1,4,3}
        Y(i1).Y(w:w+M-1,n) = ones(M,1);
        Y(i1).Y(M+w:w+2*M-1,n) = zeros(M,1);
     case {0,2}
        Y(i1).Y(w:w+M-1,n) = 2*ones(M,1);
```

```
          Y(i1).Y(M+w:w+2*M-1,n) = zeros(M,1);
       end
    end
  end
end

% SECOND INNER-LOOP
if X(i1,n) == 0
  C = tara2(X(i1,:),n);
  M = 2^(a(i1)-C);
  for w=1:2*M:2^a(i1)
     if Y(i1).Y(w,n+1) ~= 1    % like if equals to 0,3,4, or 2
       switch X(i1,n-1)
       case {1, 4, 3}
          Y(i1).Y(w:w+M-1,n) = ones(M,1);
          Y(i1).Y(M+w:w+2*M-1,n) = zeros(M,1);
       case {0,2}
          Y(i1).Y(w:w+M-1,n) = 2*ones(M,1);
          Y(i1).Y(M+w:w+2*M-1,n) = zeros(M,1);
       end
     end
   end
end
  if X(i1,n) == 2
    switch X(i1,n-1)
    case {1, 4, 3}
       Y(i1).Y(1:2^a(i1),n) = ones(2^a(i1),1);
    case {0,2}
       Y(i1).Y(1:2^a(i1),n) = 2*ones(2^a(i1),1);
    end
end

% THIRD INNER-LOOP
if X(i1,n) == 3
  for w=1:1:2^a(i1)
     if Y(i1).Y(w,n+1) == 0 || Y(i1).Y(w,n+1) == 3 || Y(i1).Y(w,n+1)== 4
        Y(i1).Y(w,n) = 3;
     end
     if Y(i1).Y(w,n+1) == 1
        switch X(i1,n-1)
        case {1, 4, 3}
          Y(i1).Y(w,n) = 1;
        case {0, 2}
          Y(i1).Y(w,n) = 2;
        end
     end
  end
```

```
end
end  % end of the k-1 generation
%--------------------------------------%
%          First row, for n = 1
%--------------------------------------%
if X(i1,1)== 1 || X(i1,1) == 4
  for w=1:2:2^a(i1)
    if Y(i1).Y(w,2) == 0 || Y(i1).Y(w,2) == 3 || Y(i1).Y(w,2) == 4
       Y(i1).Y(w:w+1,1) = [3;4];
    end
    if Y(i1).Y(w,2) == 1
       Y(i1).Y(w:w+1,1) = [1;0];
    end
  end
end
if X(i1,1)== 0
  for w=1:2:2^a(i1)
    if Y(i1).Y(w,2) ~= 1    % like if equals to 0,3,4, or 2
        Y(i1).Y(w:w+1,1) = [1;0];
    end
  end
end
if X(i1,1)== 3
  for w=1:1:2^a(i1)
    if Y(i1).Y(w,2) == 0 || Y(i1).Y(w,2) == 3 || Y(i1).Y(w,2) == 4
        Y(i1).Y(w,1) = 3;
    end
    if Y(i1).Y(w,2) == 1
       Y(i1).Y(w,1) = 1;
    end
  end
end
Y(i1).XY=[X(i1,:);Y(i1).Y];
end    % end of loop iteration over rows of X

%-----------------------------------------------------------------------%
%               Transition Probability Matrix:
%-----------------------------------------------------------------------%
results = [];
for qi = 1:Nq    % loop iteration 'for every value of q'
for ri = 1:Nr    % loop iteration 'for every value of r'

% Defining the r and q probabilities per loop run:
r  = r_var(ri,:);
if QplusR == 1
   q = q_var(qi,:)-r;
else
```

