MODELLING, OPTIMISATION AND CONTROL OF SERIES SUPPLY CHAINS AND PRODUCTION PROCESSES

By

Christos I. Papanagnou

SUBMITTED IN PARTIAL FULFILLMENT OF THE
REQUIREMENTS FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY
AT
CITY UNIVERSITY OF LONDON
NORTHAMPTON SQUARE, EC1V 0HB
JULY 2007

CITY UNIVERSITY OF LONDON

DEPARTMENT OF

SCHOOL OF ENGINEERING AND MATHEMATICAL SCIENCES

The undersigned hereby certify that they have read and recommend to the School of Engineering and Mathematical Sciences for acceptance a thesis entitled "**Modelling, optimisation and control of series supply chains and production processes**" by **Christos I. Papanagnou** in partial fulfillment of the requirements for the degree of **Doctor of Philosophy**.

Dated: July 2007

External Examiner: _____

Prof. Antonios Kokossis

Research Supervisor: _____

Dr. George Halikias

Examing Committee: _____

Prof. David Stupples

CITY UNIVERSITY OF LONDON

Date: **July 2007**

Author: **Christos I. Papanagnou**

Title: **Modelling, optimisation and control of series supply chains and production processes**

Department: **School of Engineering and Mathematical Sciences**

Degree: **Ph.D.** Convocation: **August** Year: **2007**

Permission is herewith granted to City University of London to circulate and to have copied for non-commercial purposes, at its discretion, the above title upon the request of individuals or institutions.

Signature of Author:

Zwar geht in der Welt

alles mit naturlichen Dingen zu.

Nichtsdestotrotz ist das Ergebnis

wunderbar.

"Das Erbe des Neandertalers"

Hoimar von Ditfurth

Table of Contents

List of Tables

List of Figures

Abstract

In recent years supply chains have gained the attention of both academia and industry. In this thesis, a novel state-space model of a multi-node supply chain is presented, controlled via local proportional inventory-replenishment policies. The model is driven by a stochastic sequence representing customer demand. The model is analysed under stationarity conditions, guaranteed to arise if the control parameters lie in a certain range which is identified and a simple recursive scheme is further developed for updating the covariance matrix of the system in closed form, i.e., as an explicit function of the control parameters. This allows us to analyse the effect of inventory policies on the "bullwhip effect" (demand amplification) for chains with an arbitrary number of nodes.

The three-node model is subsequently analysed in detail under information-sharing and the optimal policy is derived, which minimises inventory fluctuations (and inventory mean) under a probabilistic constraint related to downstream demand. It is shown that this policy can never lead to demand amplification in the chain, as long as the gain parameter of the downstream node lies in the stability region. Finally, issues related to estimation schemes based on local historical data are discussed. The main results and conclusions are illustrated via numerous examples and simulations.

An alternative model of the supply chain is also developed using timed Hierarchical Coloured Petri Nets (HCPN). This approach considers supply chains as event-driven systems and studies decentralised control structures by analysing the impact of various continuous inventory policies and known forecasting methods followed by supply chain participants. CPN-Tools [fCPN] are used for the design of decision-making policies and simulation results are presented to highlight the main issues arising in real systems and to provide insights for future modelling and

simulation work.

Finally, a detailed case study is undertaken, for the production line of the "Bridngorth Aluminium Ltd" company which produces high quality rolled aluminium lithographic strips. An efficient representation for such production processes is provided and subsequently used for an extensive analysis and performance evaluation through appropriate metrics. In particular, the work addresses the implementation of an overall model in a simulation environment, capable of integrating the various aspects of the specific production management processes.

Acknowledgements

I would like to thank Dr. George Halikias, my supervisor, for his many suggestions and constant support during this research. I am also thankful to Professor Nick Karcanias for his guidance through the first year of survey of literature and for his useful ideas and experience in the field of control and supply chain management.

I would also thank Mr. Graham Flukes production manager, Mr. Sneyd Gareth supply chain manager, and Mr. Yannis Angelis purchasing manager in "Bridgnorth Aluminium Ltd", who expressed their interest in my work. I am also grateful to my brother Elias who was the link between me and the "Bridgnorth Aluminium Ltd" company . Without him this collaboration would never have achieved.

I should also mention that my postgraduate studies in City University were supported by the Greek State Scholarship Foundation.

Of course, I am grateful to my father Ioannis who instilled courage into me, my mother Catherine and Evi for their patience and *love*. Without them this work would never have come into existence.

Finally, I wish to thank all my colleagues and academic staff at Control Engineering Research Centre (CERC) in City University of London, especially Dr. Efstathios Milonides and Professor David Stupples for their useful advices.

This thesis has been written using the LaTeX typesetting system and its implementation has been achieved through MiKTeX (Version 2.4) packages.

Nomeclature

$\mu_t = E(x_t)$: Mean of the signal x_t

σ : Standard deviation

$Var(x_t) = \sigma^2$: Variance of signal x_t

$N(\mu, \sigma^2)$: Normal distribution of mean μ and variance σ^2

$\mathbf{G_i}$: Node i in series supply chain

$\mathbf{\Phi}$: Manufacturer node in series supply chain

$I_i(t)$: Inventory of node i at time t

$IP_i(t)$: Inventory position of node i at time t

$O_{i,i+1}(t)$: Amount of orders placed by node i to node $i+1$ at time t

$O_i^\star(t)$: Standing orders of node i at time t

$Y_{i,i-1}(t)$: Amount of products dispatched from node i to node $i-1$ at time t

$SP_i(t)$: Target Set-point of node i at time t

L : Lead time

k_i : Inventory replenishment gain factor of node i

$w_{i,l}$: Left input vector of node i in series supply chain

$w_{i,r}$: Right input vector of node i in series supply chain

$z_{i,l}$: Left output vector of node i in series supply chain

$z_{i,r}$: Right output vector of node i in series supply chain

λ_i : The i-th eigenvalue of a matrix

$trace(\cdot)$: Sum of the diagonal elements of a matrix

$vec(\cdot)$: Vectorisation operator

$\|\cdot\|_F^2$: Frobenious norm

Abbreviations

AO : Aggressive Ordering replenishment policy
APS : Advanced Planning and Scheduling
ARMA : Autoregressive and Moving Average
BWG : Levelling, stretching and degreasing machine centre
CPN : Coloured Petri Nets
CPFR : Collaborative Planning Forecasting and Replenishment
CRM : Customer Relationship Management
DES : Discrete Event Systems
EDI : Electronic Data Interchange
EOQ : Economic Order Quantity
ERP : Enterprise Resource Planning
ES : Exponential Smoothing replenisment policy
FIFO : First-In First-Out
FSM : Finite State Machines
HC : Holding Cost
HCTPN : Hierarchical Coloured Timed Petri Nets
iid : independent and identically distributed
IMC : Internal Model Control
JIT : Just in Time
LIFO : Last In First Out
LTI : Linear Time Invariant
MA : Moving Average replenishment policy
MTO : Make To Order
MRP : Manufacturing Resource Planning
MPC : Model Predictive Control
OC : Ordering Costs
SC : Shortage Costs
SCC : Supply Chain Council
SIRO : Service In Random Order
TCU : Total Cost per Unit time
TSM : Cold rolling machine centre
WMS : Warehouse Management Systems

Chapter 1

Introduction

1.1 Supply Chain and Logistics

In recent years supply chains have gained the attention of both academia and industry. The distribution of products with shortened life cycles, and the continuously increasing customers' expectations has led companies to invest more on supply chain management. Producers have made an effort to reduce product costs significantly while maintaining excellent product quality and customer high level of services. The globalisation of markets together with the elimination of import trade duties and restrictions has also forced manufacturers to look for ways to improve their competitive positions by focusing on supply chain management. Many companies are discovering that significant savings that be achieved by managing their supply chain more effectively. At the same time, information and communication systems have been widely implemented providing access to elaborate data from all the components of the supply chain. Although the main research work is related to manufacturing systems and the transport of finished products, supply chains also arise in service systems.

Numerous definitions of a supply chain have been suggested while what is meant by "supply chain" appears to be different for various companies across industry. It has also been argued that we should not talk about "chains", as what is described by this term is much more complex entity. Real "supply chains" look more like complex "networks" with information and goods flowing across and between firms (echelons)

at all parts of the system. A supply chain is a network of facilities and distribution options that performs the functions of procurement of materials, transformation of these materials into intermediate and finished products, and the distribution of these finished products to customers. Supply chains exist in both service and manufacturing organisations, although the complexity of a chain may vary greatly from industry to industry and company to company. In a typical supply chain, raw materials are procured and items are produced at one or more factories, shipped to warehouses for intermediate storage, and then shipped to retailers and customers whose satisfaction is a measure of a company's success. Consequently, to survive in the present global competitive environment, organisations need to show a heightened awareness to customers needs. A typical supply chain network is illustrated in Figure 1.1

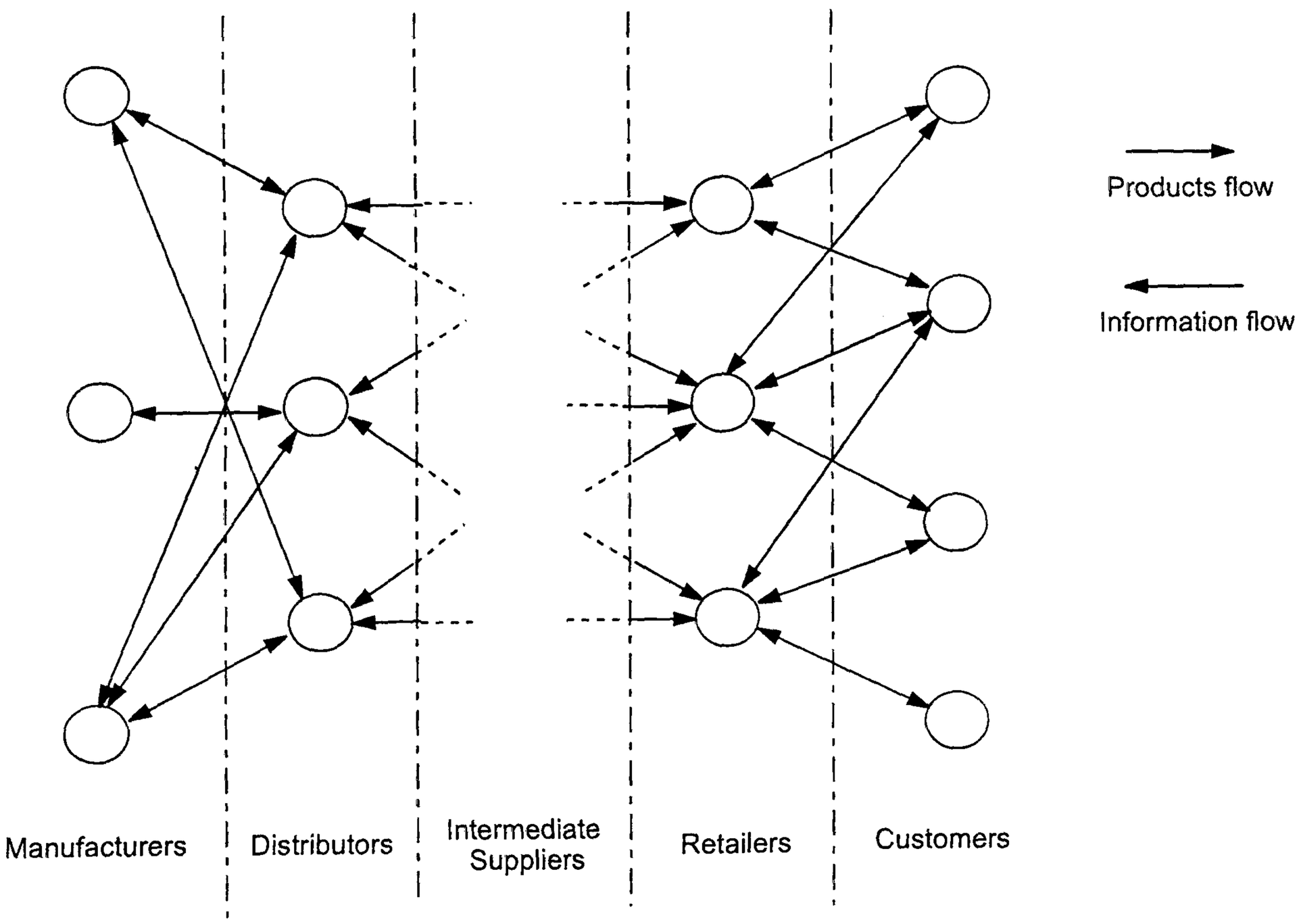

Figure 1.1: A typical supply chain network

Another term that has been widely used in recent years is "logistics". It has been

noticed through the study of supply chains that managing directors and people who deal more with the practical aspect of supply chains within companies, in contrast with academics, use the term logistics to describe the flow of goods and information. It can been argued that logistics and supply chains have both similarities and differences. Logistics summarises all activities connected with the supply, storage, and carriage of goods (transportation) or services. Therefore, logisticians in most companies focus more on how to create intelligent warehouses and flexible facilities in distribution centres (i.e. packaging and loading of finished products) by upgrading equipment such as forklifts and wreckers, and on how to manage better the inventory in a storehouse by using special software tools and computerisation. They also focus on how to improve transportation services and other related issues like customs duties and customs declaration.

Leaving the borders of companies and considering the whole logistics network as a work-flow process, which needs to be modelled and controlled with well-defined inputs-outputs and constraints between individual parts, we encounter the problem of supply chain management, or to use a more precise term, to supply-production-distribution network management. Thereby, supply chains involve a type of integration of all individual parts (manufacturer, distributors, suppliers and customer). Using this holistic and process oriented viewpoint, supply chains appear more like as a system of a continuous flow of materials and information. This macroscopic view can also underlie the appliance of quantitative methods for modelling, planning and control. Therefore, it can be inferred that logistics deal only with a part of the supply chain process, which draws, materialises and checks the effective and efficient flow and storage of materials, services and relative information from the point of creation to the point of consumption, and concentrates on the satisfaction of requirements of customers.

The concept of logistics and supply chains seem to be appeared many years ago. Alexander the Great [Eng78] in 3rd century B.C introduced first the inclusion of Logistics in strategical planning. Supply Chain was the basis of Alexander the

Great's strategy and tactics during his expedition to Asia. He was aware of the significance of military intelligence and securing methods for both provisions and transportation. Alexander's logistic system inspired many other military governors including Julius Caesar and Napoleon.

Although the idea of logistics and Supply Chain was first demonstrated by Alexander the Great, Leo VI the Wise (866-912) the Byzantine Emperor in 900 A.D introduced the notion of logistics in his book "Tactica". Leo VI describes the appropriate arrangement in time of both provision and cannons according to soldiers' needs and the preparation of the campaign in terms of time and space (battlefield and camp) computation. He presents logistical tactics, and also suggests the establishment of a special provision and transportation corps, responsible for the estimation of the opponent's strength, and providing solutions for the next possible movement, future plans and allocation of their own armed forces.

Despite several economic, cultural and technological changes the main goal of logistics and supply chain has remained the same, i.e to transfer finished products from a manufacturer site to a (sometimes global) end customer in the presence of other players in the market. The term "players" used here suggests the existence of a "game" in which participants cooperate and/or compete with the objective of winning end customers from other companies. It is clear that with the passage of time supply chains have become more complex, dynamic and heterogeneous. Increasing customer demand has led those involved in supply chain management to develop more efficient and effective supply chain networks.

Logistics as an area of study first gained attention in the early 1900 with the distribution of farm products, as part of an organisational strategy and as a way of providing time and place utilities to sell these products. Supply chains in the modern sense, i.e. as a flow of products and information first appeared in the 20th century. In 1913 Henry Ford coined mass production of the automobile in the wide market by using assembly lines and material flow hard automation. This artefact has led to the creation of shopping stores (outlets) and the establishment of sale-points

outside the borders of the main production areas. Meanwhile, the construction of roads and the evolution of transport has allowed many alternative ways of products' carriage and ameliorated the conditions of shipping, on which today's transportation system is based. Technological innovation and the development of communication together with the globalisation of the market has changed the business environment which has shown in recent years to be more complex and competitive. The internet created yet another major shift, allowing people to purchase from their homes. This increased the needs for delivery through multiple channels and the coordination of returns. In between, the exponential growth of computer technology and information was a tremendous breakthrough in business computing. Businesses introduced a wide range of application such as manufacturing planning (MRP), computer aided engineering (CAD) and inventory management systems [EP97], [GBS01], [BHT+96]. Information technology (IT) is essentially the platform for businesses to develop advanced software programmes and resource planning applications (ERP) [DDVK00], [Fre01], [Mer01]. Shared information among multiple individual and functional areas of a company is the vehicle for partners to have real time access on information and applications related to supply chain. This "extraction" and sharing of information is called Electronic-Data Interchange (EDI) and nowadays tends to be a competitive advantage for many companies [MHM03], [LSS03], [MB04]. Another useful application of computers and software implementation is the development of Warehouse Management Systems (WMS) which make distribution centres and warehouses run more efficiently and profitably [VRZ00], [KMPS05]. Advanced Planning and Scheduling (APS) technology and Customer Relationship Management (CRM) are decision support systems that help a company to develop advanced decision support systems and to obtain better selection of business partners and customers during the material flow among the several stages of the supply chain, especially the upstream levels [dKG03], [Hei02].

Many companies today make an effort to develop systems in which products

are "pulled" through the manufacturing process from the end rather than "pushed" through from the beginning. Each section of the production process makes the necessary units only when they are required by the next stage of the manufacturing process, or by distributors or customers. These systems are known as Just in Time (JIT) production systems and first appeared in TOYOTA in the early 1950's. Many manufacturing companies have followed this idea by developing JIT systems (some managers also refer to them as "lean production") and managed to increase throughput time, which increases productivity [KK02], [Mer01], [vdVHH04], [KH03]. JIT systems on the other hand require large networks of subcontractors and they are based on long-term relationships and mutual trust, essential factors that are difficult to find in today's supply chains.

The activities executed in the frame of a supply chain today vary from company to company and depend mainly on their organisational structure. The majority of today's supply chain activities concentrate on key issues such as customer service, minimising total cost and delivery time, inventory control, flow of products and pertinent information, and order processing. Methods for improving of all these issues are extremely discussed in supply chain literature, via techniques related to supply chain management, planning, integration, coordination, design and operation. The Supply Chain Council (SCC) developed in 1998 a supply chain operations reference model as the cross-industry standard for supply chain management [Cou03]. This process reference model contains a standard description of management processes, a framework of relationships among the standard processes, and standard metrics to measure process performance. In most cases, however the management and analysis performed by companies is based on experience and intuition.

The area of supply chains is very broad and it would be impossible for a research thesis to cover all relevant topics in depth. Although supply chains are very complex networks there are some key factors that have gained the attention of researchers. Recently, significant attention is given to the control problem of supply chains. The motivation for this stream of research has arisen from problems faced by a diverse

set of companies dealing with the flow of products and information throughout the supply chain. At the heart of the control problem is decentralised decision making, production capacity and inventory levels in all echelons of the supply chain, holding costs of inventory, minimising lead times, demand variability and demand forecasting.

Supply chain management is becoming increasingly aware that the overall efficiency of company's operation is related to inventory level existing within the company. Inventory control is an issue which many researchers and practitioners have found important and a great amount of work on inventory control, (also referred in the literature as inventory management) has been published in the last 40 years.

There are obviously advantages to having a large inventory of raw materials and components parts. It gives manufacturing companies protection against temporary price rises, and delays in the delivery of finished products due to shortages, strikes, orders that get lost, incorrect or defective shipments, and so on. Inventory managers can also take many advantages of quantity discounts in purchasing. Having a large inventory of finished goods allows a company to meet varying product demand profiles and be more flexible in product scheduling, with longer production lead times and reduced costs because of larger production runs with fewer set-ups. If managers have a long delivery lead time there is always a risk that some customers may go to other suppliers or that new competitor will enter the market.

On the other hand, keeping an inventory involves various costs. Storage requires warehousing, more packaging facilities, and administration. Handling goods involves labour costs, and unsold goods have to be protected and insured. All this money could perhaps be more profitably spent in several other ways. Furthermore, there is always a risk of obsolescence, theft or breakage, especially for those firms producing high-tech products with a short life cycle. If an inventory of finished products gets too large, it may be necessary to reduce prices to stimulate demand. Some organisations (notably wholesalers and retailers) have an inventory of finished products only, while many industrial companies or businesses hold different types of inventory in order to achieve better inventory management. A company's stockholding policy is

implemented by explicit rules which determine the manner and time certain decisions concerning the holding of stock should be made. This set of rules is known as an inventory policy.

As stated earlier, in contrast with logistics, a supply chain network within a company can be seen as a delivery system, by which goods and information can be moved from one place to another as well the means by which goods undergo the transitions in manufacturer sites from raw materials to finished goods. Like any system with inputs, outputs and its dynamic variables, a supply chain's input can be considered as the demand customer patterns, output typical metrics that companies use to assess their supply chain's performance, while dynamic variables consist of all those parameters which are (directly or indirectly) affected by the flow of information and goods within the supply chain networks. Hence, any change in these dynamic variables has great sway in the output of the system. Considering the metrics associated with the amount and time the goods being delivered to customers, companies need to decide how to strike the right balance between customer satisfaction (by storing a lot of extra supplies) and the efficiency of having inventory just when it is needed. Customer satisfaction (or resentment) relates to customer service level, mainly determined by the amount of time the goods reach the customers after orders are placed. Thus customer satisfaction (output) depends on the inventory policies of the companies which participate in the supply chain network.

The way the companies deliver goods to downstream participants of their supply chains is also linked with the total-order delivery. Supply chain managers must frequently decide if it could be better to hold onto an order until all the parts of the order are ready. In this case, when the order is complete, it is shipped to the downstream participant. Alternatively, the company could immediately ship the products on hand, and follow up with a second shipment. This decision is taken principally by manufacturers by dint of product postponements, when the last few steps of production process for a product can be postponed until demand for it arises.

There are also several other metrics within a supply chain system. A common metric [Sch03] is that of *logistic costs* which can be divided to Holding Costs (how much a company pays to store its goods in warehouse each day and the costs for the number of warehouses), Ordering Costs (the costs to process an order), Shortage Costs (the penalty costs arising when a company is running out of stock) and Transportation Costs (the total cost of moving goods to and from a company's facilities). Logistic costs are also associated with customer satisfaction since it is often possible to measure the direct costs of providing a certain level of customer service.

Another issue pertaining to dynamic variables is the amount of orders that a company must place to the upstream parts of its supply chain. Managers must often show anticipation by taking decisions about order quantities with a perspective of future inventory changes and demand patterns. These logistical postures have led managers to adopt forecasting techniques in order to predict future customers' demand. However, due to rapid market changes and supply chain complexity, there is no currently proposed demand forecasting technique that is universally valid. Managers who are considered competent to perform this task, often have the ability to visualise the demand forecasting problem holistically by understanding and implementing the concepts of effective forward-looking supply chain management and do not necessarily rely on demand forecasting trends [Poi99].