```
    q  = q_var(qi,:);
end
qb = ones(1,k)-q;   % q_bar
rb = ones(1,k)-r;   % r_bar

for i1=1:N
    ny = size(Y(i1).Y,1);
    Y(i1).P = ones(ny,1);
    for ik = 1:k
      if X(i1,ik) == 1
         for iny = 1:ny
            switch Y(i1).Y(iny,ik)
            case {0,4}
               Y(i1).P(iny) = Y(i1).P(iny)*q(ik);
            case {1,2,3}
               Y(i1).P(iny) = Y(i1).P(iny)*qb(ik);
            end
         end
      elseif X(i1,ik) == 0
         for iny = 1:ny
            switch Y(i1).Y(iny,ik)
            case {0}
               Y(i1).P(iny) = Y(i1).P(iny)*rb(ik);
            case {1,2}
               Y(i1).P(iny) = Y(i1).P(iny)*r(ik);
            end
         end
      elseif X(i1,ik) == 4
         for iny = 1:ny
           switch Y(i1).Y(iny,ik)
           case {0,4}
               Y(i1).P(iny) = Y(i1).P(iny)* rb(ik);
           case {1,2,3}
               Y(i1).P(iny) = Y(i1).P(iny)*r(ik);
           end
         end
      end
    end
end

%-----------------------------------------------%
%         Filling in the probabilities:
%-----------------------------------------------%
T = zeros(N,N);     % Initializing the Transition Probability Matrix
for i1=1:N
    ny = size(Y(i1).Y,1);
    for i2=1:ny
```

```
      for i3=1:N
         if X(i3,:) == Y(i1).Y(i2,:)
            hm=i3;
         end
      end
      T(i1,hm) = Y(i1).P(i2);
   end
end

%----------------------------------------%
%        Solving phi * T = phi
%              phi * (T-I) = 0
%----------------------------------------%

TI = T-eye(N);
% replacing one column of T-I with the condition sum(phi_i) = 1
A = TI;  A(:,1)=ones(N,1);
rhs = [1 zeros(1,N-1)];

% Computing the steady-state solutions (phi)
phi = rhs*A'*inv(A*A');        % Linear Least Squares; assuming A*A' nonsingular

% mathematical checks
sumone(qi,ri) = sum(phi);             % should be 1
isnul(qi,ri) = max(phi - phi*T);      % should be 0 (close to zero)

% Save results in the Structure Q:
Q(qi).Res(ri).X = X;            % Current States
Q(qi).Res(ri).Y = Y;            % Transition States
Q(qi).Res(ri).T = T;            % Transition Propability Matrix
Q(qi).Res(ri).phi = phi;        % Steady State Propability
Q(qi).Res(ri).M = MFPT(T,phi);  % Mean First Passage Time

Q(qi).qs = q_var(qi,:);
Q(qi).rs = r_var;

%-----------------------------------------------------------------%
%             Computing the Production Rates and WIP:
%-----------------------------------------------------------------%

% Production Rates:
for i = 1:k
   indices = find(X(:,i)==1);
   indices2 = find(X(:,i)==2);
   indices3 = find(X(:,i)==3 | X(:,i)==4);
   indices4 = find(X(:,i)==1 | X(:,i)==3 | X(:,i)==4);
   Q(qi).ProdRates(ri,i) = sum(phi(indices));
```

```
    Q(qi).StarvRates(ri,i) = sum(phi(indices2));
    Q(qi).BlockRates(ri,i) = sum(phi(indices3));
    Q(qi).WIP(ri,i) = sum(phi(indices4));
end

% Total WIP (as the k+1 column):
Q(qi).WIP(ri,k+1) = sum(Q(qi).WIP(ri,:)')';

% QR = [q^3 q^2 q ];

end   % end loop r varialbes

tempres = [kron(ones(Nr,1),q_var(qi,:)) r_var(:,:) Q(qi).ProdRates(:,1) Q(qi).WIP...
          Q(qi).StarvRates Q(qi).BlockRates];
results = [results ; tempres];
display(['count ',num2str(qi)]);

end   % end loop q variables

svstr = ['HH_',num2str(k),'Machines.mat'];
save(svstr,'Q','results','q_var','r_var','QplusR');
display(['Saved for k=',num2str(k)]);

save temp.mat k Nq Nr q_varr r_varr
clear all
load temp.mat
load(['HH_',num2str(k),'Machines.mat']);

end
```

APPENDIX B

SAMPLE OF EXACT ANALYSIS RESULTS

Table A.1 Sample Results

				Starvation Probability			Blockage Probability		
q	**r**	***PR(3)***	***WIP(3)***	**m/c 3**	**m/c 2**	**m/c 1**	**m/c 3**	**m/c 2**	**m/c 1**
0.01	0.05	0.628	2.256	0	0.123	0.247	0.248	0.124	0
0.01	0.10	0.771	2.543	0	0.076	0.151	0.152	0.076	0
0.01	0.15	0.835	2.670	0	0.054	0.109	0.110	0.055	0
0.01	0.20	0.871	2.742	0	0.043	0.085	0.086	0.043	0
0.01	0.25	0.894	2.788	0	0.035	0.070	0.071	0.035	0
0.01	0.30	0.910	2.820	0	0.030	0.059	0.060	0.029	0
0.01	0.35	0.922	2.844	0	0.026	0.052	0.052	0.026	0
0.01	0.40	0.931	2.862	0	0.023	0.046	0.046	0.022	0
0.01	0.45	0.938	2.877	0	0.020	0.041	0.041	0.020	0
0.01	0.50	0.944	2.888	0	0.018	0.037	0.037	0.018	0
0.02	0.05	0.460	1.921	0	0.177	0.355	0.359	0.180	0
0.02	0.10	0.631	2.261	0	0.121	0.243	0.246	0.123	0
0.02	0.15	0.719	2.439	0	0.092	0.185	0.187	0.093	0
0.02	0.20	0.774	2.547	0	0.074	0.149	0.151	0.075	0
0.02	0.25	0.811	2.621	0	0.062	0.125	0.126	0.063	0
0.02	0.30	0.837	2.674	0	0.053	0.107	0.108	0.054	0
0.02	0.35	0.857	2.714	0	0.047	0.094	0.095	0.047	0
0.02	0.40	0.873	2.746	0	0.042	0.084	0.085	0.042	0
0.02	0.45	0.885	2.771	0	0.037	0.075	0.076	0.038	0
0.02	0.50	0.896	2.791	0	0.034	0.068	0.069	0.034	0
0.03	0.05	0.365	1.730	0	0.206	0.416	0.422	0.212	0
0.03	0.10	0.535	2.071	0	0.151	0.304	0.309	0.155	0
0.03	0.15	0.634	2.267	0	0.119	0.240	0.244	0.122	0
0.03	0.20	0.698	2.395	0	0.098	0.198	0.201	0.101	0

APPENDIX C

MATLAB CODE FOR CURVE-FITTING AND 3D-PLOTTING

```
clear all; close all; clc

global qry unity

% number of stations:
k = 3;

% load saved data from the main file
svstr = ['HH_',num2str(k),'cMachines.mat'];
load(svstr);


%-------------------------------------------------------------%
%    LSQ NonLIN Curve-fit optimization routine   %
%-------------------------------------------------------------%

display([num2str(k),' Machines']);

% The Function:
% y = (b0 + b1q + b2r + b3qr)/(1 + b4q + b5r + b6qr)
% y = b0 + b1q + b2r + b3qr - b4qy - b5ry - b6qry
% NOTE: y = PR or y = WIP, b=[b0,....b6] = b(1),...b(7)

ii = 0;
for qi=1:length(q_var(:,1))
    for ri=1:length(r_var(:,1))
        q = q_var(qi,1);
         r = r_var(ri,1);
         ii = ii+1;

        PR(ii,:)  = Q(qi).ProdRates(ri,1);
        WIP(ii,:) = Q(qi).WIP(ri,1);

        qr(ii,:) = [q r];
        Xq(qi) = q;
        Yr(ri) = r;
```

```
% Matrices used for 3D plots:
        X_q(ii) = q;
        Y_r(ii) = r;
        Z_prod(ri,qi) = Q(qi).ProdRates(ri,1);
        Z_wip(ri,qi) = Q(qi).WIP(ri,1);
   end
end

% summary of results:
totalres = [qr PR WIP];

% starting position of the optimization routine
x0 = [1 1 1 1 1 1 1];

% used options set for the leoptimization
options=optimset('MaxIter',100);
options=optimset(options,'MaxFunEvals',3000);
options=optimset(options,'TolFun',1e-6);
options=optimset(options,'LargeScale','off');
options=optimset(options,'LevenbergMarquardt','on');
options=optimset(options,'Jacobian','off');

% Running the optimization problem for PR:
unity = 1;
qry = [qr PR];
[xpr ep] = lsqnonlin('HRationalfit',x0,[],[],options);

% round solution to 4 decimals
xpr = round(xpr*10000+0.0055)/10000;

% Mean and Variance of PR:
MeanPR = sum(PR)/length(PR);
VarPR = PR'*PR/length(PR)-MeanPR^2;