An important factor that affected the supply chain management was the growth of third-party logistics or 3PL [Men99]. 3PL began to proliferate in the 90's and involved the voluntary outsourcing of a company's transportation function to an outside firm catering specifically to the logistics market. 3PL has wrested distribution and logistics from in-house activities and has helped managers to deal with the lack of skills, capabilities or infrastructure required to manage the complexities of the global environment. Outsourcing has also helped also managers to understand the globalisation objectives of the company and to ensure in many cases that requirements for achieving these objectives are a part of the outsourcing decision.

Schechter and Sander [SS01] allege that during the coming century smart logistics will increasingly become the strategic differentiator between companies that succeeded and those that drop behind. Colossal companies have created so far tremendous value for their shareholders by seeing the logistical light and considering supply chain management as a competitive weapon.

Companies that will be deemed to comprise advanced technology as an asset will be always on the front and those that will keep up with logistics evolution. Burt et al. [BDS03] argue that industry, companies and government will continue to have a supply management function - one which grows in importance. Many of the manual activities previously performed by purchasing personnel are being automated or reassigned so that supply professionals focus on producing high value. Buyers of the future will understand the entire supply chain, all innovation trends and global capacity. They will develop suppliers worldwide, who will meet their needs and they will segment and articulate where their suppliers fall within their portfolios. E-procurement will be one of the most exciting developments in supply management in recent years. Buyers will be no longer responsible for non-value-adding activities and paperwork processing. Meanwhile, they will be increasingly empowered to place orders through the Internet directly to supplier. Web-based tools will facilitate the flexibility to allow companies both to view deep into the supply chain to see how their suppliers are performing and to reconfigure their supply chain as circumstances demand. This new tendency will bring over present supply chains to the new era of value chains where companies will recognise the importance of demand in addition to supply.

The opportunity to improve supply chain performance by sharing information has long been acknowledged. Much of the focus thus far has been on exchanging inventory and product movement data throughout the supply chain. More recently, companies have found that sharing information relating to market intelligence and promotional plans can dramatically improve forecasting, thus smoothing the replenishment process [Bal04]. Increasingly, the goal is to replace physical assets

with information in such a way that every member of this extended supply chain benefits.

One of the first signs of this has been the development of Business-to-Business (B2B) exchanges in which groups of manufacturers create an electronic hub linking suppliers and buyers (e-business and e-commerce) [Sch03]. These exchanges will lead to virtual supply chains able to support both relationships and collaborative partnerships. The choice of which to adopt is not a function of technology, but rather good supply chain management. A virtual supply chain will exploit technology to allow a company or an organisation to connect, align ways of working and transact for an optimum period. Relationships will not be exclusively transactional, nor are virtual supply chains at odds with long term strategic relationships. On the contrary, the technology which underpins a virtual supply chain also underpins integrated product development, collaborative forecasting and synchronised flow.

1.2 Survey of Literature

Supply chain management literature is vast and rapidly expanding. In this research work we restrict our attention in issues which are related to inventory control, decentralised supply chains, modelling and simulation methods and the use of control theory in supply chains. The inventory management problem was first studied in 1960 by Clark and Scarf [CS60] who developed a periodic review inventory control model for a serial multi echelon inventory system without setup costs and assuming a finite planning horizon. By using a base-stock control framework they established that while inventories are managed locally, ordering policy at each node is optimal. Federgruen and Zipkin [FZ84] extended these results further by studying the optimal policies in the infinite-horizon case. Muckstadt and Thomas [MT80] adapted the Clark and Scarf model ([CS60]) to a specific situation and conducted a computational study. A direct generalisation of [CS60] is presented in [Ros89], where an inventory model of an assembly system with random demands and proportional costs of

production and stock holding activities is considered. Axsäter [Axs93], [Axs98] evaluates different inventory policies where all the stages in supply chain place orders in batches.

Several inventory policies can be found in literature [Zip00], [vHSWY03], [ML88]. Those policies where decisions concerning replenishment are based on the current inventory level are known as "Re-order level policies". In this type of policy an order for replenishment is placed when the inventory level (stock on-hand) drops to or below a fixed value ξ known as the reorder level. The amount of inventory held can be reviewed continuously or periodically. When decisions are made on a time basis then the inventory policy is known as "Re-order cycle policy". In (s, S) policies [FZ84], [Cap85] the stock on-hand is reviewed periodically, where S represents the fixed inventory level and s the level to which the stock on-hand drops at review for a further replenishment order $(S - s)$ to be placed. Muckstadt and Thomas [MT80] present an $(s-1, s)$ ordering policy where after each demand taking place at a stage, an order is placed for one unit, bringing on-hand plus on-order inventory at the stage up to s units. Axsäter [Axs98] considers an (R, Q) policy, where while the installation stock level declines to or below R a batch of size Q is ordered. (R, S)-policy discussed in [vHSWY03] follows the "Re-order cycle policy" where every R time units (Re-order interval) an order is placed to return the inventory position to S.

Decision making at each stage in supply chains has also gained a lot of attention recently. Many important problems arise when decisions are made locally and therefore the supply chain can not be controlled by a central supervisor, i.e., without full information or with distorted information. Distorted information within a supply chain can lead to very high inefficiencies, such as excessive inventory investment, ineffective transportation, poor customer service, lost revenues/profits, and misguided capacity plans or missed production schedules [LPW97a]. Chen [Che98] assesses the value of centralised demand information and how this value depends on several key system parameters, i.e., lead-times (the time needed for the goods

to be delivered to the downstream level after they have been dispatched by the upstream level of supply chain), batch sizes, number of stages in supply chain, and demand variability. An important issue of more recent research work is decentralised control design and decentralised supply chain formation when the model itself is a framework that combines different decision policies at separate supply chain stages [CPA99], [LLBC00], [AM00], [JK04], [PLGYT01], [GP04], [CA98]. Welsh and Wellman [WW03] present a simple model of supply chains with hierarchical subtask decomposition, and resource contention. They use agents to choose locally optimal allocations with respect to prices. Agents are able to communicate with each other in order to solve challenging competition problems, while multi-agents systems can provide supply chain integration. Watson and Zheng [WZ05] present decentralised serial supply chains subject to order delays and information distortion by sharing real-time sales data across all stages of supply chains whereas Caramanis and Anli [CA98] describe a hierarchical decomposition framework that facilitates near-optimal dynamic production control through coordinated decentralised decision making. Lee and Billington [LB93] develop a model framework from which general supply chain inventory problems can be tackled. This model has been applied to the Deskjet printer decentralised supply chain structure at Hewlett-Packard company.

In order to estimate total market potential, companies need to forecast the number of buyers and the average quantity that they intend to purchase. Wrong estimates can result in excessive inventories and increase of costs, or on the contrary, to lost sales due to insufficient production and shortages. This provident task is called demand forecast or market forecasting. There are various methods of forecasting although they all depend on past information, surveys and interviews of a statistically selected sample of customers [Jar91], [MWH98]. The selection of a forecasting model influences the performance of the supply chain and the values of information sharing [ZXL02]. Forecasting can be very difficult when there is uncertainty on demand, in periods when demand is amplified or fluctuates sharply between low and high values [MP95].

Kahn [Kah87] presents a model for production decisions and demonstrates how demand uncertainty has effect on inventories. A framework comparing the variance of demand to the variance of replenishment orders at different stages of a supply chain is presented in [BC98]. Albertson and Aylen [AA03] report a successful approach to forecasting UK manufacturing stock behaviour sponsored by a leading European metals manufacturer. Another method that has attracted the attention of a considerable number of companies recently is the Collaborative Planning, Forecasting, and Replenishment (CPFR) process [Sei03]. CPFR is the sharing of forecasts and related both long and short term business information among participants in supply chains to improve the flow of goods to the downstream supply chains levels.

Lee et al. [LPW97a], [LPW97b] made an important observation in supply chains. They discover that demand variability increases as one moves to the upstream parts of supply chains. They also found that there is empirical evidence that orders placed by a retailer are more variable than the actual customer demand (orders) received by that retailer. This phenomenon was coined first by managing directors in Procter and Gamble (P&G) who named it the "bullwhip effect[1]" (derived from the observation that even small variations in actual customer demand can "bash the whip" for upstream parts of supply chain, causing them to alternately order more than the actual demand). (P&G) directors observed that even though customer demand for Pamper's diapers was constant for a certain period, the orders placed by retailers to their wholesalers appeared with significant fluctuations over time. Sterman [Ste89] discusses the bullwhip effect in the context of a simulated industrial production and distribution system developed at MIT; the "Beer Distribution Game". Although the "bullwhip effect" is a new term, the analysis and impact of demand amplification was first studied by Forrester [For61].

In recent years there has been a potential interest on the bullwhip effect by many researchers and practitioners. Miragliotta [Mir06] presents an interesting

[1]In literature can be found as "Forrester effect" or "Whip-lash"

extensive literature review on the subject of the bullwhip effect. He classifies the causes of this phenomenon by introducing a twofold distinction between layers and mechanisms, whose interaction may lead to the bullwhip effect. The majority of the research work revolves around the quantification and reduction of this phenomenon [Met97], [CDRSL00], [DDLT02], [Gil05]. A motivation which is demonstrated by many researchers is the implementation of forecasting methods, where participants of supply chain can build their own forecasts based on the historical demand patterns of the downstream stages [HEC00]. Sun and Ren [SR05], study the impact of three known forecasting methods (moving average, exponential smoothing, and minimum mean square error) on the bullwhip effect in a two-stage supply chain consisting of a single retailer and a single manufacturer. Chen et al. [CRSL00] use the same supply chain structure to measure the bullwhip effect and they demonstrate initially that the use of an exponential smoothing forecast by the retailer can cause the bullwhip effect. Then they contrast these results with the increase in variability due to the use of a moving average forecast. Zhang [Zha04b], Hosoda and Disney [HD04], and Chandra and Grabis [CG04] continue their study by using optimal forecasting procedure that minimises the mean-squared forecasting error for a specified demand process. Customer demand is described by a first-order autoregressive process while the replenishment method is based on an order-up-to inventory policy. Xu et al. [XDE01] complement the above work by incorporating forecast uncertainty and alternative demand scenarios. This type of forecasting reckons the effectiveness of supply chain coordination programs in terms of linking information flows, reduces both the bullwhip effect and safety stocks, and investigates how these programs can be applied with stationary and non-stationary demand patterns. The same supply chain model with similar replenishment policy but with autoregressive and moving average (ARMA) demand process is presented in [Zha04a].

Collaboration and information sharing between supply chain participants has become in recent years one of the main issues to alleviate efficiently and effectively the bullwhip effect. The value of information sharing and its advantages is presented

in [LST00]. [Fia05], [MPvLV02] and [ZT04]. Kim [Kim00] shows by developing a mathematical model that information sharing between two collaborator participants in supply chains can be sustainable only in cases where their relationship result in enhancing the profitability of both participants. Kok et al. [dKJvD+05] by applying stochastic multi-echelon inventory theory, they developed an advanced planning and scheduling system that supports weekly collaborative planning of operations by Philips Semiconductors and one of its customers. They discovered that their project has brought substantial savings to the company by eliminating simultaneously the bullwhip effect. A novel co-ordinated supply chain modelling approach is proposed in [LKL02], in order to capture the complexity of supply chains and provide the basis for supply chain integration.

Bullwhip effect research is also interested in the control of inventories of all parts of supply chain. Moreover, the underlying structure of supply chain can be considered as complex system with dynamic behaviour, inputs, outputs, disturbances and a well-defined mathematical description. This has led many researchers to apply control theory and several control methods and techniques in order first to describe and then eliminate the bullwhip effect in supply chains. Dejonckheere et al. [DDLT03] measure the variance amplification of orders within order-up-to policies from a control engineering perspective and prove that classical order-up-to policies always generate a bullwhip effect. They consider demand patterns as inputs and the corresponding replenishment of production orders as outputs while the interactions between different parts of the supply chain are modelled by transfer functions. Hoberg et al. apply linear control theory to study the effect of three different inventory policies on order and inventory variability in a two-echelon supply chain..

A discrete control theory model of a generic model for a replenishment rule is presented in [DT03]. From this model, an analytical expression for bullwhip is derived that is directly equivalent to the common statistical measure often used in simulation, statistical and empirical studies to quantify the bullwhip effect. Extended

results obtained through statistical analysis and important insights in the dynamic behaviour of the replenishment rules are reported in [DDLT04]. Kim et al. [KCHH06] extend Dejonckheere et al. [DDLT03] and Chen et al. [CDRSL00] works by including stochastic lead time and by providing expressions for quantifying the bullwhip effect, both with information sharing and without information sharing.

Sheu [She04] presents a multi-layer demand-responsive stochastic optimal control strategy for alleviating, effectively and efficiently, the bullwhip effect. This control strategy estimates the time-varying demand-oriented logistics system states, which originate directly and indirectly downstream to the targeted member of a supply chain, and associates this estimated demand with different time varying weights under the goal of systematically optimising supply chain performance. Gaalman [Gaa06] uses stochastic optimal control theory to compare a proportional order-up-to policy to full-state-feedback order-up-to policy in supply chains with ARMA demand patterns. Riddalls and Bennett [RB01] apply a novel optimal control algorithm to a differential equation model of a production-inventory system in order to find optimal responses to bullwhip effect. Robust control strategies are used in [BBP06], to meet each time all possible current uncertain demands bounded in an assigned compact set, in multi-inventory systems. Rodrigues and Boukas [RB06] use H_∞ control theory to design a state feedback controller to force the stock level to be kept close to zero even in cases of demand fluctuations.

Braun et al. [BRF+03] present a Model Predictive Control (MPC) methodology as a robust, flexible decision framework for dynamically managing inventories and meeting customer requirements in supply chains. The advantages of the MPC framework as a tuned-scheme to provide acceptable performance in the presence of significant uncertainty, forecast error, and constraints on inventory levels, production and shipping capacity are also discussed. Perea-López et al. [PLYG03] also describe an MPC strategy to find the optimal decision variables and to develop a responsive analysis tool to quickly update the decision making process. They show how an MPC implementation can maximise profit in supply chains with multi-

product and multi-echelon distribution networks with multi-product batch plants. A simulation-based optimisation framework involving simultaneous perturbation stochastic approximation is presented in [SWR06] as a means for optimally specifying parameters of Internal Model Control (IMC) and MPC based decision policies for inventory management and demand uncertainty. Hennet [Hen06] uses linear programming and MPC techniques to construct a stationary production and supply policy in order to react to random variations of deterministic demand profiles.

A framework which captures the dynamic behaviour of supply chains by modelling the flow of materials and information within the supply chain is presented in [PLGYT01] and [PGYT00]. Both works also considers supply chains as decentralised systems and use concepts from dynamics and control, which allow the design of systematic decision-making processes for the supply chain. Using this approach, decisions are seen as the control or manipulated variables of a dynamic system, and an analysis of the impact of different heuristic control laws on the performance of multi-stage supply chains is achieved. Lin et al. [LWJ+04] present also a discrete time series supply chain model based on material and information flow balances. Mathematical expressions are derived to capture the quantity of products and information (orders) while transfer functions for each stage are obtained via z-transforms. The supply chain is then modelled as an overall closed-loop transfer function. Stability of the supply chain system is analysed by using the characteristic equation while control design rules are proposed to alleviate the bullwhip effect phenomenon. Daganzo [Dag03], [Dag04] examines the stability of supply chains in small and large demand perturbations and shows that all decentralised policies that reduce inventories on extended periods of reduced demand under certain conditions (i.e., reliable future demand information) are unstable and lead to bullwhip effect.

Various computer simulation tools have been also proposed recently for the analysis of supply chain performance [TC04]. Most of these tools model supply chains as discrete event systems and examine their behaviour following alternative methodologies. Typically, simulation tools can be used for quantitative analysis

(measurement/prediction of variables) or even quantitative analysis (evaluation of reciprocal effects between individual processes). Chapter 6 describes techniques for modelling and simulating supply chains systems via Hierarchical Coloured Timed Petri Nets (HCTPN) [Jen97].

The use of Petri nets has been recently proposed in supply chain literature. Mevius and Pibernik [vMP04] present a special type of high-level Petri nets (XML-nets) to introduce an integral approach to supply chain process management. Elmahi et al. [EGE02] propose a Petri net model based on max plus algebra to control supply chains. Landeghem and Bobeanu [vLB02] present a method for modelling supply chains via Petri nets by using the well known example of the Beer Game, while Liu et al. [LKvdA04] develop a similar approach for modelling event relationships in supply chains. Makajić-Nikolić et al. [MNPV04] use Coloured Petri Nets to study the performance of a series supply chain by means of CPN-Tools [fCPN]. A HCTPN model has been constructed to study the bullwhip effect in decentralised supply chains where individual nodes use aggressive ordering (AO) based on deterministic customer demand patterns. A more generic approach is also presented in [Bö02].

Production management problems in industry play an essential role in the supply chain management area, by which managers can determine the production loading plan in order to satisfy the end customer demand [Bli86], [Lee96], [GKZ]. Moreover, the bullwhip effect leads to demand amplification in upstream nodes of supply chains (e.g., manufacturers) and may have a tremendous effect in production management of manufacturers. Thus, production planning in manufacturing involves in most cases the synchronisation with the downstream demand and thereby has a strong impact in warehouses of both manufacturers and other participants of supply chains [SRP+04]. A more detailed task in manufacturing is production scheduling where managers in the context of the optimal production planning must couple individual products with individual productive resources in the shortest times [MVJE05]. Scheduling can be a cumbersome task especially in cases when last minute changes are imposed by machine breakdowns, new high-priority orders arrival, and the occurrence of other

disruptions. Chapter 6 presents the modelling and simulation of an aluminium coils production plant, by providing an efficient representation for such production processes [FM84].

1.3 Main objectives of the research work

- *Previous discussion shows that many researchers described the dynamics of supply chains using control theory in order to analyse known empirical phenomena such as the bullwhip effect. The majority of those works consider a simple supply chain consisting of a single retailer and a single manufacturer. The work presented in this thesis aims to analyse more complex models consisting of arbitrary number of nodes.*

- *The analysis of the effects of certain aspects of proportional (continuous) inventory policies on the stability and performance of serial supply chains. Traditional inventory replenishment policies commonly used for supply chain control (e.g., (S, s) policies) have been extensively analysed in the literature. In contrast, continuous policies (e.g., P or PI policies) have only recently been proposed, apparently inspired from the area of classical process control engineering. Their main characteristic is that orders take place continuously, rather than being triggered by specific events (e.g., when inventory falls below a certain target level). Despite possible practical limitations and other issues related to the merits of continuous versus batch ordering, continuous policies can in principle offer additional flexibility (e.g., by smoothing out flows) which can be beneficial for the stability and performance properties of the supply chain. An important objective of the work is to investigate the potential benefits of continuous policies on the stability and performance properties of the chain (e.g., customer satisfaction levels) and to develop a general methodology for modelling and analysing their effect in series supply chains. The analysis should include the case when nodes hold insufficient inventory to meet downstream*

demand.

- *The development of a simple stochastic series supply chain model and the analysis of its properties in the steady-state, under white noise end-customer demand-profiles. Although a white-noise demand profile is clearly unrealistic for real supply chains (as it ignores, for example, trends, seasonal variations or more complex patterns) this assumption offers the advantage of simplicity and can be easily extended to more complex cases, e.g., ARMA demand-profile models. The model should be tuned to the analysis of the dynamic properties of the chain, especially the effects of inventory policies on the bullwhip effect.*

- *The undertaking of a thorough statistical (covariance) analysis of the model. For this purpose a state space modelling approach is more suitable (rather than more traditional transfer function based techniques). In a certain sense, state-space and transfer function approaches are equivalent for discrete LTI systems. For example, if a transfer-function technique is followed, the covariance functions of the output variables of the system can be obtained by taking the inverse (two-sided) Z-transform of the spectral density $\Phi(z) = \sigma^2 G(z)G(z^{-1})$, where σ^2 is the variance of the white-noise input and $G(z)$ is the system's transfer function. However, the state-space approach is more direct and offers the following advantages: (a) State-space methods can be extended to time-varying and non-linear systems and (b) State-space techniques are more suitable for covariance analysis.*

- *The examination of the potential advantages of information-sharing between supply chain participants and the analysis of optimisation techniques for each node under full or partial information. Further, it is aimed to study the applicability of local estimation schemes based on historical data in the absence of information sharing.*

- *The illustration of the main conclusions arising from the model via a detailed industrial case study. The main issues that need to be investigated include*

the modelling and estimation of the cost and inefficiencies arising from highly fluctuating demand patterns.

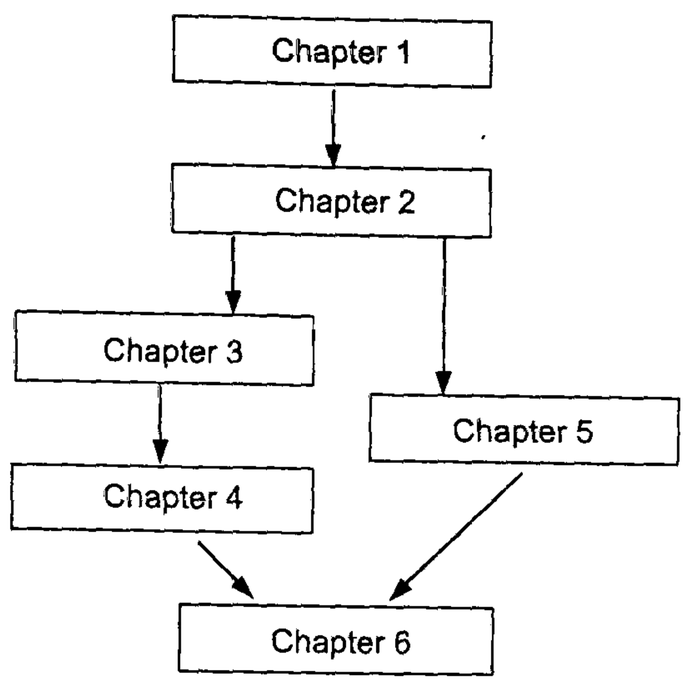

Figure 1.2: Structure of Thesis

As a result of this research work the following papers have been published:

- C. I. Papanagnou and G. D. Halikias (2005). *A state-space approach for analysing the bullwhip effect in supply chains.* In: Proceedings of the 5th International Conference on Technology and Automation. Thessaloniki, Greece. pp. 79-85.

- C. I. Papanagnou and G. D. Halikias (2006). *Supply Chain Modelling and Control under proportional inventory-replenishment policies.* In: Proceedings of the 12th IFAC Symposium on Information Control Problems in Manufacturing. Saint-Etienne, France. pp. 277-282

- C. I. Papanagnou and G. D. Halikias (2006). *Analysing different ordering policies in a series supply chain by using Coloured Petri Nets.* In: Proceedings of the 20th European Conference on Modelling and Simulation. Bonn, Germany. pp. 399-404

The following work is under review in *International Journal of Systems Science*:

- C. I. Papanagnou and G. D. Halikias. *A state-space approach for analysing the bullwhip effect in supply chains under proportional inventory-replenishment policies.*

The following title has been selected after an additional peer review process by Guest Editors to be published in one of the associated International Federation of Automatic Control (IFAC) journals as an extended version of the paper presented in the 12th IFAC Symposium on Information Control Problems in Manufacturing, held in Saint Etienne on May 2006.

- C. I. Papanagnou and G. D. Halikias. *Supply Chain Modelling and Control under proportional inventory-replenishment policies: Covariance analysis, Information sharing, Optimisation and Local Estimation schemes.*

The remaining parts of this thesis are organised as follows. Chapter 2 introduces the main control and modelling methods used in this research work. Chapter 3 discusses the bullwhip effect in supply chains and develops a stochastic state space model for its analysis. Chapter 4 discusses issues related to the selection of optimal policies, information-sharing and estimation schemes in series supply chains. Chapter 5 shows how Hierarchical Coloured Petri Nets can be used for supply chains modelling. Chapter 6 presents a case-study involving modelling methods for "Bridngorth Aluminium Ltd". Conclusions and further work issues are presented in chapter 7. Technical developments related to state space computations and the proofs of various technical results can be found in Appendix A and Appendix B. Simulation results and the software programme used in chapter 6 are included in Appendix C and Appendix D, respectively. The logical connections between the chapters of the thesis are illustrated in Figure 1.2.

Chapter 2

Modelling methods and control problems in supply chain networks

2.1 Modelling methods in supply chains

In many fields of study, a system is studied indirectly through modelling methods, which describe it. Modelling methods are concerned primarily with the quantitative analysis of systems, and the development of techniques for design, control, and the explicit measurement of system performance based on well-defined criteria. Modelling methods must also cope with all dynamic characteristics of the system and should duplicate its observed behaviour. Modelling of a system requires the development of mathematical methods for describing its behaviour, by defining a set of measurable variables. A modelling process also presumes well-defined inputs and outputs as can been seen in Figure 2.1. In complex systems, like supply chains, a model can only approximate the behaviour of the real system.

In continuous-time systems the state generally changes continuously with time, while in discrete time systems the state variables changes at discrete instances. The transition of the system from one discrete state to another is characterised by the events occurring between these instances. Such discrete-state systems are called Discrete Events Systems (DES) and their state transitions can be synchronised either by a global clock generator distributed to all its components or at various specified time instants.

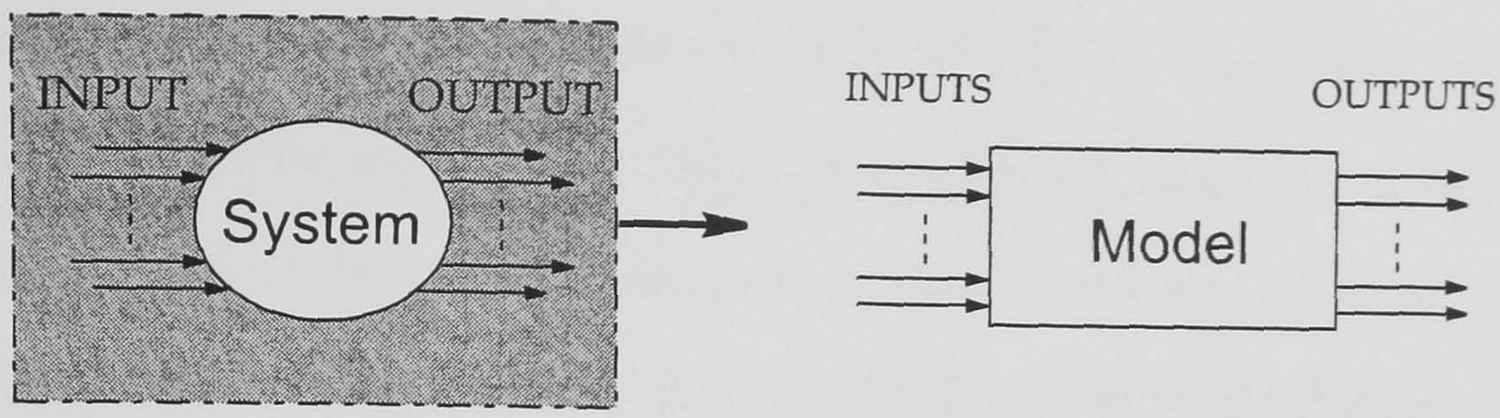

Figure 2.1: Simple modeling process

Definition 2.1.1. A discrete event system is an event-driven system, if its state transition depends entirely on the occurrence of discrete events over time. If the admissible time instances are taken from a continuous or discrete set as defined above, then a discrete event system in continuous time or discrete time, respectively.

We can define a supply chain system as a DES if we think for instance that the state, described by equations associated with the flow of information and products between its components (e.g., suppliers), changes every time an event take place (e.g., receipt of goods from an upstream level). Developers and practitioners are faced with a number of problems when it comes to specifying, simulating, designing, and optimising such complex systems. Due to numerous constraints, implementations typically comprise different models of computation and different types of optimisation. Examples of available tools include:

1. Queueing systems

2. Theory of Finite State Machines (FSM)

3. Data flow descriptions such as marked graphs, synchronous data flow graphs or boolean data flow graphs

4. Languages and automata

5. (Coloured) Petri nets

6. General discrete event models.

Queueing systems and *Petri nets* are used widely for modelling supply chains. More specifically, queueing systems are used to capture more the system dynamics in multi-stage supply chains where each stage (node) consists of many intermediate participants which are acting competitively [RV99b], [RV99a], [BK04] and [AM02]. Similarly, system dynamical models have been considered by Sterman [Ste89] mainly to analyse competitive behaviour in multi-agent distribution systems. Problems of this type are not considered in this work which focuses instead in analysing the dynamics of series supply chains models for which purpose difference equation models and Coloured Petri Nets are more appropriate.

2.1.1 Theory of Finite State Machines

Types of variables

Since complex systems have a lot of variables, it is very convenient to separate the variables which characterise the system into:

1. Input variables, which represent the stimuli generated by systems other than the one under investigation, and which influence the system behaviour. These variables can also be denoted as excitation variables.

2. Output variables, representing those aspects of system behaviour which are of interest to the investigator. These variables can also be denoted as response variables.

3. State variables, which are neither input nor output variables. While the input and output variables are usually quantities which can be observed and measured, state variables are often unmeasurable. These variables can also be denoted as intermediate variables.

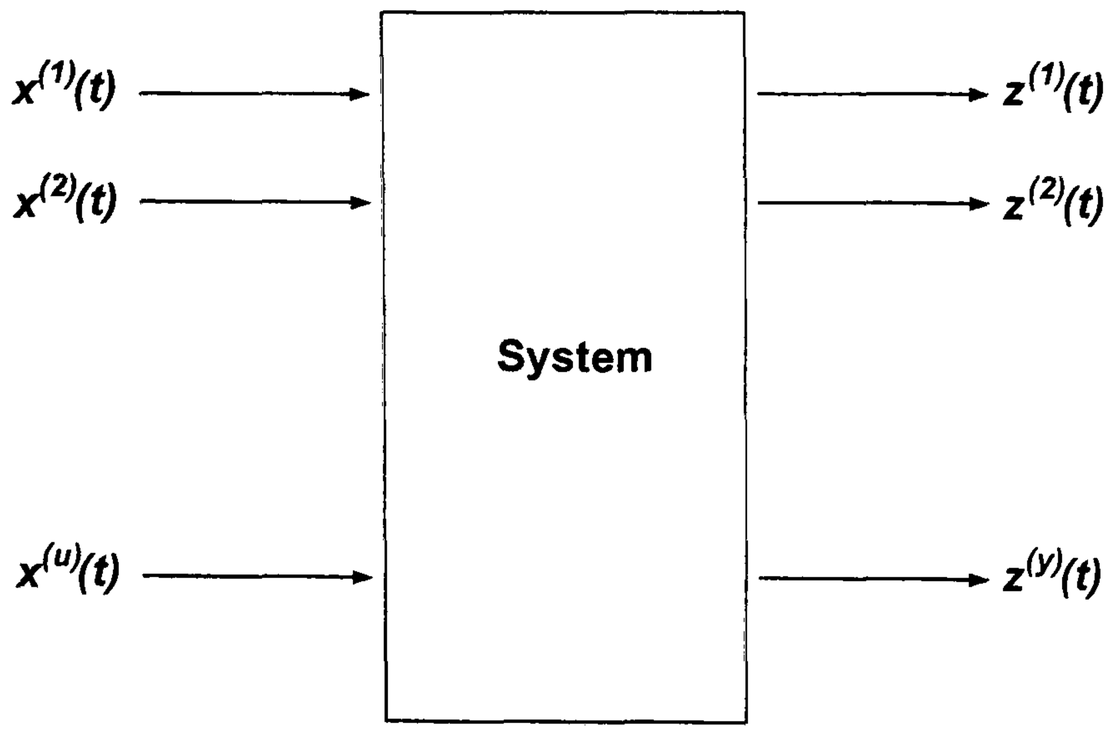

Figure 2.2: Representation of a System with u inputs and y outputs

Schematically, we can use a black box to depict a system, with a finite number of accessible "terminals". The input terminals represent the excitation variables and are identified by arrows pointing toward the box. The output terminals represent the response variables and are identified by arrows pointing away from the box. The intermediate variables, which are of no direct interest, are assumed to be embedded inside the box. The input and output terminals, as well as the box itself, need not have any physical significance; they merely serve to place in evidence those system variables which are pertinent to the problem at hand.

Figure 2.2 shows the black-box representation of a system having u input variables and y output variables, all assumed to be time-dependent. $x^{(i)}(t), i = 1, 2, ..., u$, denote the input variables, and $z^{(j)}(t), j = 1, 2, ..., y$, denote the output variables.

Assumption of Time Discreteness

The theory of Finite state machine is used in discrete time series. There are two important assumptions when we make a research on the time discreteness:

- Each finite-state model is controlled by an independent synchronising source, in the following order: The system variables are not measured continuously, but only at the discrete instants of time at which a certain specified event, called a *synchronising signal*, is exhibited by the source. These instants of time are

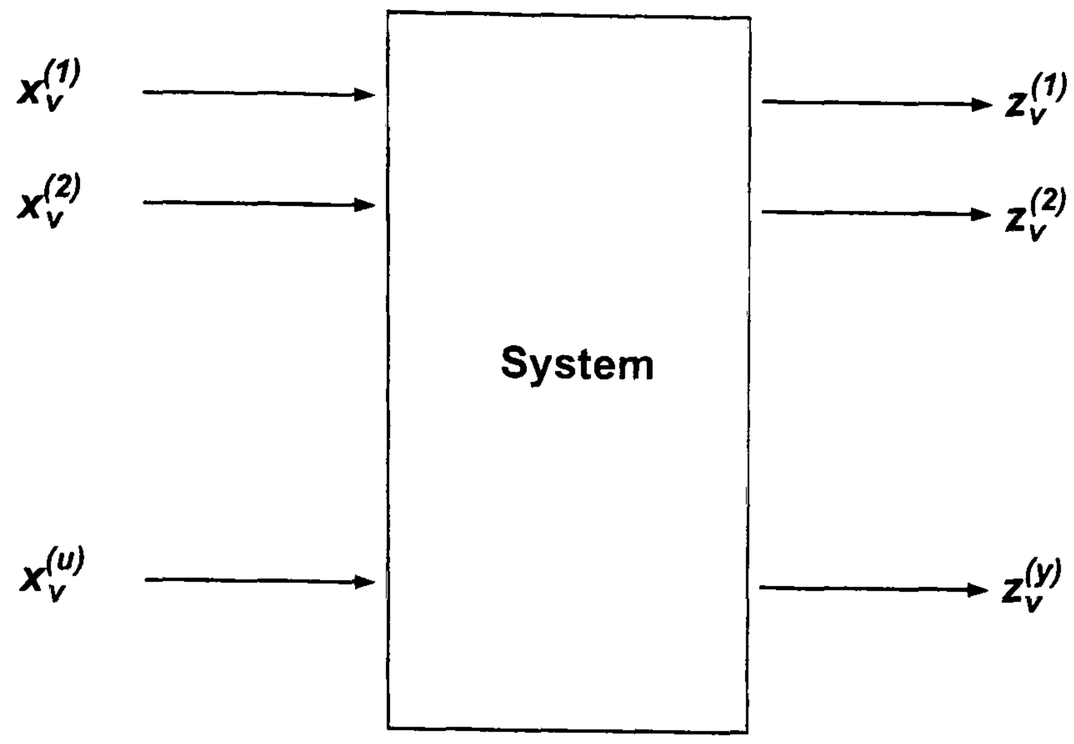

Figure 2.3: Represantation of a System under the time discreteness assumption

called sampling times, the v-th sampling time being denoted by t_v, $(v = 1, 2, ...)$.

- The behaviour of the system at any sampling time t_v is independent of the interval between t_v and the previous sampling time t_{v-1}. Thus, the true independent quantity is not time, but the ordinal number associated with the sampling times. Therefore, a system variable $v(t)$ can be written as v_v , which designates the value of variable v at the vth sampling time.

Systems which conform with the time-discreteness assumption made above are said to be synchronous, while systems in which this assumption is not valid are called asynchronous systems. In this thesis for the case study presented in Chapter 6, our system is asynchronous system but the modelling approach considers the system as a synchronous system.

Based on above notations, we can modify the Figure 2.2 so that it is in accordance with the time discreteness assumption. In Figure 2.3, $x_v^{(i)}, i = 1, 2, ..., u$, denote the input variables at time t_v , and $z_v^{(j)}, j = 1, 2, ..., y$, denote the output variables at time t_v.

Alphabet Finitude Assumption

Besides the time discreteness assumptions mentioned above, another assumption to be made for the theory of finite state is that a variable v can assume only a finite number of distinct values. The set of values which the variable v can assume is called the v *alphabet* and denoted by V and each element in V is called a v *symbol.*

Let's assume a given system has a finite number of variables $e_v^{(1)}, e_v^{(2)}, \ldots, e_v^{(m)}$, at time t_v. Based on the definition mentioned above about alphabet, we will have:

$$E = E^{(1)} \otimes E^{(2)} \otimes \cdots \otimes E^{(m)} \tag{2.1.1}$$

where E is denoted as the alphabet of the system and $E^{(i)}, i = 1, 2, \ldots, m$, is the $e^{(i)}$ alphabet. We can also get:

$$p = p_1 \, p_2 \, \cdots \, p_m \tag{2.1.2}$$

where p is the size of E and p_i is the size of E_i. If each variable $e^{(i)}, i = 1, 2, \ldots, m$, has a finite size alphabet, we can conclude that the system has a finite size alphabet.

Based on proof above, we can say, for a given system, if any input variable $x_v^{(i)}, i = 1, 2, \ldots, u$, and any output variable $z_v^{(j)}, j = 1, 2, \ldots, y$, has a finite alphabet, then the system has a finite input alphabet and a finite output alphabet. Furthermore, it is seen that a single input symbol is enough to describe all u input variables and a single output symbol is enough to describe all y output variables at a given time t_v. Therefore, we can replace all the input variables $x^{(1)}, x^{(2)}, \ldots, x^{(u)}$ by a single input variable x , whose alphabet is defined as:

$$X = X^{(1)} \otimes X^{(2)} \otimes \cdots \otimes X^{(u)} \tag{2.1.3}$$

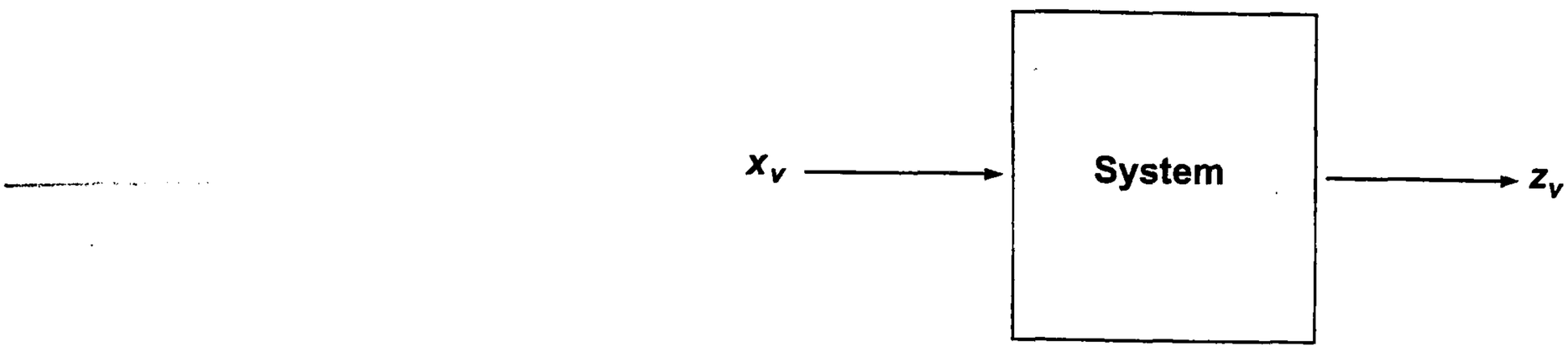

Figure 2.4: Represantation of a Finite State Machine

where each input variable $x^{(i)}, i = 1, 2, \ldots, u$, has an alphabet $X^{(i)}$.

Similarly, we can replace all the output variables $z^{(1)}, z^{(2)}, \ldots, z^{(y)}$ by a single input variable z , whose alphabet is defined as:

$$Z = Z^{(1)} \otimes Z^{(2)} \otimes \cdots \otimes Z^{(y)} \tag{2.1.4}$$

,where each output variable $z^{(j)}, j = 1, 2, \ldots, y$, has an alphabet $Z^{(j)}$.

Also, we can get a new depiction as Figure 2.4.

Definition of State

Though we have mentioned that a state variable is neither input nor output variables, the concept of a state can be accurately defined only through the exact modelled system in the postulation of the basic finite-state model. The role of state in a finite state model can be described through the following two statements:

- The out symbol at the present sampling time is uniquely determined by the input symbol and state at the present sampling time.
- The state at next sampling time is uniquely determined by the input symbol and state the present sampling time.

Thus, roughly, for a finite state machine at any given sampling time, if and only if state and input variables are known, we can predict the output variables at this sampling time and the state variables at the next sampling time.

Like input and variables, we can use $s_v^{(k)}, k = 1, 2, \ldots, n$, to denote state variables at the vth sampling time t_v, also, we can replace all the output variables $s^{(1)}, s^{(2)}, \ldots, s^{(n)}$ by a single input variable s, whose alphabet is defined as:

$$S = S^{(1)} \otimes S^{(2)} \otimes \cdots \otimes S^{(l)} \tag{2.1.5}$$

where each state variable $s^{(k)}, k = 1, 2, \ldots, n$, has an alphabet $S^{(k)}$.

The State Space Modelling process

With the idea of a system state in mind, we are now in a position to enhance the modelling process of Figure 2.2. In addition of selecting input and output variables, we can also identify state variables. The modelling process then consists of determining suitable mathematical relationships involving the input $u(t)$, the output $y(t)$, and the state $x(t)$. In particular, we are interested in obtaining expressions for $x(t)$ given $x(t_0)$ and the input function $u(t)$, $t \geq t_0$.

Definition 2.1.2. The set of equations required to specify the state $x(t)$ for all $t \geq t_0$ given $x(t_0)$ and the function $u(t)$, $t \geq t_0$, are called *state equations.*

Definition 2.1.2 leads to the following Definition 2.1.3 for the *state space*:

Definition 2.1.3. The state space of a system, usually denoted by X, is the set of all possible values that the state may take.

The state equations could take several different forms. Most of systems, theory of finite state machine and control theory, however, are based on differential equations of the following form:

$$x_{k+1} = f(x_k, u_k, k)$$

$$y_k = g(x_k)$$

An important point is that the selection of a state in any given problem is not unique. Finally, Definition 2.1.4 gives a definition of a finite state machine.

Definition 2.1.4. A finite-state machine M is a synchronous system with a finite input alphabet $U = \{u_1, u_2, \ldots, u_p\}$, a finite output alphabet $Y = \{y_1, y_2, \ldots, y_q\}$, a finite state set $X = \{x_1, x_2, \ldots, x_r\}$, and a pair of characterising functions g_Y and f_X, given by:

$$\begin{cases} x_v + 1 = & f_X(x_v, u_v, v) \\ y_v = & g_Y(u_v, x_v) \end{cases}$$

where u_v, y_v, and x_v are, respectively, the input, output, and state variables of M at the v-th sampling time t_v.

2.1.2 Queueing systems

Queues are important components in many discrete event systems. They are mainly used if we are faced with limited resources. As a consequence, entities have to wait until they can be served (e.g., people waiting in a line for a bank teller). A simple queueing system is depicted in Figure 2.5.

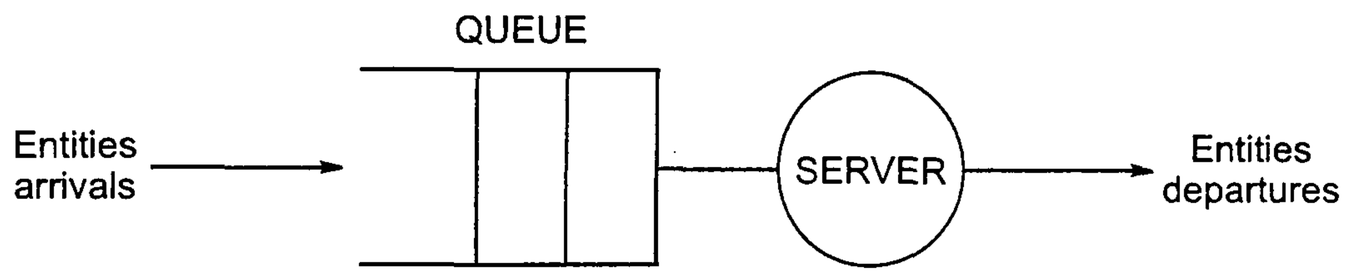

Figure 2.5: A simple queueing system

In this diagram the circle represents servers and the left-opened rectangle a queue preceding the server. The slots in the rectangle represent the number of entities waiting to be served. Entities arrive to the queue and wait to be served. Examples of entities are people, tasks or jobs, while examples of servers include people, various types of machines, provided services etc. The basic idea behind queueing systems is that resources are unlimited and can be accessed in fair and efficient ways among different entities. Their main characteristics are. (i) the capacity of the queue and (ii) queueing discipline. The capacity of the queue is the maximum number of

waiting entities *(queue length)* and in many systems is assumed to be finite. Queueing discipline indicates the rule according to which the next entity to be served is selected from the queue. One widely used rule is called First-In First-Out *(FIFO)* as the entities are served in the ordering of their arrival. Other possibilities also exist such as Last-In-First-Out *(LIFO)* where the entities are served in reverse order of their arrival, or service in random order *(SIRO)*. Another property we may need to specify in queueing models is the timing sequencing of events, as a server may need some time to deal with an entity, i.e. to process an event. A typical discrete event model of a queue involves events with values $V = \{a, d\}$ where a denotes the arrival of a entity and d denotes its departure. A typical state variable x may denote the queue length, i.e. $\mathbb{X}= \{0, 1, 2, 3, \ldots\}$. A queueing system may be studied according to the waiting entity or in respect to the service provider.

Analysis of a simple Queue

We consider the simple queueing system depicted in Figure 2.5 with infinite storage space and a single server. The arrival process is associated with arrival events α of a stochastic sequence $S = \{Y_1, Y_2, Y_3, \ldots\}$ where Y_k is the kth inter-arrival time (time elapsed between $(k-1)$th and kth arrivals) and Y_1 is the time of the first arrival. If we assume that Y_k's are independent and identically distributed (iid), then a single probability distribution $A(t) = P[Y_k \leq t], k \in \mathbb{N}^* - \{0\}$ can describe completely the inter-arrival time sequence. The mean of the distribution function $A(t)$ is $\mu_a = \frac{1}{\lambda}$, where λ is the average arrival rate of entities.