% Compute the stochastic characteristics of the error (PR):
errPR = HRationalfit(xpr);
MinAbsErrPR = min(abs(errPR))
MaxAbsErrPR = max(abs(errPR))

MeanErrPR = mean(errPR)
MeanAbsErrPR = mean(abs(errPR))

% percentage error:
PerErrPR = errPR./PR*100;
MinAbsPerErrPR = min(abs(PerErrPR))
MaxAbsPerErrPR = max(abs(PerErrPR))
MeanPerErrPR = mean(PerErrPR)
```

```
MAPE_PR = mean(abs(PerErrPR))


% Variance and MSEP:
VarErrPR = errPR'*errPR/length(errPR)-MeanErrPR^2
MSEP_PR = mean(sqrt(VarErrPR)./PR*100)

f10 = figure(10);
subplot(2,1,1);
plot(qry(:,1),'k');
xlabel('Samples');
ylabel('q');
subplot(2,1,2);
plot(qry(:,2),'k');
xlabel('Samples');
ylabel('r');

f1 = figure(1);
plot(PerErrPR,'k');
title(['Percent Error of PR for ',num2str(k),' Machines']);
ylabel('e (%)');
xlabel('Samples');
f2 = figure(2);
plot(errPR,'k');
title(['Error for PR for ',num2str(k),' Machines']);
ylabel('e');
xlabel('Samples');


% Running the optimization problem for WIP:
unity = 1;
qry = [qr WIP];
[xwip ew] = lsqnonlin('HRationalfit',x0,[],[],options);

% round to 4 decimals
xwip = round(xwip*10000+0.0055)/10000;

% Mean and Variance of WIP:
MeanWIP = sum(WIP)/length(WIP);
VarWIP = WIP'*WIP/length(WIP)-MeanWIP^2;

% Compute the stochastic characteristics of the error (WIP):
errWIP = HRationalfit(xwip);
MinAbsErrWIP = min(abs(errWIP))
MaxAbsErrWIP = max(abs(errWIP))
```

```
MeanErrWIP = mean(errWIP)
MeanAbsErrWIP = mean(abs(errWIP))

% percentage error (WIP:
PerErrWIP = errWIP./WIP*100;
MinAbsPerErrWIP = min(abs(PerErrWIP))
MaxAbsPerErrWIP = max(abs(PerErrWIP))
MeanPerErrWIP = mean(PerErrWIP)
MAPE_WIP = mean(abs(PerErrWIP))

% Variance and MSEP (WIP):
VarErrWIP = errWIP'*errWIP/length(errWIP)-MeanErrWIP^2
MSEP_WIP = mean(sqrt(VarErrWIP)./WIP*100)

f3 = figure(3);
plot(PerErrWIP,'k');
title(['Percent Error of WIP for ',num2str(k),' Machines']);
ylabel('e (%)');
xlabel('Samples');
f4 = figure(4);
plot(errWIP,'k');
title(['Error for WIP for ',num2str(k),' Machines']);
ylabel('e');
xlabel('Samples');

saveas(f10,['qrVSsamples.jpg'],'jpg');
saveas(f1,['PR_',num2str(k),'MC_PercentErr.jpg'],'jpg');
saveas(f2,['PR_',num2str(k),'MC_AbsErr.jpg'],'jpg');
saveas(f3,['WIP_',num2str(k),'MC_PercentErr.jpg'],'jpg');
saveas(f4,['WIP_',num2str(k),'MC_AbsErr.jpg'],'jpg');

%-----------------------------------------------%
%                 3D PLOTS                      %
%-----------------------------------------------%

Y_r = r_var(:,1)';
X_q = q_var(:,1)';
for qi=1:length(q_var)
    for ri=1:length(r_var)
      q = q_var(qi,1);
      r = r_var(ri,1);
     Z_xpr(ri,qi) = (xpr(1)+xpr(2)*q+xpr(3)*r+xpr(4)*q*r)/(unity...
                    + xpr(5)*q+xpr(6)*r+xpr(7)*q.*r);
       Z_xwip(ri,qi) = (xwip(1)+xwip(2)*q+xwip(3)*r+xwip(4)*q*r)/(unity...
                      + xwip(5)*q+xwip(6)*r+xwip(7)*q.*r);
    end
end
```