Since we now know the entities' arrivals we must calculate the servicing time. We can associate with d the servicing events of a stochastic sequence $R = \{Z_1, Z_2, Z_3, \ldots\}$, where Z_k is the kth servicing time (time elapsed between $(k-1)$th and kth services) and Z_1 is the time of the first service. If we assume that the Z_k's are (iid), then a single probability distribution $B(t) = P[Z_k \leq t], k \in \mathbb{N}^* - \{0\}$ can similarly describe the servicing time sequence. The mean of the distribution function $B(t)$ is $\mu_d = \frac{1}{\mu}$ so that μ is the average service rate of the server in the queueing

system. In queueing systems we are also interested in calculating the *traffic density* ϱ, which gives information about the utilisation of the system. We define $\varrho = \frac{\lambda}{\mu}$, $0 \leq \varrho < 1$.

Queueing systems are in general more complex than the system shown in Figure 2.5 and their analysis is more involved. The storage capacity of a queue usually denoted by K and the number of servers m define the structure of a queueing system. The design of the service facility may include parallel servers or servers in series. In the simple queueing system of Figure 2.5 $K = \infty$ and $m = 1$. There is a standard notation system (Kendalls notation) to classify queueing systems as A/B/m/K/p/E, where:

A : is the probability distribution for the interarrival time

B : is the probability distribution for the service time

m : is the number of servers, m=1,2,...

K : is the number of entities, K=1,2,...

p : is the system population

E : is the queueing discipline

Using above notation the system in Figure 2.5 is described by $A/B/1/\infty/$, while system population and system discipline can vary. A common notation is also used for the A and B distributions:

M : Poisson arrival distribution (exponential interarrival/service distribution)

D : General distribution (with deterministic or constant interarrival/service times)

G : General distribution (with an unknown mean and variance)

GI : General distribution where interarrival/service times are iid

Queueing systems can be open or closed. In an open queueing system the number of arrival entities is infinite, while a closed queueing system (shown in Figure 2.6) can serve a finite number of entities. In this system an entity after being served always returns for more services and never leaves the system.

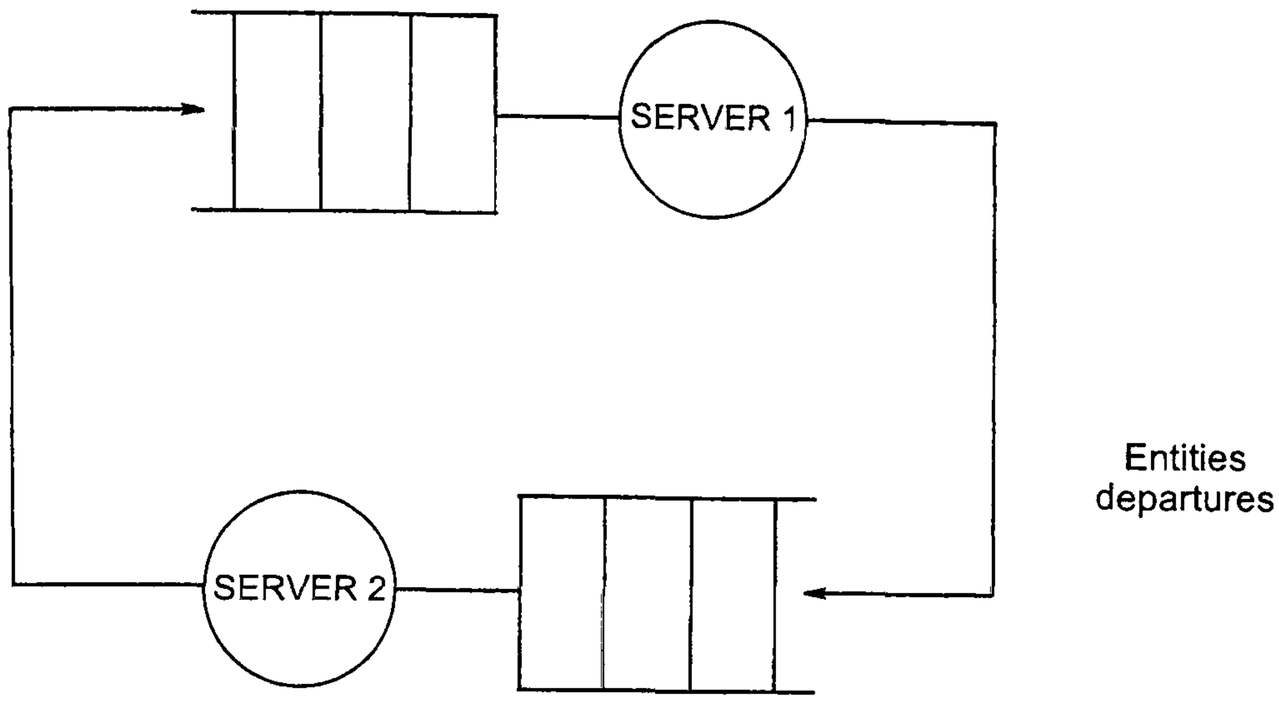

Figure 2.6: A closed queueing system

In the case of a closed queueing system *traffic density* ϱ_c, is given by $\varrho_c = \frac{\lambda}{m\mu}$, where m is the number of servers. More details concerning queueing theory and more complex queueing systems can be found in [CL99], [Bos01], [GH98].

2.1.3 Petri Nets

Petri Nets (PNs) is one of the mathematical and graphical modelling tools well suited for describing and analysing discrete events systems (DES). Petri nets allow us to model, verify, implement and visualise systems which contain concurrence, resource sharing or synchronisation. Petri nets offers numerous advantages for modelling (DES) systems and have been used in many different application areas with a high degree of success. A large variety of powerful and universal university and commercial tools have been developed for the analysis of Petri nets, providing models and algorithms to meet the needs in different application domains and achieve industrial or business standards. Petri nets were first introduced in 1962 by Carl Adam Petri [Pet62], while a major extension of his work was carried out at MIT in

the early 70's. In the 90's Petri nets were first used in industry to model flexible manufacturing systems and now they can be found in a wide variety of applications. Recently, Petri nets were used for design, modelling and performance analysis of supply chain systems, especially as a tool for computation of key factors of supply chains such as lead times, customer satisfaction and inventory levels.

There are several different types of Petri, whose use each time depends on the main attributes of the application or the characteristics of the problem. Nevertheless, it would be impossible to discuss all these different types here. For the purposes of this research we use *Coloured Petri nets* (CP-nets or CPN). On the following section we introduce the main concepts of Petri nets and we restrict our attention to the principles of Timed Coloured Petri nets (TCPN).

Petri net notation and definitions

Petri Nets are bipartite directed multigraphs with two types of nodes; (i) circles which represents the states of the system called *places*, and (ii) bars which are associated with the events and are called *transitions*. These two different nodes are joined by directed *arcs* which can are connect places to transitions and transitions to places. A simple Petri net is depicted in Figure 2.7.

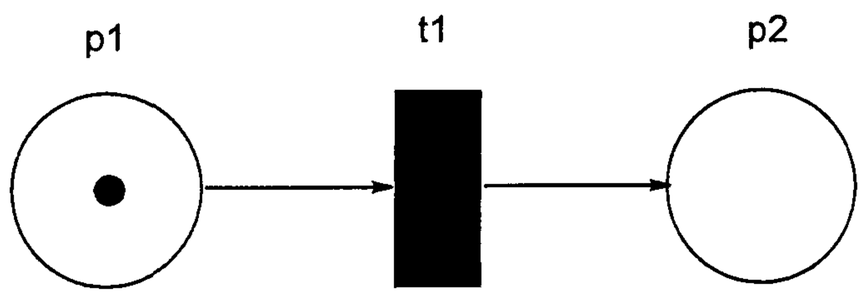

Figure 2.7: A simple Petri net

In order for a transition to occur several conditions have to be satisfied. These conditions are stored in data associated with places which are viewed as the input to a transition. After the occurrence of a transition, places may be enabled which is viewed as the output of a transition. In Figure 2.7 p_1 and p_2 are the input and output places for the transition t_1, respectively. A transition can stand for a processor, event,

computation step or algorithm, task or job. An input place can represent buffers, preconditions, input data, conditions or input signals. Similarly output places can stand for buffers, post-conditions, output data, conclusions or output signals. A marked PN contains *tokens*. Tokens are depicted graphically by dots and reside in places. A *marking* of a PN is a mapping that assigns a non negative integer (the number of tokens) to each place of the net. However, in high-level Petri nets such as CP-nets tokens can be not only non negative integers, but also boolean expressions or strings. The marking characterises the state of the PN. The existence of one or more tokens represents the availability of a resource, while the absence of tokens in places indicates that the resource is not available. The places and the marking capture the distributed nature of the system. The marking M can be represented by a token (i.e., $M(p_1) = 1, M(p_2) = 0$). In Figure 2.7 place p_1 contains a single token and therefore the current snapshot represents also the current marking of the PN. A formal definition of a Petri net is given next.

Definition 2.1.5. A Petri net is a four-tuple:

$$(P, T, A, M)$$

where:

P : is a finite set of places.

T : is a finite set of transitions.

A : is the set of arcs from places to transitions such that $A \subseteq (P \times T) \cup (T \times P)$.

M : is the marking of the set of the places P.

Definition 2.1.5 states the main components of the ordinary Petri nets. In theory, ordinary and high-level Petri nets have exactly the same computational power but in practice, high-level nets have much more modelling power because they have

better structuring facilities, e.g. types and modules. With respect to description and simulation the two models are nearly identical but according to formal verification there are some differences. Each CP-net has an equivalent PT-net and vice versa. This equivalence is used to derive the definition of basic properties and to establish the verification methods.

Several other kinds of high-level Petri Nets also exist. They all rely on the same basic ideas, but use different languages for declarations and inscriptions. Description, simulation and verification for the purposes of this research work are all done directly in terms of CP-nets which are discussed in the next section.

Coloured Petri nets

As it has been mentioned in the previous section, ordinary Petri nets have no types and no modules, but only one kind of tokens. With Coloured Petri Nets (CP-nets) it is possible to use data types and complex data manipulation. Each token has a data value attached called the *token colour*. The token colours can be investigated and modified by the occurring transitions. In CP-nets places are defined by ellipses while transitions are represented by quadrangles. Each place has the following inscriptions: Name (for identification), *colour set* (specifying the type of tokens which may reside on the place), and an *initial marking* (multi-set of token colours). Each transition has the following inscriptions: *Name* (for identification) and a *guard* (boolean expression containing some of the variables). An arc may carry an expression which describes how the state of the CP-net changes when the transitions occur. When the arc expression is evaluated it yields a multi-set of token colours. A simple CP-net is shown in Figure 2.8.

CP-nets in contrast with ordinary Petri Nets can combine text, graphics and the use of a formal language (e.g. a programming language). Declarations and net inscriptions are specified by means of this formal language. To make a CP-net readable it is important to make a nice graphical layout, although this has no formal importance.

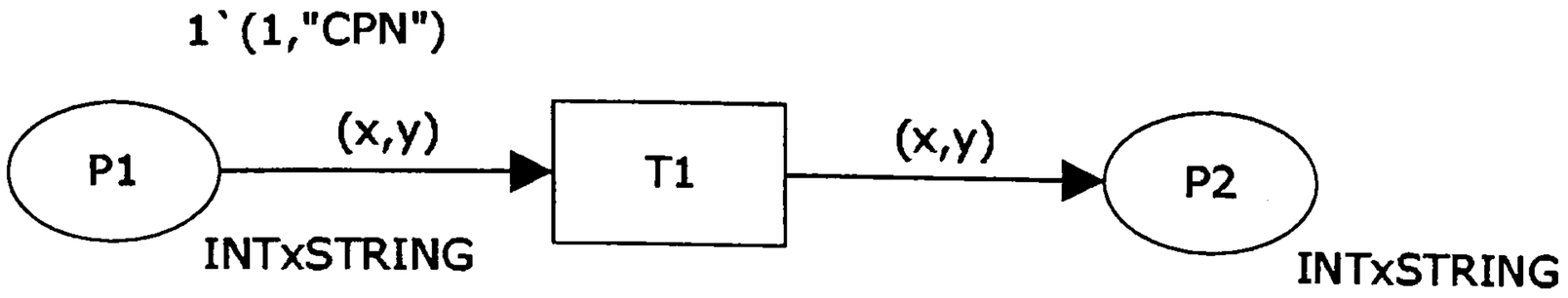

Figure 2.8: A simple Coloured Petri net

A *binding* assigns a token colour (i.e. a value) to each variable of a transition. A binding element is a pair (T_n, b_i) where T_n is a transition while b_i is a binding for the variables of T_n. For the simple Petri net depicted in Figure 2.8 a binding for the transition T_1 is given as follows: $(T_1, < x = 1, y = "CPN" >)$. For any transition to occur, we must bind these two variables (these could be one or more) to values in their types in such a way that the arc expression of each incoming arc evaluates to a token value that is present in the corresponding input place. A binding element is enabled iff: (i) there are enough tokens (of the correct colours on each input-place), and (ii) the guard evaluates to true. When a binding element is enabled, a multi-set of tokens is removed from each input-place, and a multi-set of tokens is added to each output-place. A binding element may occur concurrently to other binding elements iff there are so many tokens that each binding element can get its "own share". The formal definition of a non-hierarchical CP-net is given below [Jen97]:

Definition 2.1.6. A Coloured Petri net is a tuple CPN:

$$(\Sigma, P, T, A, N, C, G, E, I)$$

where: satisfying the requirements below:

1. Σ is a finite set of non-empty types, called colour sets

2. P is a finite set of places

3. T is a finite set of transitions

4. A is a set of arcs from places to transitions such that:

   - $P \cap T = P \cap A = T \cap A = \varnothing$.

5. N is a is a node function. It is defined from A into $P \times T \cup T \times P$

6. C is a colour function. It is defined as a mapping from P to Σ.

7. G is a guard function. It is defined as a mapping from T to expressions such that:

   - $\forall t \in T : [Type(G(t)) = \mathbb{B} \wedge Type(Var(E(a))) \subseteq \Sigma]$

   - where: $Type(G(t))$ is the type of the guard $G(t)$ and $Var(E(a))$ is the set of variables in $E(a)$.

8. E is an arc expression function. It is defined as a mapping from A into expressions such that:

   - $\forall a \in A : [Type(E(a)) = C(p(a))_{MS} \wedge Type(Var(E(a))) \subseteq \Sigma]$

   - where: $p(a)$ is the place of $N(a)$.

9. I is an initialisation function. It is defined as a mapping from P into closed expressions such that:

- $\forall p \in P : [Type(I(p)) = C(p)_{MS}]$.

The set of colours determines the types, operations and functions that can be used in the net inscription (i.e. guards, arc expressions, colour sets etc.). If desired, the colour sets can be defined by means of a many-sorted sigma algebra. We assume that each colour set has at least one element. Places, transitions and arcs are required to be finite and pairwise disjoint. In contrast with ordinary Petri nets, the net structure for pragmatic reasons can be empty (i.e. $P \cup T \cup A = \varnothing$). Hence the user can define and syntax-check a set of colour sets without having to invent a "dummy" net structure.

The node function maps each arc into a pair where the first element is the source node and the second the destination node. In contrast with classical Petri nets, a CPN can have several arcs between the same ordered pair of nodes. The colour function C maps each place, p, to a colour set $C(p)$ and therefore each token on p must have a token colour that belongs to the type $C(p)$. The guard function G maps each transition, t, to an expression of type boolean, i.e., a predicate. Moreover, all variables $G(t)$ must have types that belong to Σ. The arc expression function E maps each arc, a, into an expression which must be of type $C(p(a))_{MS}$. This means that each evaluation of the arc expression must yield a multi-set over the colour set that is attached to the corresponding place. Arc expressions can be omitted and this leads to a shorthand of empty. Finally, the initialisation function I maps each place, p, into a closed expression which must be of type $C(p)_{MS}$, i.e., a multi-set over $C(p)$. Analogously to arc expression an initial expression can be also omitted.

Hierarchical Coloured Petri nets

With CP-nets it is possible to make hierarchical descriptions. A large model can be obtained by combining a set of submodels. Hierarchical Coloured Petri nets (HCPN) allow a modeler to construct a large model by combining a number of small CP-nets

into a larger net. HCPN are very useful when coping with large systems, like supply chains, when there is a need to develop strong structuring and abstraction concepts. The basic idea behind HCPN is that we can relate a transition (called substitution transition) and its connecting arcs to a more complex CP-net, which usually gives a more precise and detailed description of the activity represented by the substitution transition. Thus, a HCPN contains a number of interrelated subnets called pages. A page may contain one ore more substitution transitions.

Each substitution transition is related to a page, i.e., a subnet providing a more detailed description than the transition itself. The page is a subpage of the substitution transition. There is a well-defined interface between a substitution transition and its subpage. The places surrounding the substitution transition are called socket places and a subpage may contain a number of port places. Socket places are related to port places in a similar way as actual parameters are related to formal parameters in a procedure call. A socket place has always the same marking as the related port place. The syntax and semantics of hierarchical CP-nets have formal definitions similar to the definitions for non-hierarchical CP-nets. Each hierarchical CP-net has an equivalent non-hierarchical CP-net and vice versa. The two kinds of nets have the same computational power although hierarchical CP-nets have much more modelling power.

Definition 2.1.7. A Hierarchical Coloured Petri net is a tuple HCPN:

$$(S, SN, SA, PN, PT, PA, PP)$$

satisfying the requirements below:

1. S is a finite set of pages such that:

   - Each page $s \in S$ is a non-hierarchical CP-net:

     $(\Sigma_s, P_s, T_s, A_s, N_s, C_s, G_s, E_s, I_s)$

- The sets of net elements are pairwise disjoint:

  $$\forall s_1, s_2 \in S : [s_1 \neq s_2 \Rightarrow (P_{s_1} \cup T_{s_1} \cup A_{s_1}) \cap (P_{s_2} \cup T_{s_2} \cup A_{s_2}) = \varnothing]$$

2. $SN \subseteq T$ is a set of substitution nodes

3. SA is a page assignment function. It is defined from SN into S such that:

   - No page is a subpage of itself:

     $$\{s_0 s_1 \ldots s_n \in S^* | n \in \mathbb{N}_+ \wedge s_0 = s_n \wedge \forall k \in 1 \ldots n : s_k \in SA(SN s_{k-1})\} = \varnothing$$

4. $PN \subseteq P$ is a set of port nodes.

5. PT is a port type function. It is defined from PN into {in, out, i/o, general}.

6. PA is a port assignment function. It is defined from SN into binary relations such that:

   - Socket nodes are related to port nodes:

     $$\forall t \in SN : PA(t) \subseteq X(t) \times PN_{SA(t)}$$

   - Socket nodes are of the correct type:

     $$\forall t \in SN \forall (p_1, p_2) \in PA(t) : [PT(p_2) \neq general \Rightarrow ST(p_1, t) = PT(p_2)]$$

   - Related nodes have identical colour sets and equivalent initialization ex-

pressions:

$$\forall t \in SN \forall (p_1, p_2) \in PA(t) : [C(p_1) = C(p_2) \wedge I(p_1) <>= I(p_2) <>].$$

7. $PP \in S_{MS}$ is a multi-set of prime pages.

Each page is a non-hierarchical CP-net, and we require that none of these have any net elements (places, transitions, and arcs) in common. Each substitution and port node are transition and place, respectively. A page may have port nodes even when it is not a subpage. The page assignment relates substitution transitions to their subpages and no page is a subpage of itself, while the port type divides the set of port nodes into input, output and general ports. The port assignment relates socket nodes with port nodes. Each related pair of socket/port nodes must have matching socket/port types. Moreover, they must have identical colour sets and equivalent initialisation expressions. Initialisation expressions are not required to be identical, but it is required that they evaluate to the same value. Note also that it is possible to relate several sockets to the same port, and vice versa. The prime pages is a multi-set over the set of all pages and they determine - together with the page assignment - how many instances the individual pages have. Often the prime page multi-set contains only a single page (with coefficient one).

Simulation of Coloured Petri nets

Petri Nets are executable. The graphical nature of Petri Nets allows the visualisation of the complexity of the system. Petri Nets capture the precedence relations and structural interactions of concurrent and asynchronous events. Petri Nets also subsume many other discrete event dynamical system models.

CP-nets integrated with a set of robust computer tools provide (i) Construction and modification of CPN models, (ii) Syntax checking (e.g., types and module interfaces), (iii) Interactive simulation, to gain additional understanding of the

modelled system, (iv) Debugging procedures, (v) Automatic simulations, e.g., to obtain performance measures, (vi) Verification to prove behavioural properties, and (vii) State spaces (also called reachability graphs and occurrence graphs).

In current research work we use CPN-Tools - developed by CPN Group at University of Aarhus in Denmark. In this computer tool, when a syntactical correct CPN diagram has been constructed, the CPN tool generates the necessary code to perform simulations. The simulation code calculates whether the individual transitions and bindings are enabled and the effect of occurring transitions and bindings. The code generation is incremental. Hence it is fast to make small changes to the CPN diagram. These CPN-tools have two kinds of simulations: An interactive simulation the user is in control, but most of the work is done by the system, and an automatic simulation where the system does all the work. A powerful simulation tool is a very important issue. If we consider a supply chain system with a vast number of states, the simulation, analysis and optimisation require a large amount of computational effort, while problems of realistic scale quickly become analytically and computationally intractable.

CP-nets use standard (Markup-Language) ML declarations, net inscriptions and code segments are specified in a programming language called *Standard ML*. ML is strongly typed, functional language, while data types can be integers, reals, strings, booleans and enumerations or structured types such as products, records, unions, lists and subsets.

Time analysis of CP-nets can be extended by introducing a time concept. This means that the same language can be used to investigate logical correctness, desired functionality, absence of deadlocks, performance, and can also remove bottlenecks, predict mean waiting times and average throughput, and compare different strategies. In a timed CP-net each token carries a colour (data value) and a time stamp (telling when it can be used). Time stamps are specified by expressions. Time stamps can depend upon colour values and can be specified by probability distributions. This means that we can specify for instance fixed delays, interval delays and exponential

delays. Timed simulations have the same facilities as untimed simulations, e.g. we can switch between interactive and automatic simulation. Simulation reports tell the time at which the individual transitions occurred. More about Coloured Petri Nets and CPN-Tools can be found in [Jen97], [KCJ98], and [fCPN].

2.2 Control problems in Supply Chains

Systems and control theory plays an intrinsic role in a wide range of technological areas. There is a strong growth in applications of control technology in production processes and the business environment. Systems theory is primarily driven by the desire for fundamental insight. Control technology is primary motivated by engineering problems, with technical relevance and feasibility as the main constraints. Nowadays, control techniques are used widely in complex systems like supply chains, where companies need to integrate two decision levels: control and planning. At the planning level of supply chains a coherent integrated planning of all nodes is needed. This integration not only applies to the material flows from raw material suppliers to finished product delivery, but also to the inventory control of each node, and information and product flows from the customer back to supply chain participants (nodes). Modern supply chain control methods are focused on inventory control and use of control theory to analyse the dynamic behaviour of the interactions between different parts of the chain.

2.2.1 Inventory control

An inventory model in a supply chain node represents the behaviour of an inventory system over time. It represents the diminishing inventory as goods are dispatched to the downstream level and the increase of inventory as goods are replenished following various inventory policies. Inventory control systems are designed primarily to cope with ranging demand situations by ensuring smooth flow of goods through the entire supply chain and maintaining low holding-costs. An inventory model

according to demand can be either deterministic or probabilistic. In a deterministic model demand from the downstream level is assumed to be known with certainty while probabilistic inventory models are stochastic and the demand is described by a probability distribution. Both models can be further classified according to continuous or periodic review policies.

Quantitative models for inventory control

The objective of any inventory manager in a company is to maintain the inventory at the lowest possible level to minimise cost, while meeting order demands from the downstream nodes. Ordering the right amount at the right time is a constant headache for managers, while the nature of the inventory problem consists of placing and receiving orders repeatedly. In order to tackle this inventory problem, many managers use a mathematical technique to determine the lowest total variable costs needed to order and hold inventory. This technique is called *Economic Order Quantity* (EOQ).

1. Deterministic inventory models.

If we consider the simplest inventory model with a constant rate demand, instantaneous order replenishment and no shortages we can define following Taha [Tah03]:

$$y : Order\ quantity\ (number\ of\ units)$$

$$D : Demand\ rate\ (units\ per\ unit\ time)$$

$$t_0 : Ordering\ cycle\ length\ (time\ units)$$

Since the order quantity y is known, we can easily find the ordering cycle length t_0. Then, our aim is to find optimal values for y and t_0. Changes on inventory level as a function of time are depicted in Figure 2.9. An order of y units is placed when inventory reaches the zero level, and is assumed to be

met instantly (lead-time is zero) at a constant demand rate D. Since no stock remains after the ordering cycle length, we need to place a new order. We know also that during the cycle the amount entering the inventory is y while the amount leaving is $D \times t_0$. Since these two entities are equal, then:

$$y = t_0 \bullet D \Rightarrow t_0 = \frac{y}{D} \text{ time units}$$

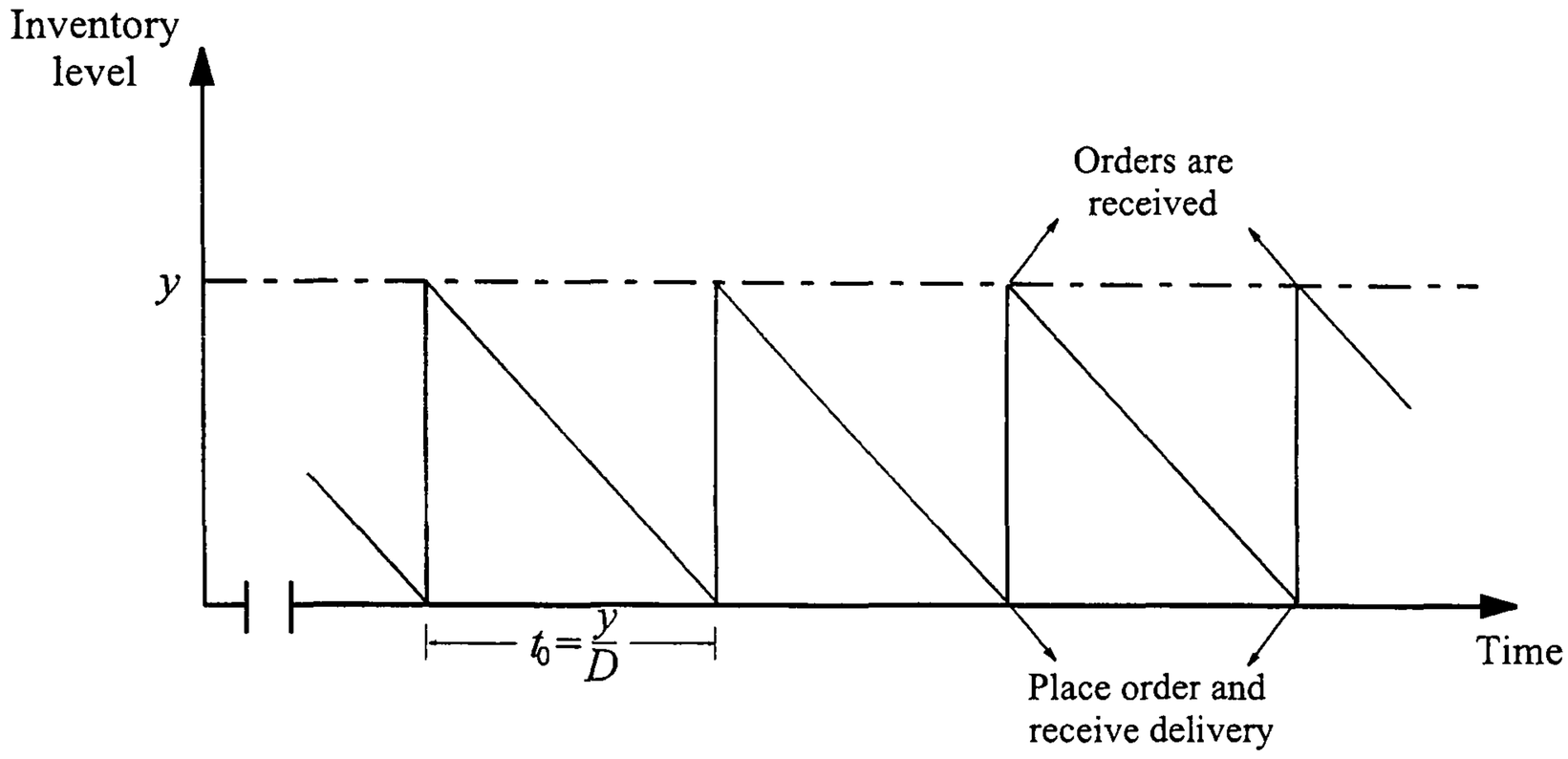

Figure 2.9: Inventory level with fixed order size

The resulting average inventory level is $\frac{y}{2}$ *units*.

The first step of the analysis is to find the total cost of the cycle. Hence, we define the following two cost parameters:

K : *Setup cost associated with the placement of an order* (*reorder cost*)

h : *Holding cost per inventory unit per unit time*

The *total cost per unit time* (TCU) is then computed as:

$$\begin{aligned} TCU(y) &= Setup\ cost\ per\ unit\ time + Holding\ cost\ per\ unit\ time \\ &= \frac{K + h(\frac{y}{2})t_0}{t_0} \\ &= \frac{K}{(\frac{y}{D})} + h(\frac{y}{2}) \end{aligned}$$

The optimum value of the order quantity y can be computed by minimising TCU(y) with respect to y. Assuming that y is continuous, the minimum cost per unit time is given by:

$$y^* = \sqrt{\frac{2KD}{h}} \tag{2.2.1}$$

Thus, the above equation defines the optimal order size every $t_0^* = \frac{y^*}{D}$ time units.

In cases where a constant finite *lead-time* L occurs between the placement and receipt of an order, as shown in Figure 2.10, there is no benefit to order at the end of each cycle since each order should be timed to arrive just as existing stock runs out. To achieve this, we have to place an order a time L before the delivery is needed. Thus we should define a reorder level such that when the current inventory drops to this level an order is placed.

Figure 2.10 assumes that lead time L is less than the ordering length cycle time t_0^* which is not necessarily the case in general. In order to calculate the lead time for these special cases, we define the *effective* lead time as $L_e = L - nt_0^*$, where n is a cycle integer $n \leq \frac{L}{t_0^*}$. It can be inferred from the equation 2.2.1 that EOQ does not depend on lead time.

In case of price breaks (i.e., when the inventory item is purchased at a discount price if the order size exceeds a given limit q) the EOQ $\hat{y}$ is: (see [Tah03])

$$\hat{y} = \begin{cases} y^*, & q \in (0, y^*) \cup q \in (Q, \infty) \\ q\,, & q \in (y^*, Q) \end{cases} \tag{2.2.2}$$

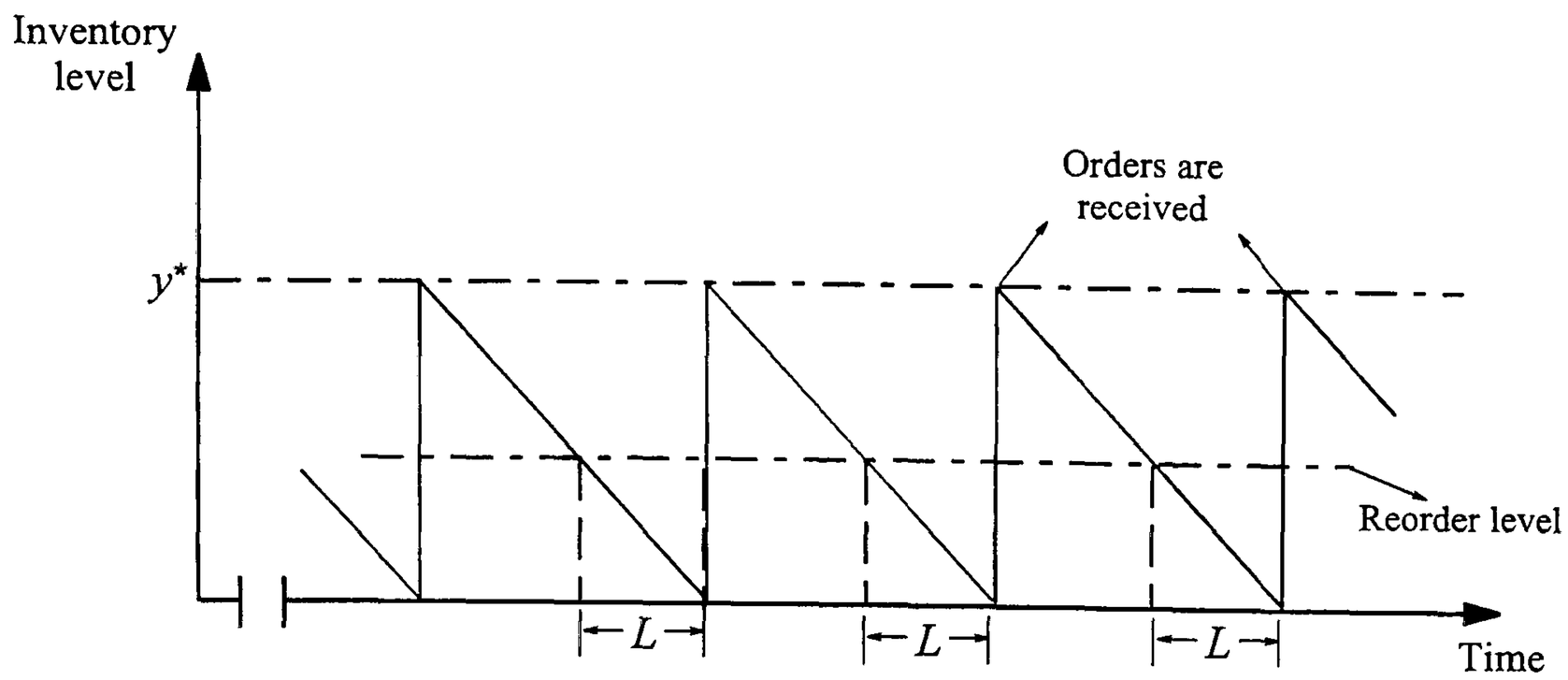

Figure 2.10: Inventory level with lead time and the corresponding reorder level

where $Q > y^*$.

Lemma 2.2.1. *The EOQ y_i^* for n items $(n > 1)$ with limited storage facilities is given as:*

$$y_i^* = \sqrt{\frac{2K_iD_i}{h_i - 2\lambda^*\alpha_i}} \tag{2.2.3}$$

where i is the index of the ith item, α_i is the storage area requirement per inventory unit for the ith item and λ is the Lagrangian multiplier $(\lambda < 0)$.

Proof. See Appendix A □

2. Probabilistic inventory models.

A probabilistic EOQ model can be derived if we adapt to equation 2.2.1, which reflects the deterministic EOQ model, a probabilistic demand pattern by using an approximation that appends a constant buffer stock on the inventory level. The size of buffer is determined so that the probability of running out of stock

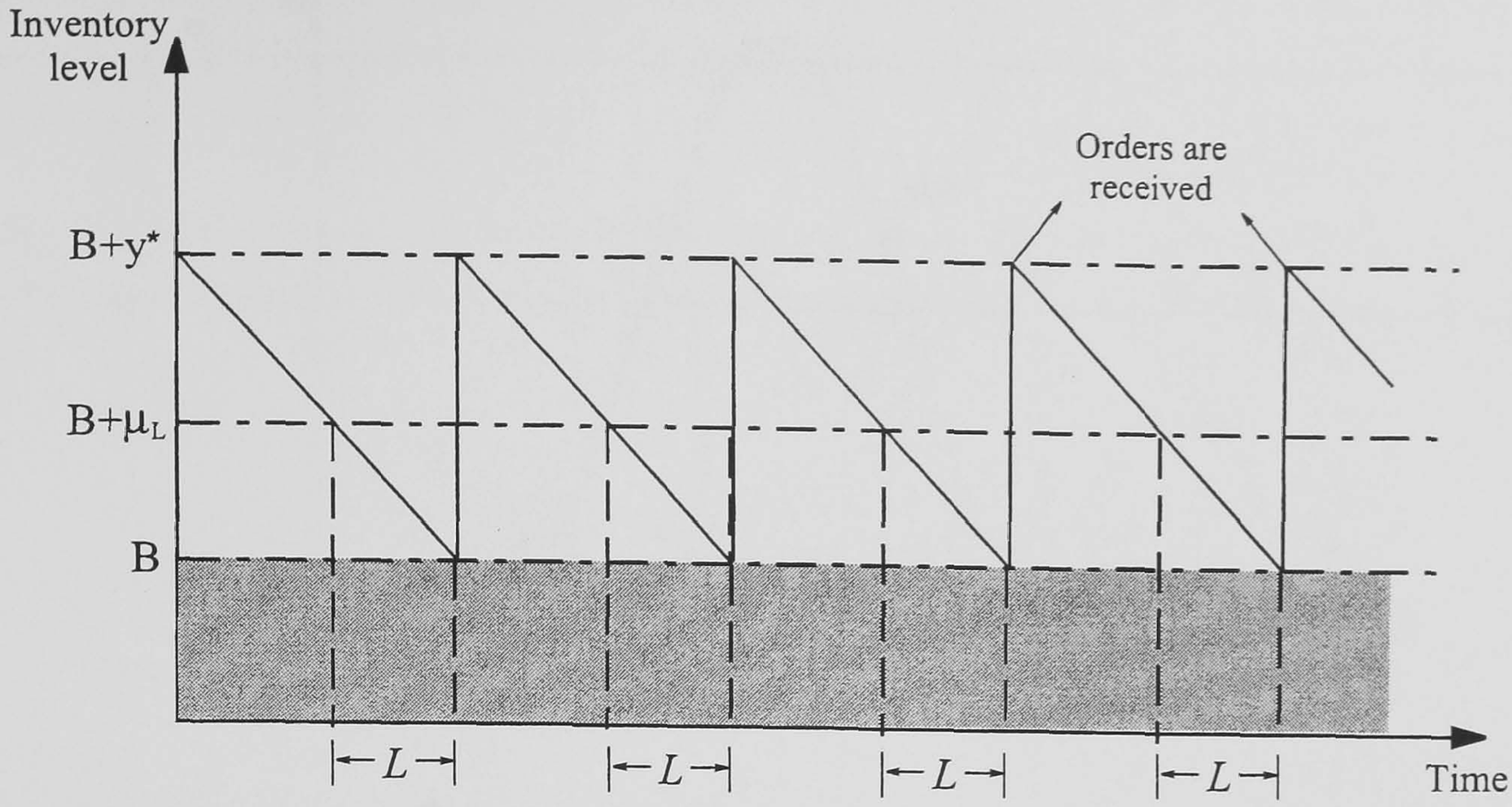

Figure 2.11: Buffer stock imposed on the deterministic EOQ model

during the lead time does not exceed a preassigned value. We define:

B : *Buffer stock size*

L : *Lead time between placing and receiving an order*

x_L : *Random variable representing demand during lead time*

μ_L : *Average demand during lead time*

σ_L : *Standard deviation of demand during lead time*

α : *Maximum allowable probability of running out of stock during lead time*

We assume that demand x_L during lead time L is normally distributed with mean μ_L and standard deviation σ_L,that is $x_L \sim N(\mu_L, \sigma_L)$.

Figure 2.11 shows the relationship between the buffer stock B and the variables of the deterministic EOQ model that include the lead time, L, the average demand during lead time μ_L and y^*. Note that L=L_e.

The probability relation used to determine B can be written as:

$$P\{x_L \geq B + \mu_L\} \leq \alpha$$

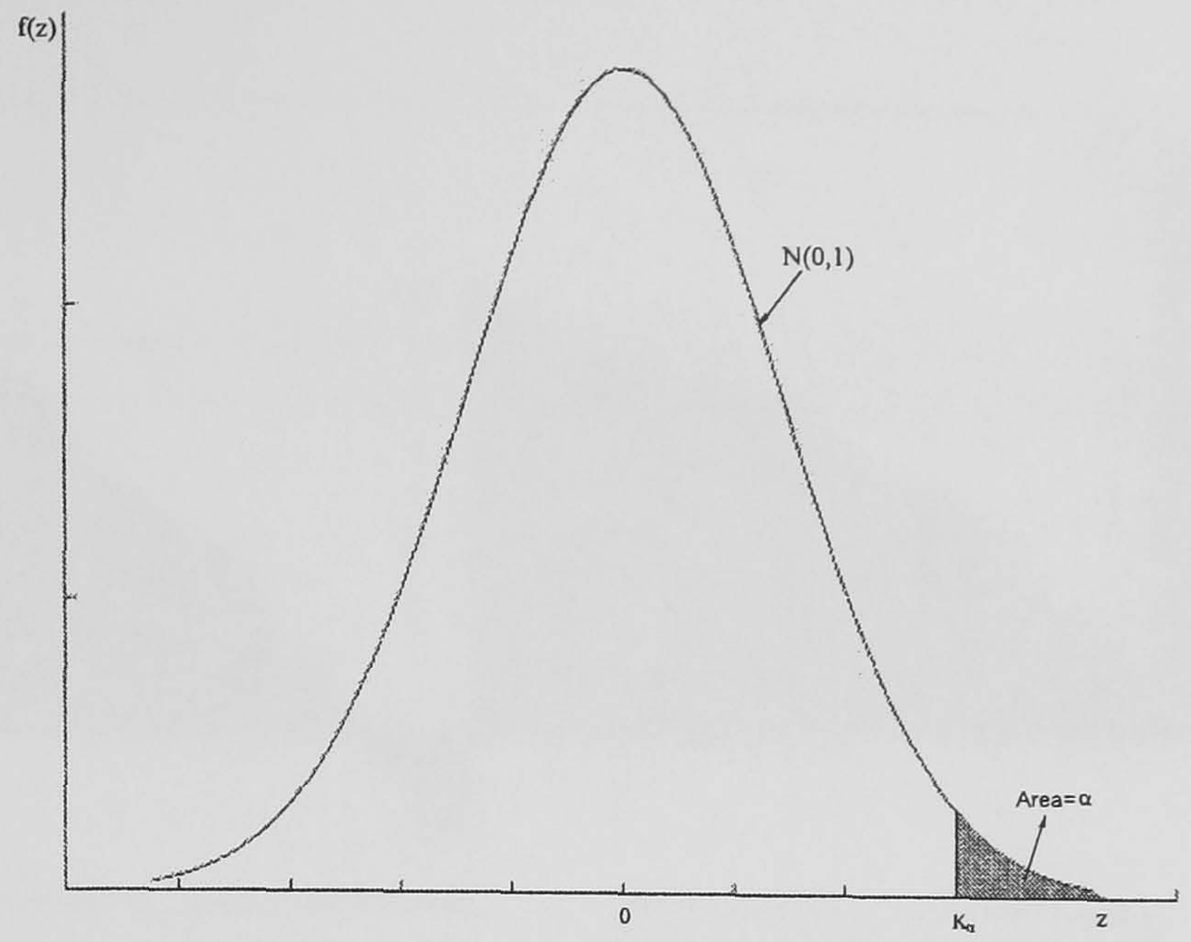

Figure 2.12: Probability of running out of stock $P\{z \geq K_\alpha\} = \alpha$

We can convert x_L into a standard $N(0,1)$ random variable by using the following transformation:

$$z = \frac{x_L - \mu_L}{\sigma_L}$$

Thus,

$$P\{z \leq \frac{B}{\sigma_L}\} \leq \alpha$$

Let K_α be the level at which the grey area depicted in Figure 2.12 is α, i.e.,

$$P\{z \geq K_\alpha\} = \alpha$$

Hence, the buffer size must satisfy:

$$B \geq \sigma_L K_\alpha$$

The demand during the lead time L, usually is described by a probability density function (pdf) per unit time (e.g., per day, or month), from which the distribution of the demand during L can be determined. Given that the demand per unit time is normal with mean μ and standard deviation σ, the

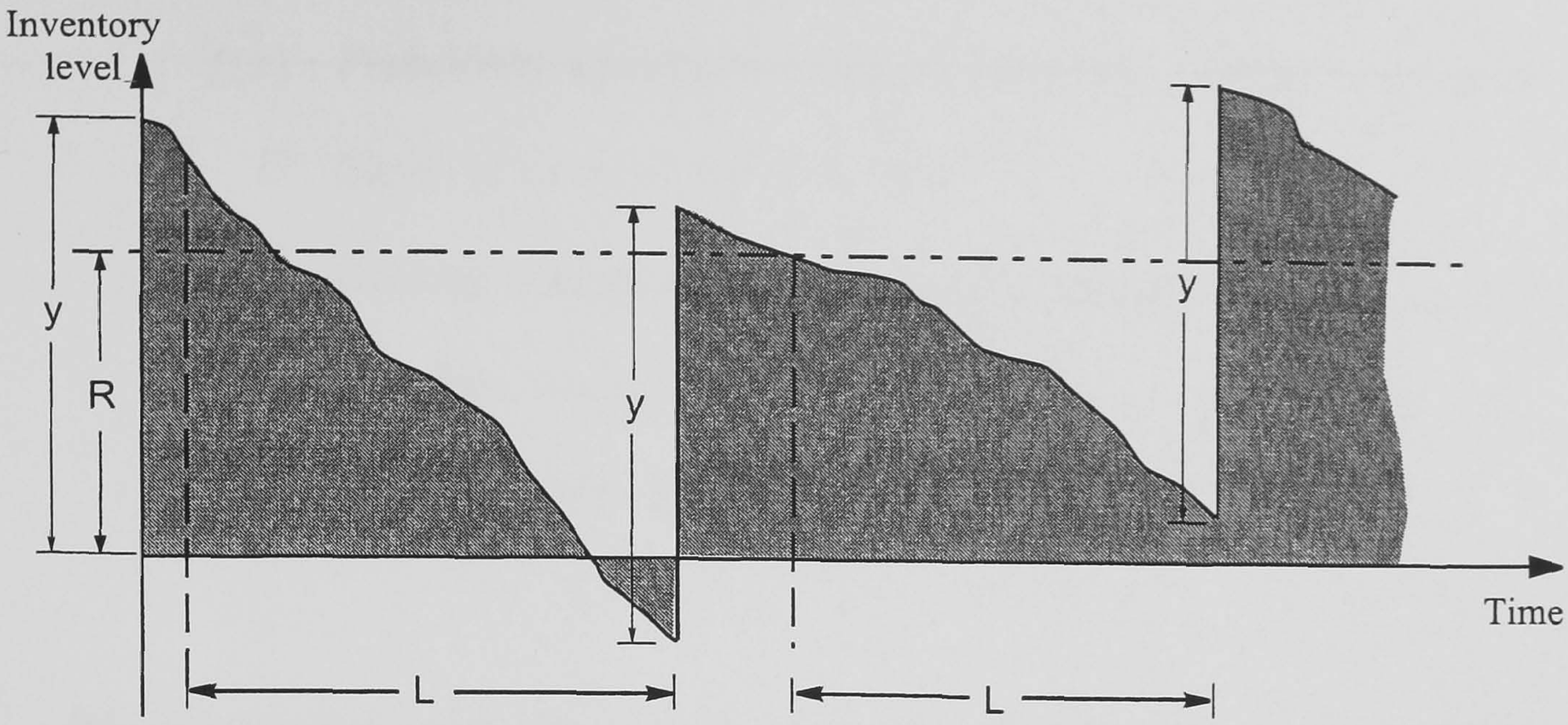

Figure 2.13: Probabilistic inventory model with shortages

mean μ_L and standard deviation σ_L during lead time L, can be calculated as:

$$\mu_L = \mu L$$

$$\sigma_L = \sqrt{\sigma^2 L}$$

Note that the formula for σ_L requires L to be rounded to an integer value.