```
f5 = figure(5);
surf(X_q,Y_r,Z_prod)
title(['Production Rate versus q and r (',num2str(k),'-station)']);
xlabel('q')
ylabel('r')
zlabel('Production Rate')

f6 = figure(6);
surf(X_q,Y_r,Z_xpr)
title(['Forecast PR versus q and r (',num2str(k),'-station)']);
xlabel('q')
ylabel('r')
zlabel('Production Rate')

f7 = figure(7);
surf(X_q,Y_r,Z_wip/k)
title(['WIP/N versus q and r (',num2str(k),'-station)']);
xlabel('q')
ylabel('r')
zlabel('WIP/N')

f8 = figure(8);
surf(X_q,Y_r,Z_xwip/k)
title(['Forecast WIP versus q and r (',num2str(k),'-station)']);
xlabel('q')
ylabel('r')
zlabel('WIP/N')

saveas(f5,['PR_',num2str(k),'MC_3Dplot.jpg'],'jpg');
saveas(f6,['PRest_',num2str(k),'MC_3Dplot.jpg'],'jpg');
saveas(f7,['WIP_',num2str(k),'MC_3Dplot.jpg'],'jpg');
saveas(f8,['WIPest_',num2str(k),'MC_3Dplot.jpg'],'jpg');

% HRationalfit function code


function e = HRationalfit(x)

% The Function:
% y = (b0 + b1q + b2r + b3qr)/(b4 + b5q + b6r + b7qr) + b8
% NOTE: y = PR or y = WIP, b=[b0,....b8] = x(1),...x(9), b8 = offset

global qry unity
q = qry(:,1);
r = qry(:,2);
y = qry(:,3);
 e = y - (x(1)+x(2)*q+x(3)*r+x(4)*q.*r)./(unity+x(5)*q+x(6)*r+x(7)*q.*r);
```

APPENDIX D

CALCULATIONS FOR MAPE AND MSEP

The Mean Absolute Percent Error (MAPE) is calculated as:

$$\text{MAPE} = \frac{1}{n}\sum_{i=1}^{n}\left|\frac{(\text{True Value})_i - (\text{Forecast Value})_i}{(\text{True Value})_i}\right| \cdot 100\%.$$

The Mean Standard Error Percent (MSEP) is calculated as:

$$\text{MSEP} = \frac{1}{n}\sum_{i=1}^{n}\frac{\texttt{RMS}(\text{True Value} - \text{Forecast Value})}{(\text{True Value})_i} \cdot 100\%.$$

Where the Root-Mean-Square (RMS) of the forecast error is equal to the square root of the variance (or standard deviation) of the error as presented in Table 7.2 and 7.4.

Note that both the MAPE and MSEP are percentage error statistics.

APPENDIX E

MATLAB CODE FOR MEAN FIRST PASSAGE TIMES

```
function M = MFPT(P,phi)
 % The MFPT function is called in the 'core routine' code.
% This function computes the Mean First Passage Time (MFPT) based the
% Transition Probability matrix 'P' and the steady state probability matrix 'phi'
% (with dimension: N x 1). The MFPT is defined as the matrix 'M' which has
% identical dimensions as 'P'. The equations to be solved are:
% M(i,i) = 1/phi(i)
% M(i,j} = 1 + SUM( T(i,k) * M(k,j) )   for 'k' not equal to 'j'
% In the latter equation, M(i,j) is solved in a column-wise manner.

% Example of solving the third column, i.e. for j=3:
% [ M(1,3)]                                                    [M(1,3)]
% [ M(2,3)]                                                    [M(2,3)]
% [ M(4,3)]  = ones(N-1,1) + P([1:2,4:N],[1:2,4:N]) * [M(4,3)]
% [     :     ]                                                [     :    ]
% [M(N,3)]                                                    [M(N,3)]
% In the above the column M(:,3) can easily be solved.
% NOTE: in the example the M(3,3) related parts are discarded.

% ------ Start Function Code ----- %
% ------------------------------------ %
N = length(phi);
Icol = ones(N-1,1);
 % Computing the diagonal entries, i.e. M(i,i) = 1/phi(i):
% (note: the M(i,j) entries are initialized as zero)
M = diag(1./phi);

% Solving M(i,j) in a column-wise approach:
for i=1:N
   % discard the M(i,i) related parts:
   ind = [1:i-1,i+1:N];
   % Solve M(:,i) using linear least squares (pseudo inverse):
   temp = (eye(N-1)-P(ind,ind));
   M(ind,i) = pinv(temp)*Icol;
end
```