The EOQ model described above is followed by many managers today despite the fact that the pertinent information regarding the probabilistic nature of demand is initially ignored or is restored at a later stage. There is still some doubt if this is an optimal inventory policy. To remedy this situation we need to develop a more accurate model in which the probabilistic nature of the demand is included directly to the formulation of the model.

Moreover, a more realistic inventory policy must allow shortages of demand as depicted in Figure 2.13 and should call the quantity y whenever the inventory falls to level R. In a similar way to the deterministic case a reorder level R, is a function of the lead time between placing and receipt of an order. The optimal values of y and R are determined by minimising the expected cost per

unit time that includes the sum of the setup, holding and shortage costs. Let:

$f(x)$: *Probability density function of demand x, during lead time*

D : *Expected demand per unit time*

h : *Holding cost per inventory unit per unit time*

ρ : *Shortage cost per inventory time*

K : *Setup cost per order*

We assume that unfulfilled demand during the lead time is backlogged and that no more than one outstanding order is allowed (single ordering policy). We also assume that demand distribution during lead time remains stationary.

In order to find the setup cost we need to calculate the approximate number of orders per unit time $\frac{D}{y}$ so that the setup cost per unit time is approximately $\frac{KD}{y}$. The average inventory I is given by:

$$I = \frac{(y + E\{R - x\}) + E\{R - x\}}{2} = \frac{y}{2} + R - E\{x\}$$

The above formula is based on the average of the beginning and ending expected inventories of a cycle, $y + E\{R - x\}$ and $E\{R - x\}$, respectively. As an approximation, the expression ignores the case where $R - E\{x\}$ may be negative. The expecting holding cost per unit time thus is equal to hI.

Shortages occur when $x > R$. Thus, the expected shortage quantity per cycle is:

$$S = \int_R^\infty (x - R) f(x) dx$$

ρ is assumed to be proportional to the shortage quantity only, the expected shortage cost per cycle is $S\rho$ and based on $\frac{D}{y}$ cycles per unit time, the shortage cost per unit time is $\frac{DS\rho}{y}$.

The resulting total cost function per unit time is:

$$TCU(y, R) = \frac{DK}{y} + h(\frac{y}{2} + R - E\{x\}) + \frac{D\rho}{y} \int_R^\infty (x - R) f(x) dx$$

The solutions for optimal y^* and R^* are determined from:

$$\frac{\partial TCU}{\partial y} = -\frac{DK}{y^2} + \frac{h}{2} - \frac{\rho DS}{y^2} = 0$$
$$\frac{\partial TCU}{\partial R} = h - \frac{\rho D}{y}\int_R^\infty f(x)dx = 0$$

The solution of above equations are given below:

$$y^* = \sqrt{\frac{2D(K+\rho S)}{h}} \tag{2.2.4}$$

$$\int_R^\infty f(x)dx = \frac{hy^*}{\rho D} \tag{2.2.5}$$

Using equation 2.2.4 and equation 2.2.7 we can not calculate y^* and R^* in closed form and therefore we need to use a numeric algorithm, e.g. the algorithm developed by Hadley and Whitin [HW63]. The proposed algorithm, converges in a finite number of iterations provided a feasible solution exists.

For $R = 0$, equation 2.2.4 and equation 2.2.7 become, respectively:

$$\hat{y} = \sqrt{\frac{2D(K+\rho E\{x\})}{h}} \tag{2.2.6}$$

$$\tilde{y} = \frac{\rho D}{h} \tag{2.2.7}$$

If $\tilde{y} \geq \hat{y}$ a unique optimal values of y and R exist. The solution method implies that the smallest value of y^* is $\sqrt{\frac{2kd}{h}}$, which is achieved when $S = 0$.

The implementation of the above algorithm can be realised by following 2 steps:

- Step 1. First we use the initial solution $y_1 = y^* = \sqrt{\frac{2KD}{h}}$ and let $R_0 = 0$. We set $i = 1$ and go to the next step.
- Step 2. Use y_i to determine R_i from equation 2.2.7. If $R_i \approx R_{i-1}$ we terminate the procedure and we consider that the optimal solution is $y^* = y_i$ and $R^* = R_i$. Otherwise, use R_i in 2.2.4 to compute y_i. In this case we set $i = i + 1$ and repeat the second step.

2.2.2 Control theory and supply chain

A supply chain is a dynamic system where inputs $u(t)$ and outputs $y(t)$ are time-dependent. The state of such a system at a time instant t describes the system behaviour at that time in a measurable way. In addition to selecting inputs and outputs we should also identify state variables. In the modelling process then we have to determine suitable mathematical relationships between inputs, outputs and states $x(t)$. These relationships encapsulate the *dynamics* of the system and allow us to obtain expressions for $x(t)$ given $x(t_0)$, and input $u(t)$, $t \geq t_0$. The set of equations required to specify the state $x(t)$ given $x(t_0)$ and $u(t)$ are called *state equations*, while the set of all the possible values $x(t)$ may take is called *state space* of a system and denoted by $\mathbf{X}$.

The modelling process must provide the flexibility of choosing not only inputs and outputs but also the state variables depending on the problem of interest, although there is no unique state representation for a given system. Mathematical relationships between inputs, outputs and states can determine also the linearity or nonlinearity nature of the system. Supply chains are mainly nonlinear system and in practice the analysis of such systems is a cumbersome task. Under certain conditions we can transform a nonlinear system to a linear one; such a system is studied in chapter 3. In the linear case we can represent the state equations of the model as:

$$\begin{aligned} \dot{x}(t) &= Ax(t) + Bu(t) \\ y(t) &= Cx(t) + Du(t) \end{aligned} \tag{2.2.8}$$

In equation 2.2.8 if we assume that there are n state variables, p input variables and s output variables, then A is an $n \times n$ matrix, B is an $n \times p$ matrix, C is an $s \times n$ matrix and D is an $s \times p$ matrix. The elements of these four matrices are called *model parameters.* The study of the above state equations in the linear time-invariant case pertain to the analysis of the system in the time or frequency domain.

In cases where the inputs and outputs variables of the system are defined at

discrete times we obtain a discrete-time system. If we consider a supply chain system where data is recorded only at regular discrete intervals (e.g., monthly) then we obtain a discrete-time supply chain model. In discrete time models, the time instances is a sequence of points $t_0 < t_1 < \ldots t_k$. It is assumed that all intervals have the same length (sampling interval) T, i.e., $t_{k+1} - t_k = t_k - t_{k-1} = T$, $\forall k \in N^+$. The real time variable is now replaced by an integer variable k, which is used as an index to indicate the number of intervals elapsed from a reference point, usually $k = 0$.

We can now define the corresponding input, output and state-vector for a discrete-time model as $u(k)$, $y(k)$, $x(k)$, respectively. Thus the state space equations (2.2.8) become:

$$\begin{aligned} x(k+1) &= Ax(k) + Bu(k) \\ y(k) &= Cx(k) + Du(k) \end{aligned} \tag{2.2.9}$$

where A, B, C, and D are matrices containing the system parameters.

2.2.3 Stochastic models

Stochastic processes

A stochastic process ia a sequence of random variables

$$\{x_t\} = \{\ldots, x_{-1}, x_0, x_1, \ldots\}$$

with joint probability distributions defined for all $t_1, t_2, \ldots, t_n$. Since this description requires too much information to be of practical interest, we typically use the following two statistical characteristics of $\{x_t\}$:

1. The mean:

$$\mu_t = E(x_t) = \int_{-\infty}^{\infty} \xi f_{x_t}(\xi) d\xi$$

where $f_{x_t}(\cdot)$ is the probability density function of x_t, and

2. The covariance function:

$$R(t,s) = E\{(x_t - \mu_t)(x_s - \mu_s)\} = E(x_t x_s) - \mu_t \mu_s = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} xy f_{x_t x_s}(x,y)dxdy$$

where $f_{x_t x_s}(\cdot,\cdot)$ is the joint density function of x_t, x_s.

For a stationary process we assume that $\mu_t = \mu$ for all k and that $R(t,s)$ depends only on the time difference $t-s$; hence to simplify notation we define in this case $R(t,s) = R(t-s,0) = R(t-s)$. A stochastic process of particular interest is the "white noise" sequence $\{x_t\}$. For this process we have: (i) μ_t=0 and (ii) R_t is constant for t=0, and $R(t)$=0 for $t \neq 0$.

Given a stationary stochastic sequence $\{\eta_k\}$ and its covariance function $\{R(k)\}$, its spectral density function is defined as the (two-sided) **Z**-transform of $\{R(k)\}$, i.e.

$$\Phi_{\eta\eta}(z) = \sum_{k=-\infty}^{\infty} R(k) z^{-k}$$

The frequency content of $\{\eta_k\}$ may be obtained by evaluating $\Phi_{\eta\eta}(z)$ for $z = e^{j\omega}$, $|z| \leq \pi$, where ω is the normalised angular frequency measured in rads/sample and hence π corresponds to one half the sampling frequency, i.e.

$$\Psi(\omega) = \Phi_{\eta\eta}(e^{j\omega}) = \sum_{k=-\infty}^{\infty} R(k) e^{-jk\omega} \qquad |\omega| \leq \pi$$

The spectral density function of a white noise sequence is:

$$\Psi(\omega) = R(0) = constant$$

Hence the spectral density function of a white noise sequence does not depend on the normalised angular frequency ω and has "equal content" at all frequencies.

Stochastic signals and models

We usually assume stationary, zero mean noise sequences $\{\eta_k\}$ - note that non-zero means can always be accommodated as a dc offset in the "deterministic" part of the model. This class of models is also too large for practical purposes. A simple widely-used class of models is generated by the output of a linear time-invariant (LTI)

discrete-time system driven by white noise. Hence we can user the LTI discrete-time state-space model (see also equation 2.2.9):

$$\begin{aligned} x(k+1) &= Ax(k) + Be(k) \\ y(k) &= Cx(k) + De(k) \end{aligned} \tag{2.2.10}$$

where $\{e(k)\}$, $k \in \mathbf{Z}$ denotes a white vector-noise sequence of unit intensity, i.e. uncorrelated with $E(e_k) = 0$ and variance $Var(e_k) = \sigma^2$. An LTI system can equivalently be defined via difference equations. The most widely used model is:

- $A(z^{-1})\eta(k) = B(z^{-1})e_k,\ k \in \mathbb{Z}$ (ARMA model) , where $A(z^{-1}) = 1 + a_1 z^{-1} + \ldots + a_n z^{-n}$ and $B(z^{-1}) = b_0 + b_1 z^{-1} + \ldots + b_m z^{-m}$, and its two special cases:

- $\eta_k = B(z^{-1})e_k,\ k \in \mathbb{Z}$ (Moving average (MA) model). The model name derives from the fact that η_k may be written as a (weighted) moving average of present and past samples of the input white noise sequence, i.e. $\eta_k = b_0 e_k + b_1 e_{k-1} + \ldots + b_m e_{k-m}$.

- $A(z^{-1})\eta(k) = e_k,\ k \in \mathbb{Z}$ (Autoregressive (AR) model)

Model name derives from the fact that η_k may be written in regression form with its past values (i.e. "with itself"), i.e.

$$\eta_k = \begin{pmatrix} \eta_{k-1} & \eta_{k-2} & \cdots & \eta_{k-n} \end{pmatrix} \begin{pmatrix} -a_1 \\ -a_2 \\ \vdots \\ -a_n \end{pmatrix}$$

Note we use z^{-1} both as a $\mathbf{Z}$-domain variable and as a time-domain (unit delay) operator. We have made the assumption that $\{\eta_k\}$ must be stationary. We want to investigate the restrictions that this assumption imposes on the coefficients of $Az^{(-1)}$ and $Bz^{(-1)}$, which defines the ARMA model.

Definition 2.2.1. The transfer function $z \to G(z^{-1})$ is: (i) Stable iff $z \to G(z^{-1})$ has all its poles inside the unit circle ∂D, (ii) Minimum-phase iff $z \to G(z^{-1})$ has all its zeros inside ∂D.

Theorem 2.2.2. *If* $\{e_k\}$,$k \in \mathbf{Z}$ *is a white noise sequence with constant variance then* $\eta_k = A^{-1}(z^{-1})B(z^{-1})e_k$ *defines a stationary process iff* $A^{-1}(z^{-1})B(z^{-1})$ *is stable. If* $A^{-1}(z^{-1})B(z^{-1})$ *is stable and has minimum phase, then* $\{e_k\}$ *can be recovered from* $\{\eta_k\}$ *as* $e_k = B^{-1}(z^{-1})A(z^{-1})\eta_k$, *i.e., system is "invertible".*

Note that the ARMA model formally defines a stochastic difference equation. It is important to realise that here the underlying time-index set of this equation is $\mathbf{Z}$ (all integers, both positive and negative). Theorem 2.2.2 shows that when we drive the LTI system by white noise, the output is stationary iff the system is stable.

Spectral properties of ARMA models

Given a process $\{\eta_k\}$ we define the covariance function as:

$$R(k) = E\{\eta_t \eta_{t-k}\},\ k = 0, \pm 1, \pm 2, \ldots$$

and its spectral density function:

$$\Phi_{\eta\eta}(z) = \sum_{k=-\infty}^{\infty} R(k) z^{-k}$$

Note that:

$$\Psi(\omega) = \Phi_{\eta\eta}(e^{j\omega}) = \sum_{k=-\infty}^{\infty} R(k) z^{-j\omega k}$$

is the Fourier transform of the covariance sequence $\{\ldots, R(-1), R(0), R(1), \ldots\}$.

Note also that due to stationarity:

$$R(-k) = E\{\eta_t \eta_{t+k}\} = E\{\eta_{t-k} \eta_t\} = R(k)$$

and this implies that:

$$\Phi_{\eta\eta}(z) = R(0) + \sum_{k=-\infty}^{\infty} R(k)(z^{-k} + z^{k})$$

so that:

$$\Psi(\omega) = \Phi_{\eta\eta}(e^{j\omega}) = R(0) + \sum_{k=-\infty}^{\infty} R(k)(e^{j\omega k} + e^{-j\omega k}) = R(0) + 2\sum_{k=-\infty}^{\infty} R(k)\cos(\omega k)$$

which is a real function of ω (in fact non-negative). The spectral density of the output of a given ARMA model driven by white noise can be defined by the Theorem 2.2.3, see [DV85] .

Theorem 2.2.3. *Suppose that* $G(z^{-1}) = A^{-1}(z^{-1})B(z^{-1})$ *is a stable transfer function. Then* $\{\eta_t\}$ *is defined by* $A(z^{-1})\eta_t = B(z^{-1})e_t$, *where* $\{e_t\}$, $t \in \mathbf{Z}$ *is a white noise sequence of unit variance, has spectral density:*

$$\Phi_{\eta\eta}(z) = \frac{B(z^{-1})}{A(z^{-1})}\frac{B(z)}{A(z)} = G(z^{-1})G(z) \tag{2.2.11}$$

Proof. See [DV85]. □

Note that ARMA models give rise to rational spectral densities. In fact the converse is also true since any rational spectral density function may be generated by an ARMA model driven by white noise. (This is established by showing that any function of the form $\sum_{k=-\infty}^{\infty} R(k)z^{-k}$ with $R(-k) = R(k)$ can be spectrally factored in the form $G(z^{-1})G(z)$ where $G(z^{-1})$ is rational). This result justifies the use of ARMA models. Most noise processes are adequately described by their spectral densities and these can be approximated (to any degree of accuracy) by rational functions.

Note that many ARMA models of the form $A(z^{-1})\eta_t = B(z^{-1})e_t$, $t \in \mathbf{Z}$ correspond to a single spectral density given in 2.2.11. If we impose the restriction, however, that:

$$z \to G(z^{-1}) = \frac{B(z^{-1})}{A(z^{-1})}$$

is stable and minimum phase, the polynomials corresponding to $\Phi_{\eta\eta}(z)$ are uniquely determined (up to a sign).

Calculation of covariance functions for ARMA models

Given an ARMA model $A(z^{-1})\eta_t = B(z^{-1})e_t$ where $\{e_t\}$ is white noise sequence with unit covariance, we sometimes require to obtain $\{R(k)\}$ in closed-form. There are several approaches for this. Consider for example the first order auto-regression $\eta_t - \alpha\eta_{t-1} = e_t$ with $|\alpha| < 1$. Then:

- Multiply by η_{t-1} and take expectations:

$$E\{\eta_t\eta_{t-1} - \alpha\eta_{t-1}^2\} = E\{\eta_{t-1}e_t\} = 0 \Rightarrow R(1) - aR(0) = 0 \tag{2.2.12}$$

- Next multiply by η_t and take expectations:

$$E\{\eta_t^2 - \alpha\eta_t\eta_{t-1}\} = E\{\eta_t e_t\} \tag{2.2.13}$$

- Multiply through by e_t and take expectations:

$$E\{\eta_t e_t - \alpha\eta_t et\} = E\{e_t^2\} = 1 \Rightarrow E\{\eta_t e_t\} = 1 \tag{2.2.14}$$

- From equation 2.2.13, equation 2.2.12 and equation 2.2.14:

$$R(0) - \alpha^2 R(1) = 1 \Rightarrow R(0) = \frac{1}{1-\alpha^2} \; and \; R(1) = \frac{\alpha}{1-\alpha^2} \tag{2.2.15}$$

- Finally multiply through by η_{t-k} $(k = 1, 2, \ldots)$ and take expectations:

$$E\{\eta_t\eta_{t-k} - \alpha\eta_{t-k}\} = E\{e_t\eta_{t-k}\} = 0 \Rightarrow R(k) = aR(k-1) \tag{2.2.16}$$

Combining 2.2.15 and 2.2.16 gives:

$$R(k) = \frac{\alpha^k}{1-\alpha^2} \tag{2.2.17}$$

which gives the covariance function in closed form.

This procedure of multiplying by past samples and taking expectations works in general. The result is summarised in the following theorem [DV85] (Yule-Walker equations):

Theorem 2.2.4. *Let* $\{R(0), R(1), R(2), \ldots\}$ *be the covariance function of the stable ARMA model* $A(z^{-1})\eta_t = B(z^{-1})e_t$, *where* $A(z^{-1}) = 1 + a_1 z^{-1} + \ldots + a_n z^{-n}$ *and* $B(z^{-1}) = b_0 + b_1 z^{-1} + \ldots + b_m z^{-m}$ *and suppose that* $\{g_0, g_1, g_2, \ldots\}$ *is the unit-pulse response of* $A(z^{-1})^{-1}B(z^{-1})$. *Then:*

$$\begin{aligned} \sum_{i=0}^{m} a_i R(\ell - 1) &= \sum_{i=\max(0,\ell)}^{n} b_i g_{i-\ell} \quad \ell \leq n \\ &= 0 \qquad\qquad\qquad \ell \leq n \end{aligned}$$

Proof. See [DV85]. □

Note that the first $m + 1$ equations for $\ell = 0, 1, 2, \ldots, m$ involve $2m + 1$ unknowns $R(-m), \ldots, R(0), \ldots, R(m)$ which may be solved by using the relations $R(-i) = R(i)$. Using these values as initial data, the higher order $R(i)$'s can be determined recursively using the remaining Yule-Walker equations.

A second method for determining the covariance function is via contour integration. Recall that given η_t satisfying $A(z^{-1})\eta_t = B(z^{-1})e_t$ the spectral density is given by equation 2.2.11. Also, $\Psi(\omega) = \Phi_{\eta\eta}(e^{j\omega})$ is the inverse Fourier transform of $\{R(0), R(\pm 1), R(\pm 2), \ldots\}$. Hence the $R(i)$'s can be obtained as the Fourier coefficients of $\Psi(\omega)$, i.e.

$$\begin{aligned} R(k) &= \frac{1}{2\pi} \int_{-\pi}^{\pi} \Phi_{\eta\eta}(e^{j\omega}) e^{jk\omega} d\omega \quad (k = 0, \pm 1, \pm 2, \ldots)) \\ &= \frac{1}{2\pi j} \int_{-\pi}^{\pi} \Phi_{\eta\eta}(e^{j\omega}) (e^{j\omega})^{k-1} d(e^{j\omega}) \\ &= \frac{1}{2\pi j} \oint_{\partial D} \Phi_{\eta\eta}(z) z^{k-1} dz \end{aligned}$$

This is a contour integral in the complex plane around the unit circle ∂D which may be calculated using Cauchy's residue theorem. Thus, assuming that $z \to \Phi_{\eta\eta}(z)z^{k-1}$ has only simple poles inside ∂D (write them $\{z_1, z_2, \ldots, z_p\}$), then:

$$R(k) = \sum_{i=1}^{p} Res(\Phi_{\eta\eta}(z)z^{k-1}, z_i) = \sum_{i=1}^{p} \left\{ \lim_{z \to z_i} (z - z_i)\Phi_{\eta\eta}(z)z^{k-1} \right\}$$

Means and covariance functions of scalar random variables can also be defined for vector random variables. These can be alternatively thought of finite truncations of (infinite) stochastic processes. For a vector random variable:

$$\underline{X} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}$$

we can define its mean:

$$\mu = E(\underline{X}) = \begin{bmatrix} E(x_1) \\ E(x_2) \\ \vdots \\ E(x_n) \end{bmatrix} := \begin{bmatrix} \mu_1 \\ \mu_2 \\ \vdots \\ \mu_n \end{bmatrix}$$

and its covariance matrix:

$$R = R_{xx} = E\{(\underline{X} - E(\underline{X}))(\underline{X} - E(X))'\} = E(XX') - \mu\mu'$$

This can be written in full:

$$R = \begin{pmatrix} R_{11} & R_{12} & \cdots & R_{1n} \\ R_{21} & R_{22} & \dots & R_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ R_{n1} & R_{n2} & \cdots & R_{nn} \end{pmatrix}$$

where $R_{ij} = E(x_i x_j) - \mu_i \mu_j$. The symmetry of covariances $R_{ij} = R_{ji}$ implies that R is symmetric and positive semi-definite [Kij02].

A case of special interest is when X is normally distributed. In this case the joint density function $f(x)$ is given by:

$$f(x) = \frac{1}{(2\pi)^{\frac{n}{2}}\sqrt{det(R)}}(e^{-\frac{(x-\mu)'R^{-1}(x-\mu)}{2}})$$

provided that R is positive definite, where $x = (x_1, x_2, \dots, x_n)'$. Note that the marginal distributions are normal and $x_t \sim N(\mu_t, R_{tt})$. If the covariance matrix is not positive definite, the random variables are linearly dependent in a stochastic sense, and some special treatment is required.

Definition 2.2.2. A symmetric matrix A is *positive definite* if, for any non-zero vector c, $c'Ac > 0$. A is called *positive semi-definite* if, for any vector c, $c'Ac \geq 0$.

Stochastic state-space models

The standard state-space stochastic model we will be using is of the form given in equation 2.2.10:

$$\begin{aligned} x(t+1) &= Ax(t) + Be(t) \\ y(t) &= Cx(t) + De(t) \end{aligned} \tag{2.2.18}$$

As usual, x_k denotes the state vector (at time-index k), y_k is the system output vector and e_k the system input vector; we assume that the e_k's are uncorrelated, zero-mean, and that $Cov(e_k) = E(e_k e_k') = I$ for all k. We can give two different interpretations of this model: (i) The time set is $\mathbf{Z}_+$ (non-negative zeros only); in this case we need to specify the (joint) statistics of the initial state vector x_0 and e_0 to properly define the processes x_k and y_k; (ii) The time set is $\mathbf{Z}$ (all integers); in this case x_k and y_k are the outputs of the ARMA models $x_k = (zI - A)^{-1}Be_k$ and $y_k = [C(zI - A)^{-1}B + D]e_k$ in the sense discussed earlier.

Theorem 2.2.5. *[DV85] Suppose that in 2.2.10 the time index is $\mathbf{Z}_+$, the initial state x_0 has zero mean and covariance P_0, and that x_0 is uncorrelated with e_k for all k. Then $P(k) := Cov(x_k)$ satisfies:*

$$P(k+1) = AP(k)A' + BB', \quad P(0) = P_0 \tag{2.2.19}$$

and

$$\begin{aligned} Cov(y_k, y_{k-j}) &= CP(k)C' + DD', \quad j = 0 \\ &= CA^j P(k-j)C' + CA^{j-1}BD', \quad j = 1, 2, \ldots, k \end{aligned}$$

If A is stable then $P(k) \to P$ as $k \to \infty$ where P is the unique solution of the (discrete) Lyapunov matrix equation:

$$P = APA' + BB' \tag{2.2.20}$$

Furthermore if $P_0 = P$ *then* $P(k) = P, \forall k \geq 0$. *Next suppose that the time index is* $\mathbf{Z}$ *and that* A *is stable. In this case* $\{x_k\}$ *and* $\{y_k\}$ *are wide-sense stationary processes,* $E(x_k) = 0, Cov(x_k) = P$ *and:*

$$\begin{aligned} Cov(y_k, y_{k-j}) &= CPC' + DD', \quad j = 0 \\ &= CA^jPC' + CA^{j-1}BD', \quad j > 0 \end{aligned}$$

Here P *is again the (unique) solution of the Lyapunov equation 2.2.20.*

The proof can be found in [DV85].

Linear least-squares estimation

Postulate a model of the form:

$$y = X\theta + e$$

where:

$$y = \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{pmatrix}, \quad e = \begin{pmatrix} e_1 \\ e_2 \\ \vdots \\ e_n \end{pmatrix}, \quad \theta = \begin{pmatrix} \theta_1 \\ \theta_2 \\ \vdots \\ \theta_q \end{pmatrix} \quad \text{and} \quad X \in \mathcal{R}^{n \times q}$$

Here y and X are known, θ is the vector of parameters to be estimated (unknown) and e is a noise/disturbance vector (unknown).

Example: Consider a linear system which consists of a simple unknown gain θ and whose output is corrupted by a noise sequence e_t (see Figure 2.14). In this case model equations may be written as:

$$\begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{pmatrix} = \begin{pmatrix} u_1 \\ u_2 \\ \vdots \\ u_n \end{pmatrix} \theta + \begin{pmatrix} e_1 \\ e_2 \\ \vdots \\ e_n \end{pmatrix}$$

or as $y = X\theta + e$.

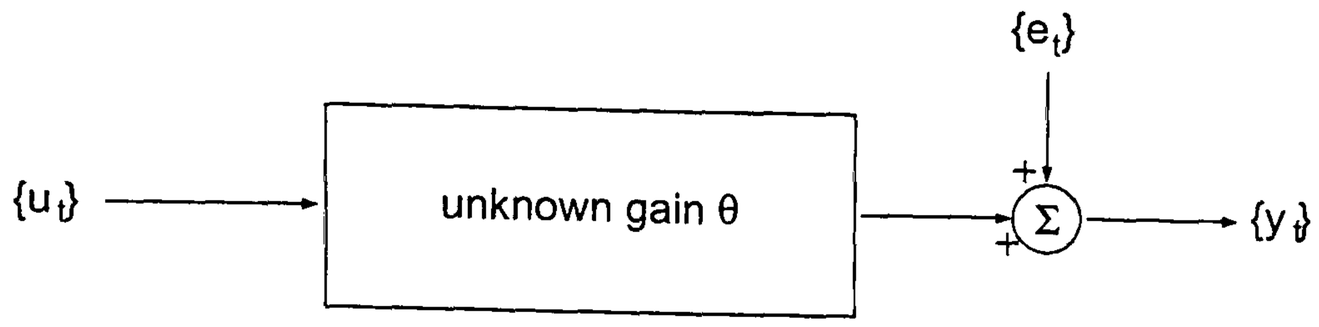

Figure 2.14: Estimation of unknown gain

Least squares estimate

In least squares estimation we choose θ to minimise the sum of squares of the "residuals" (errors), i.e., we minimise

$$J(\theta) = \sum_{i=1}^{n} (y_i - x_i^T \theta)^2 = (y - X\theta)^T (y - X\theta)$$

Here x_i^T is the i-th row of matrix X ("regression vectors"). To find the minimising θ, first expand $J(\theta)$:

$$J(\theta) = (y^T - \theta^T X^T)(y - X\theta) = y^T y - 2\theta^T X^T y + \theta^T X^T X\theta$$

Setting the derivative to zero:

$$\frac{\partial J(\theta)}{\partial \theta} = -2X^T y + 2X^T X\theta = 0 \Rightarrow X^T X\hat{\theta} = X^T y \quad \text{(Normal equations)}$$

Note that $X^T X$ is a square $(q \times q)$ matrix and therefore the normal equations can be solved uniquely if $X^T X$ is non-singular, or equivalently if X is of full column rank. In this case, therefore, the normal equations have the unique solution:

$$\hat{\theta} = (X^T X)^{-1} X^T y$$

Statistical properties least squares

Now assume that there is a "true" parameter θ, treat e as a vector of random variables and examine the statistical properties of $\hat{\theta}$. Assume that:

1. $E(e) = 0$.

2. $E(ee^T) = \sigma^2 I$

i.e., the e_i's are zero mean, uncorrelated with common variance σ^2.

Note that since $y = X\theta + e$, the y_i's are random variables, and hence any linear combination of these, $b^T y$ say, is also a random variable. Thus $\hat{\theta} = (X^T X)^{-1} X^T y$ is a vector of random variables and as such it will have a probability distribution. (Consider this as follows: Perform a sequence of random experiments, each time with a different realisation of the random vector e (drawn from the same distribution) and record in a histogram the percentage (successes over trials) that the estimate falls an interval of the histogram. In the limit (as the number of experiments increases and the histogram partitioning becomes finer and finer) we will have obtained the statistical distribution of $\hat{\theta}$).

The question now arises is when $\hat{\theta}$ is a "good" estimate. It is natural to say that $\hat{\theta}$ is "good" if the following two conditions are satisfied:

- The statistical mean of $\hat{\theta}$ coincides with the true parameter θ, i.e. if the estimates' average over noise realisations e (in the limit) is equal to θ.
- The variance of $\hat{\theta}$ is small, i.e., the statistical spread of the estimates (for different noise realisations) around the mean is small.

We start with the unbiasedness property:

Theorem 2.2.6. *$\hat{\theta}$ is unbiased.*

Proof. See [DV85]. □

Next we calculate the covariance of $\hat{\theta}$:

Theorem 2.2.7. *$Cov(\hat{\theta}) = \sigma^2 (X^T X)^{-1}$.*

Proof. See [DV85]. □

Theorem 2.2.8. *$\hat{\theta}$ is BLUE ('Best Linear Unbiased Estimate'), i.e., given any estimate β of the form $\beta = By$ (linear) such that $E(By) = \theta$ for all θ, then $Cov(\hat{\theta}) \leq Cov(\beta)$.*

Proof. See [DV85]. □

Estimation of noise variance

Consider the model

$$y = X\theta + e$$

in which e is a $n-vector$ of zero mean, uncorrelated random variables with common variance σ^2 and θ is a q-vector. We still assume that X is deterministic and that its columns are linearly independent. The parameter vector θ is constant (but unknown). Assume also that the noise variance σ^2 is unknown. A natural estimate of the variance σ^2 is:

$$\hat{\sigma}^2 = \frac{1}{n}\sum_{i=1}^{n} e_i^2 = \frac{1}{n}(y - X\theta)^T(y - X\theta)$$

Since θ is unknown, however, we may try $\hat{\theta}$ instead. It turns out that the estimate:

$$\hat{\sigma}^2 = \frac{1}{n}(y - X\hat{\theta})^T(y - X\theta)$$

(which is the "maximum-likelihood estimate of σ^2) is biased. An unbiased estimate of σ^2 is given in the next theorem. Before presenting this theorem some background material on the trace of a matrix is included.

The trace of a matrix: For a square, $(n \times n)$ say, matrix S, its trace is defined as the sum of its diagonal elements, i.e.,

$$\text{trace}(S) = \sum_{i=1}^{n} S_{ii}$$

Trace$(\cdot)$ is a linear operator: $\text{trace}(A + B) = \text{trace}(A) + \text{trace}(B)$, and $\text{trace}(aA) = a\text{trace}(A)$ for every scalar a. Another useful property is that $\text{trace}(AB) = \text{trace}(BA)$ whenever the products AB and BA are both defined. Check this: If $A \in \mathcal{R}^{m\times n}$ and $B \in \mathcal{R}^{n\times m}$ (so that both products are defined), then:

$$(AB)_{ii} = \sum_k A_{ik}B_{ki}$$

and hence

$$\text{trace}(AB) = \sum_i (AB)_{ii} = \sum_i \sum_j A_{ik}B_{ki} = \sum_k \sum_i B_{ki}A_{ik} = \sum_k (BA)_{kk} = \text{trace}(BA)$$

Another useful property of trace (not used here) is that

$$\text{trace}(S) = \sum_{i=1}^{n} \lambda_i(S)$$

where $\lambda_i(S)$ are the eigenvalues of S.

Theorem 2.2.9.

$$\hat{\sigma}^2 = \frac{1}{n-q}(y - X\hat{\theta})^T(y - X\hat{\theta})$$

is an unbiased estimate of σ^2.

Proof. See [HJ90] □

and hence $\hat{\sigma}^2$ is an unbiased estimate of σ^2.

Least squares estimation for dynamic systems

Consider the ARMA model

$$A(z^{-1})y_k = B(z^{-1})u_k + e_k$$

in which

$$A(z^{-1}) = 1 + a_1 z^{-1} + a_2 z^{-2} + \ldots + a_n z^{-n}$$

and

$$B(z^{-1}) = b_0 + b_1 z^{-1} + b_2 z^{-2} + \ldots + b_m z^{-m}$$

We require to estimate the parameters:

$$\theta = [a_1\ a_2\ \ldots\ a_n \mid b_0\ \ldots\ b_m]^T$$

from input-output measurements (u_i, y_i), $i = 1, 2, \ldots, N$. The model equations may be written as:

$$y_k = [-y_{k-1}\ -y_{k-2}\ \ldots\ -y_{k-n} \mid u_k\ \ldots\ u_{k-m}]\theta + e_k$$

for $k = 1, 2, \ldots, N$, or, in matrix form:

$$\begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_N \end{pmatrix} = \left(\begin{array}{cccc|ccc} -y_0 & -y_{-1} & \cdots & -y_{1-n} & u_1 & \cdots & u_{1-m} \\ -y_1 & -y_0 & \cdots & -y_{2-n} & u_2 & \cdots & u_{2-m} \\ \vdots & \vdots & & \vdots & \vdots & & \vdots \\ -y_{N-1} & -y_{N-2} & \cdots & -y_{N-n} & u_N & \cdots & u_{N-m} \end{array}\right) \theta + \begin{pmatrix} e_1 \\ e_2 \\ \vdots \\ e_N \end{pmatrix}$$

This is now in the form $y = X\theta + e$ and the parameter vector θ may be estimated by least squares.

Note 1: The 'regression matrix' X is partitioned as $X = [X_1 \mid X_2]$, in which X_1 and X_2 have a special ("Toeplitz") structure, i.e., entries along main diagonals are identical.

Note 2: The upper triangular parts of T_1 and T_2 contain data y_i and u_i, $i \leq 0$ which represent initial conditions. If these data are unavailable we use 0's to fill these positions.

Note 3: In this case X is a random matrix (since y is random and correlated with e), so standard least square (LS) theory does not apply. In particular:

- For any finite n, $\hat{\theta}$ is biased.
- $\hat{\theta}$ is asymptotically unbiased (i.e. $\lim_{N\to\infty} E(\hat{\theta}_N) = \theta$ for all θ if the e_l's are uncorrelated and the u_k's are "persistently exciting" (see next).
- $\hat{\theta}$ is biased even asymptotically if the e_k's are not white.

Asymptotic convergence in the case of white/correlated noise is examined in the following examples:

Example 1: (white noise)

Consider a second-order autoregressive model for simplicity:

$$y_k = \begin{pmatrix} y_{k-1} & y_{k-2} \end{pmatrix} \begin{pmatrix} -a_1 \\ -a_2 \end{pmatrix} + e_k \qquad k = 1, 2, \ldots, n$$

In this case,

$$\begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{pmatrix} = \begin{pmatrix} y_0 & y_{-1} \\ y_1 & y_0 \\ \vdots & \vdots \\ y_{n-1} & y_{n-2} \end{pmatrix} \begin{pmatrix} -a_1 \\ -a_2 \end{pmatrix} + \begin{pmatrix} e_1 \\ e_2 \\ \vdots \\ e_n \end{pmatrix}$$

and hence the least squares estimate is given by:

$$\hat{\theta} = (X^T X)^{-1} X^T y = (X^T X)^{-1} X^T (X\theta + e) = (\theta + X^T X)^{-1} X^T e$$

Now,

$$X^T X = \begin{pmatrix} y_0 & y_1 & \cdots & y_{n-1} \\ y_{-1} & y_0 & \cdots & y_{n-2} \end{pmatrix} \begin{pmatrix} y_0 & y_{-1} \\ y_1 & y_0 \\ \vdots & \vdots \\ y_{n-1} & y_{n-2} \end{pmatrix} = \begin{pmatrix} \sum_{k=1}^n y_{k-1}^2 & \sum_{k=1}^n y_{k-1}y_{k-2} \\ \sum_{k=1}^n y_{k-1}y_{k-2} & \sum_{k=1}^n y_{k-1}^2 \end{pmatrix}$$

or

$$\frac{1}{n}X^T X = \begin{pmatrix} \frac{1}{n}\sum_{k=1}^n y_{k-1}^2 & \frac{1}{n}\sum_{k=1}^n y_{k-1}y_{k-2} \\ \frac{1}{n}\sum_{k=1}^n y_{k-1}y_{k-2} & \frac{1}{n}\sum_{k=1}^n y_{k-1}^2 \end{pmatrix}$$

Under ergodicity assumptions, the "sample covariances" converge to the "true covariances" and hence:

$$\lim_{n\to\infty}\left\{\frac{1}{n}X^T X\right\} = \begin{pmatrix} R_{yy}(0) & R_{yy}(1) \\ R_{yy}(1) & R_{yy}(0) \end{pmatrix}$$

Also,

$$\frac{1}{n}X^T e = \frac{1}{n}\begin{pmatrix} y_0 & y_1 & \cdots & y_{n-1} \\ y_{-1} & y_0 & \cdots & y_{n-2} \end{pmatrix} \begin{pmatrix} e_1 \\ e_2 \\ \vdots \\ e_n \end{pmatrix} = \begin{pmatrix} \frac{1}{n}\sum_{k=1}^n y_{k-1}e_k \\ \frac{1}{n}\sum_{k=1}^n y_{k-2}e_k \end{pmatrix}$$

so that:

$$\lim_{n\to\infty}\left\{\frac{1}{n}X^T e\right\} = \begin{pmatrix} E(y_{k-1}e_k) \\ E(y_{k-2}e_k) \end{pmatrix} = 0$$

since clearly the pairs (y_{k-1}, e_k) and (y_{k-2}, e_k) are uncorrelated. Hence:

$$\hat{\theta}_n - \theta = \left\{\frac{1}{n}(X^T X)\right\}^{-1}\left\{\frac{1}{n}X^T e\right\} \longrightarrow 0$$

as $n \to \infty$, so that $\lim_{n\to\infty}\hat{\theta}_n = \theta$ and the LS estimates are asymptotically unbiased.

Example 2: (Correlated disturbances)

Consider now the model

$$y_k = ay_{k-1} + w_k, \qquad w_k = e_k + ce_{k-1}$$

in which $\{e_[\}$ is white ($E(e_k) = 0$, $\mathrm{Var}(e_k) = \sigma^2$). In this case,

$$\begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{pmatrix} = \begin{pmatrix} y_0 \\ y_1 \\ \vdots \\ y_{n-1} \end{pmatrix} a + \begin{pmatrix} w_1 \\ w_2 \\ \vdots \\ w_n \end{pmatrix}$$

which is in the form $y = Xa + w$. Hence,

$$X^T X = \sum_{k=1}^{n} y_{k-1}^2, \quad X^T y = \sum_{k=1}^{n} y_k y_{k-1}$$

and hence least-squares estimate is

$$\hat{a}_n = \frac{\sum_{k=1}^{n} y_k y_{k-1}}{\sum_{k=1}^{n} y_{k-1}^2}$$

and hence

$$\lim_{n\to\infty} (\hat{a}_n) = \lim_{n\to\infty} \frac{\frac{1}{n}\sum_{k=1}^{n} y_k y_{k-1}}{\frac{1}{n}\sum_{k=1}^{n} y_{k-1}^2} = \frac{R_{yy}(1)}{R_{yy}(0)}$$

under appropriate ergodicity assumptions. Now

$$y_k = a y_{k-1} + e_k + c_{k-1}$$

Multiplying by y_{k-1} and taking expectations:

$$E\{y_k y_{k-1}\} = aE\{y_{k-1}^2\} + E\{e_k y_{k-1}\} + cE\{e_{k-1} y_{k-1}\} \Rightarrow R_{yy}(1) = aR_{yy}(0) + cE(e_k y_k) \tag{2.2.21}$$

Multiplying by e_k and taking expectations:

$$E\{y_k e_k\} = aE\{y_{k-1} e_k\} + E\{e_k^2\} + cE\{e_{k-1} e_k\} \Rightarrow E(y_k e_k) = \sigma^2 \tag{2.2.22}$$

Substituting (2.2.22) in (2.2.21):

$$R_{yy}(1) = aR_{yy}(0) + c\sigma^2 \Rightarrow \lim_{n\to\infty} \hat{a}_n = a + \frac{c\sigma^2}{R_{yy}(0)}$$

and hence there is an asymptotic bias:

$$\text{asymptotic bias} = \frac{c\sigma^2}{R_{yy}(0)}$$

Note: $R_{yy}(0)$ may be calculated explicitly as:

$$R_{yy}(0) = \frac{\sigma^2(c^2 + 2ac + 1)}{1 - a^2}$$

Recursive least-squares

We consider a modification of the least-squares algorithm which updates the old estimates as new data come in. Consider the standard model:

$$\begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{pmatrix} = \begin{pmatrix} x_1^T \\ x_2^T \\ \vdots \\ x_n^T \end{pmatrix} \theta + \begin{pmatrix} e_1 \\ e_2 \\ \vdots \\ e_n \end{pmatrix}$$

which we now write as $Y_n = X_n\theta + E_n$ to keep track of dimensions. The least-squares estimate of θ at time $t = n$ is given by:

$$\hat{\theta}_n = (X_n^T X_n)^{-1} X_n^T Y_n$$

Define $P_n := (X_n^T X_n)^{-1}$; then

$$\hat{\theta}_n = P_n \begin{pmatrix} x_1 & \dots & x_n \end{pmatrix} \begin{pmatrix} y_1 \\ \vdots \\ y_n \end{pmatrix} = P_n \sum_{i=1}^{n} x_i y_i = P_n \left(\sum_{i=1}^{n-1} x_i y_i + x_n y_n \right)$$

Now

$$\hat{\theta}_{n-1} = P_{n-1} \sum_{i=1}^{n-1} x_i y_i \Rightarrow \sum_{i=1}^{n-1} x_i y_i = P_{n-1}^{-1} \hat{\theta}_{n-1}$$

Hence

$$\hat{\theta}_n = P_n \left\{ P_{n-1}^{-1} \hat{\theta}_{n-1} + x_n y_n \right\}$$

But,

$$P_n^{-1} = X_n X_n^T = X_{n-1}^T X_{n-1} + x_n x_n^T = P_{n-1}^{-1} + x_n x_n^T$$

Substituting we get

$$\hat{\theta}_n = P_n \{ (P_n^{-1} - x_n x_n^T) \hat{\theta}_{n-1} + x_n y_n \}$$

or

$$\hat{\theta}_n = \hat{\theta}_{n-1} - P_n x_n x_n^T \hat{\theta}_{n-1} + P_n x_n y_n$$

and thus

$$\hat{\theta}_n = \hat{\theta}_{n-1} + P_n x_n (y_n - x_n^T \hat{\theta}_{n-1})$$

To find a recursive update of P_n we have to use the "matrix inversion lemma":

Lemma 2.2.10. *For any three matrices A, B and C of compatible dimensions such that A and $A + BC$ are invertible,*

$$(A + BC)^{-1} = A^{-1} - A^{-1}B(I + CA^{-1}B)^{-1}CA^{-1}$$

Proof. See [HJ90]. □

The updating formulae of recursive least squares can be summarised as:

$$\begin{aligned} \hat{\theta}_n &= \hat{\theta}_{n-1} + P_n x_n (y_n - x_n^T \hat{\theta}_{n-1}) \\ P_n &= P_{n-1} - \frac{P_{n-1} x_n x_n^T P_{n-1}}{1 + x_n^T P_{n-1} x_n} \end{aligned}$$

Note 1: Computationally demanding matrix inversion is now completely avoided. Also, there is no need to store the old data; at time $t = n$ we receive new information from the data (x_n, y_n) and we update the old estimates $(P_{n-1}, \hat{\theta}_{n-1})$ to produce the new ones $(P_n, \hat{\theta}_n)$.

Note 2: The updating formula for $\hat{\theta}_n$ is intuitively appealing: The "predicted" value of y_n given information up to time $t = n - 1$ is

$$\hat{y}_{n|n-1} = x_n^T \hat{\theta}_{n-1}$$

When new information arrives at time $t = n$ in the form of a new measurement (y_n), this is compared with the predicted value to generate the "prediction-error":

$$e_{n|n-1} = y_n - \hat{y}_{n|n-1} = y_n - x_n^T \hat{\theta}_{n-1}$$

The new estimate $\hat{\theta}_n$ is then a correction of the previous estimate $\hat{\theta}_n$ by an amount proportional to the prediction error.

Note 3: Modified recursive least-squares algorithm: A slight disadvantage of the least-squares recursion is that (in its present form) it cannot be started at time $t = 1$, since the matrix P_n is only defined when $X_n^T X_n$ is non-singular, or equivalently when X_n has full column rank. A necessary condition for this is that $n \geq q$, i.e., that we have at least as many measurements as the number of parameters to be estimated. A possible solution is to wait for at least $n_0 \geq q$ measurements until

$X_{n_0}^T X_{n_0}$ is invertible, calculate P_{n_0} and θ_{n_0} by matrix inversion (i.e., as in batch least-squares), and proceed from then on recursively. An alternative approach is to start the recursions at time $t = 1$ by choosing reasonable initial conditions $\hat{\theta}_0$ and P_0 (e.g. in the absence of any information we can take $\hat{\theta}_0 = 0$ and $P_0 = \frac{1}{\epsilon} I_q$, where ϵ is a small positive number). It can be shown that this initialisation corresponds to the minimisation of:

$$J_n(\theta) = \sum_{i=1}^{n} (y_i - x_i^T \theta)^2 + \frac{1}{2}(\theta - \hat{\theta}_0)^T P_0^{-1} (\theta - \hat{\theta}_0)$$

This performance index differs from the "true" least-squares cost by a fixed amount $\frac{1}{2}(\theta - \hat{\theta}_0)^T P_0^{-1}(\theta - \hat{\theta}_0)$. As n increases, the deviation from the true cost decreases in relative terms. For small n, $J_n(\theta)$ can be made approximately equal to the least-square cost by choosing P_0 sufficiently large, e.g. $P_0 = \frac{1}{\epsilon} I_q$ for a small positive ϵ, as suggested previously. Note that from a previous result the covariance of the estimated vector is $\sigma^2 (X_n^T X_n)^{-1} = \sigma^2 P_n$ (assuming that E_n is white with variance σ^2). Thus, in statistical terms a "large" P_0 reflects a high initial uncertainty around the initial estimates $\hat{\theta}_0$.

Chapter 3

Bullwhip effect in Supply Chain

3.1 Introduction

Bullwhip effect is a major concern for all parts of supply chain because customers' demand variation can lead to inefficient use of resources, high costs due to overstocking, poor customer services, and increase of inventories when they try to maintain a given service level. This phenomenon is better understood in supply chains with many intermediate stages when demand information flows within different parts is distorted significantly. The impact of information distortion is more apparent to upstream stages (e.g., manufacturers) as it can been seen in Figure 3.1. Since variability in orders placed by a supplier is significantly higher than variability in customer demand, upstream levels are forced to carry more safety stock than downstream levels. Therefore, upstream levels must use advance forecasting methods and establish closed relations in terms of cooperation with the downstream parts of supply chain in order to avoid the symptoms of demand volatility.

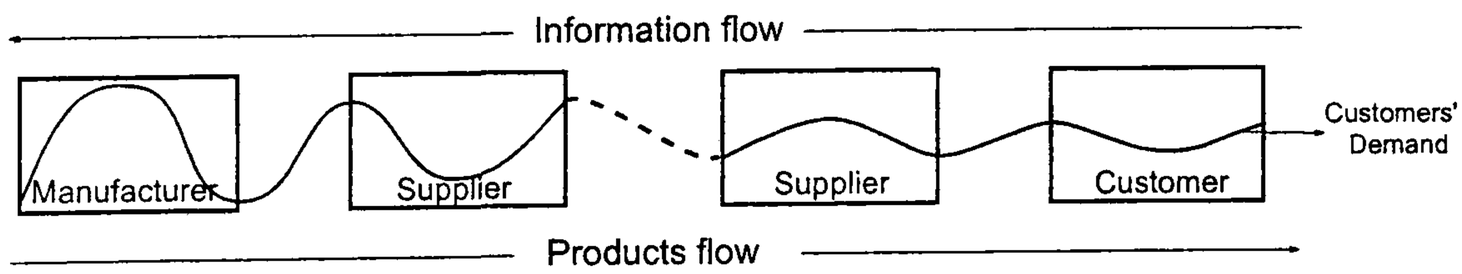

Figure 3.1: Demand variability through supply chain

Thus, it is important to find techniques and methods that will allow us to control

the increase of variability in supply chains. For this purpose we need first to identify and understand the main causes contributing to the bullwhip effect. Sterman [Ste89] shows how human behaviour such as misconceptions about inventory and demand information may cause the bullwhip effect. Lee et al [LPW97a] identify five major causes of this phenomenon shown in Figure 3.2.

1. Demand forecasting

2. Order batching

3. Price fluctuation

4. Rationing and shortage gaming

5. Lead times

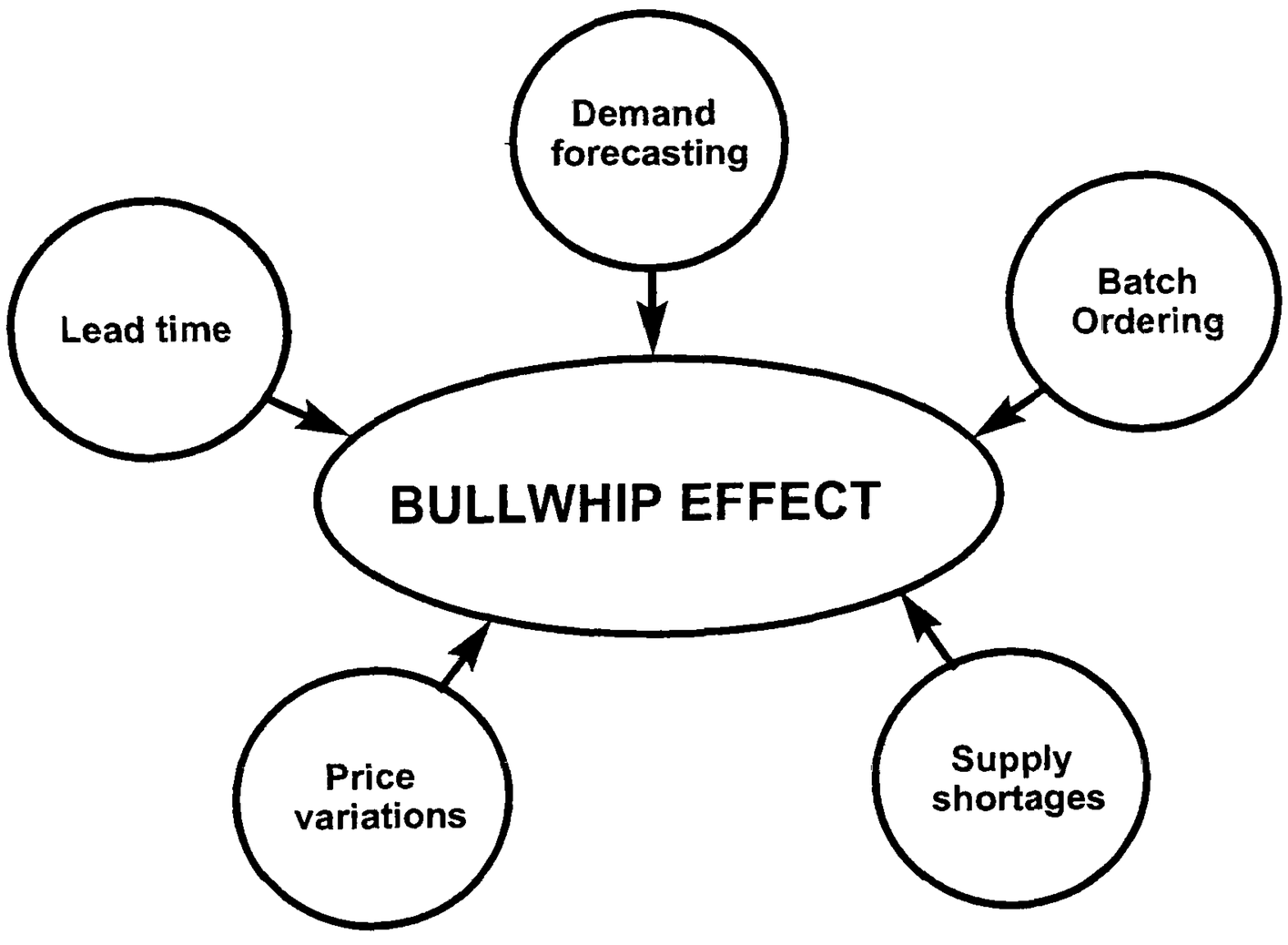

Figure 3.2: The five causes of bullwhip effect

In the next sections we discuss these main causes and we analyse the bullwhip effect using modelling and control methods in supply chains.

3.2 Causes of the bullwhip effect

3.2.1 The impact of demand forecasting

As it is discussed in previous chapters managing directors at each level use different inventory control policies in order to better forecast the demand from the downstream participants of supply chains. We consider a supply chain where every single node follows a simple inventory control policy where inventory level raises up to a target level (reorder point) in each period. The reorder point y_t is set equal to the average demand during lead time μ_L plus a multiple of the standard deviation σ_L of the demand during lead time:

$$y_t = \mu_L + \kappa\sigma_L$$

where κ is a parameter chosen to meet a desired service level.

In order to determine its target inventory level, each stage of the supply chain must forecast both the expected demand and the standard deviation of demand. This forecasting can be done using any of a number of forecasting techniques, for example, moving average or exponential smoothing. In the moving average forecast, the forecast average demand per period is simply the average of the demands observed over some fixed number of periods, say p periods. In the exponential smoothing forecast, the forecast average demand per period is a weighted average of all of the previous demand observations, where the weight placed on each observation decreases with the age of the observation. An important observation of all forecasting techniques is that as more data are observed, the more we modify the estimates of the mean and the standard deviation in customer demands.

Any forecasting technique can cause the bullwhip effect. To understand this, note that one property of most standard forecasting methods is that the forecast is updated each time a new demand is observed. Therefore, at the end of each period, each stage will observe the most recent demand, update this demand forecast based on this demand, and then use this updated forecast to update the target inventory level. It is this updating of the forecast and order-up-to point in each period that

results in increased variability in the orders placed by this stage. Finally, note that if a node follows a simple order-up-to inventory policy in which does not update the desired inventory level in each period, then would not see the bullwhip effect. In other words, if in every period each stage places an order to raise the on-hand inventory to the same fixed level, then the orders seen by the upstream level (e.g. manufacturer) would be exactly equal to the customer demand seen by a downstream stage. Therefore, the variability in the orders seen by the upstream levels would be exactly equal to the variability of the customer demand, and there would be no bullwhip effect.

Inaccuracy of forecasts is a continuing problem for most businesses. It seems no matter how sophisticated are forecasting techniques, the volatility of demand proves that the forecast will be wrong.

3.2.2 The impact of lead time

Whilst many forecasting errors are the result of inappropriate forecasting methodology the root cause of these problems is that forecast error increases as lead time increases. Lead times can add to the bullwhip effect by magnifying the increase in variability due to the demand forecasting. To understand this, note that lead times increase the target inventory level, i.e., the longer the lead time, the larger the inventory level required. In addition, if as discussed above, a downstream participants update their target inventory level in each period (using demand forecasting), then longer lead times will lead to larger changes in the target inventory level, and thus more variability in the orders placed by these participants. For example, if the demands seen by a retailer are independent and identically distributed (iid) from a normal distribution with mean $\mu.$ and variance σ^2, then an approximately optimal order-up-to inventory level in period t is:

$$y_t = L\mu_L + \kappa\sqrt{L}\sigma_L \tag{3.2.1}$$

where L is the lead time, μ_L is an estimate of μ, and σ_L is an estimate of σ. In this case, we clearly see that if, from time t to $t + 1$, our estimate of μ changes by Δ, then our order-up-to level will change by $L\Delta$, where $L \geq 1$. In other words, any changes in our estimates of the parameters of the demand process will be magnified by the lead time.

3.2.3 The impact of batch ordering

Another major cause of the bullwhip effect is the "hatching" of orders. The impact of batch ordering is quite simple to understand: if a downstream level e.g., a retailer uses batch ordering, then the manufacturer will observe a very larger order, followed by several periods of no orders, followed by another large order, and so on. Thus the manufacturer sees a distorted and highly variable pattern of orders.

Caplin [Cap85] considers the impact of batch ordering on the bullwhip effect. As we have seen in survey of literature, he considers a retailer who follows a continuous review *(s, S)* inventory policy in which the retailer continuously monitors the inventory level, and when the inventory level drops to s, places an order to raise the inventory level to S. Note that for an inventory policy of this form, when the retailer places an order, the size of that order, $Q = S - s$, is fixed and known. In this case, since the size of the order is fixed, the variability of the orders placed by the retailer is due only to the variability in the time between orders, i.e., the variability in the time it takes for the inventory level to fall from S to s. Caplin proves that, if the demands faced by the retailer are iid, then the variance of the orders placed by the retailer is greater than the variance of the customer demand observed by the retailer, and that the variance of the orders increases linearly in the size of the orders, i.e., the variability will increase linearly in Q.

3.2.4 The impact of supply shortages

A fourth cause of the bullwhip effect is associated with anticipated supply shortages. If a retailer anticipates that a particular item will be in short supply, he may place an

inflated, unusually large order with the upstream suppliers, because usually supplier will ration out the product among his customers based on the size of their orders. This rationing distorts the true demand pattern and gives the manufacturer a false impression of the market demand for the product. When the period of shortage is over, the retailer goes back to its standard orders, leading to all kinds of distortions and variations in demand estimations. An interesting discuss of the impact of this type of gaming on the variability in a supply chain can be found in [LPW97a].

3.2.5 The impact of price variations

A final cause of the bullwhip effect is the frequent price variations seen throughout a supply chain. For example, many retailers will offer products at a regular retail price with periodic price promotions and clearance sales. Clearly, when the price for an item changes, the customer demand for that item will also change. For example, during a promotion campaign the retailer will see higher than usual demand, while after the campaign, the retailer may observe unusually low demand. Therefore, periodic price promotions can cause distorted demand patterns and increased variability in demand (stocking up). This phenomenon is observed at other stages of the supply chain as well. For example, when a manufacturer offers a trade promotion, retailers may place unusually large orders and stockpile inventory, and may not order again for several periods. Again, this causes distorted demand patterns and increases the variability in demand.

3.3 Quantifying the bullwhip effect

In order to better understand and control the bullwhip effect, it would also be useful to quantify the bullwhip effect, i.e., quantify the increase in variability that occurs at every stage of the supply chain. This would be useful not only to demonstrate the magnitude of the increase in variability, but also to show the relationship between the demand process, the forecasting technique, the lead time and the increase in

variability.

To quantify the increase in variability for a simple supply chain, consider a two stage supply chain with a retailer who observes customer demand and places an order to the manufacturer. Suppose that the retailer faces a fixed lead time, so that an order placed by the retailer at the end of period t is received at the start of period $t + L$. Also, suppose the retailer follows a simple periodic review policy in which the retailer reviews inventory every period and places an order to bring its inventory level up to a target level. If we assume that the review period is one then the optimal order-up-to inventory level in period t as we have seen is given by equation 3.2.1.

If we assume that the retailer uses one of the simplest forecasting techniques, the moving average, then in each period the retailer estimates the mean demand as an average of the previous p observations of demand. The retailer also estimates the standard deviation of demand in a similar manner. If we denote D the customer demand in period i, then:

$$\mu_t = \frac{\sum_{i=t-p}^{t-1} D_i}{p}$$

and

$$\sigma_t^2 = \frac{\sum_{i=t-p}^{t-1} (D_i - \mu_t)^2}{p-1}$$

Note that the expressions above imply that in every period the retailer calculates a new mean and standard deviation based on the p most recent observations of demand. Then, since the estimates of the mean and standard deviation change every period, the target inventory level will also change in every period.

In this case, we can quantify the increase in variability by calculating the variability faced by the manufacturer and compare it to the variability faced by the retailer. If we denote *Var(D)* the variance of the customer demand seen by the retailer, and *Var(C)* the variance of the orders placed by that retailer to the manufacturer (relative to the variance of customer demand), then:

$$\frac{Var(C)}{Var(D)} \geq 1 + \frac{2L}{p} + \frac{2L^2}{p^2} \tag{3.3.1}$$

Note that equation 3.3.1 presumes that demands are identically and independently

distributed (iid).

It is easily verified, see [SLKSL03] that in a supply chain with centralised demand information where each stage of the supply chain has complete information on the actual customer demand:

$$\frac{Var(C^k)}{Var(D)} \geq 1 + \frac{2(\sum_{i=1}^{k} L_i)}{p} + \frac{2(\sum_{i=1}^{k} L_i)^2}{p^2}, \quad \forall k. \tag{3.3.2}$$

In the case of decentralised demand information where the retailer does not provide the upstream stages with any customer demand information:

$$\frac{Var(C^k)}{Var(D)} \geq \prod_{i=1}^{k}(1 + \frac{2L_i}{p} + \frac{2L_i^2}{p^2}), \quad \forall k. \tag{3.3.3}$$

where $Var(C^k)$ is the variance of the orders placed by stage k.

Note that equation 3.3.3 for the variance of the orders placed by the kth stage of the supply chain is similar to the equation 3.3.1 but now the variance increases in a multiplicative way at each stage of the supply chain. Also note that the variance of the orders becomes larger as we move up the supply chain, so that the orders placed by the manufacturer are more variable then the orders placed by the retailer.

3.4 Control the bullwhip effect

3.4.1 Supply chain model

We consider a simple series multi-stage supply chain as shown in Figure 3.3. There are n individual stages between generic Customer and Manufacturer and we denote as i the intermediate supplier index ($i \geq 1$). The supply chain depicted in Figure 3.3 also shows the flow of goods and information (orders) within the supply chain and is treated as a pull supply chain whereas production and distribution are demand driven and are based on actual customer demand. If we assume a simple scenario where no forecasts techniques are used by the suppliers the basic operation of the supply chain is follows: When a Customer (*node 1*) places an order at the Retailer level (*node i-1*), the order is assigned as backlog order (orders to be processed)

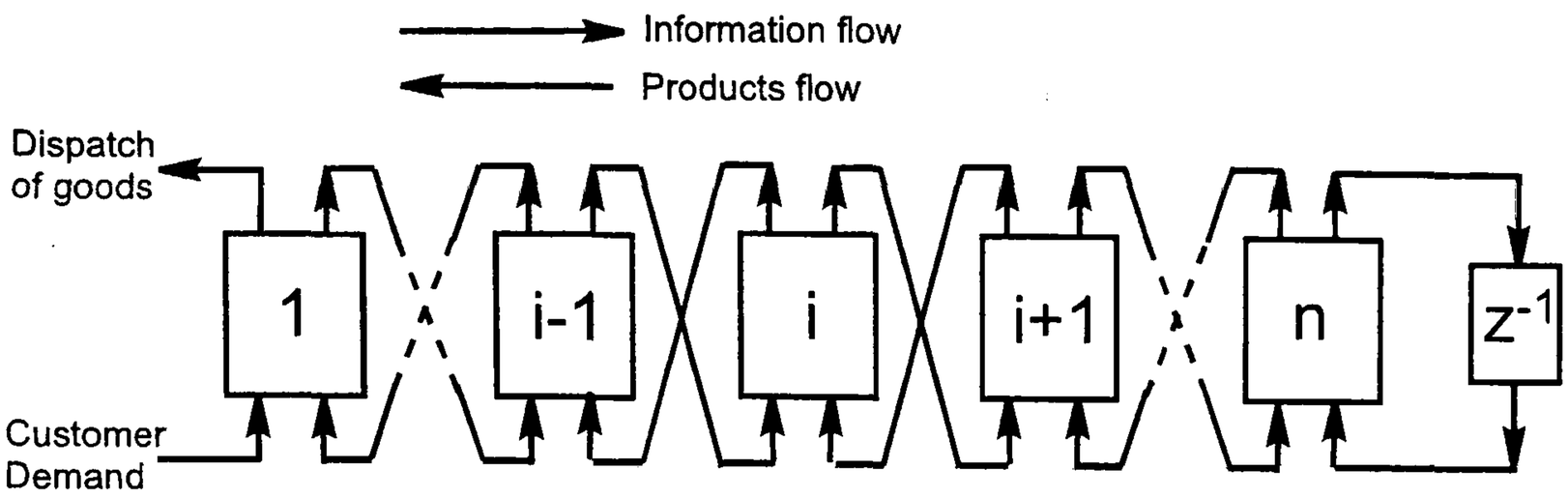

Figure 3.3: A series supply chain with n stages

of the retailer. If the inventory holds enough products retailer dispatches them to the customer and places the same amount of orders to the upstream level. If this is not the case, retailer dispatches all the products held in inventory and places a replenishment order to upstream supplier. When products arrive, they are shipped to the customer and all the backlog orders are cleared. This process is repeated in all stages of the supply chain until the orders reach the most upstream level (*node* n) e.g., Manufacturer. We assume that supply chain does not experience any delay in the flow of information but it does experience delay in the flow of products. This delay is known as lead time which is one of the causes of the bullwhip effect discussed in previous section.

Supply chain presumes that all the n-stages are making decisions locally and that there is no global supervisor. Thus, each stage is considered as decentralised system operating independently. As it can be inferred by Figure 3.3 the nodes have two kinds of inputs and outputs. Products and information flows, products when it sends or receives products and information when it receives or places orders.

Let $I_i(t)$ denote the inventory level of node i at time t. We let also $Y_{i,i-1}(t)$ indicate the amount of goods dispatched to downstream node $i-1$ by the upstream node i at time instant t. We also introduce a time delay L, which is the lead time needed for the goods to be delivered to the downstream node (i.e., the goods dispatched at time t are delivered at time $t+L$). However, due to the need for

examination and administrative processing, this new delivery is only available to a customer at $t + L + 1$. The model is based on [LWJ+04], from where additional details can be obtained.

Balancing the inventory $I_i(t)$ of node i at time step t gives:

$$I_i(t) = I_i(t-1) + Y_{i+1,i}(t-L) - Y_{i,i-1}(t) \tag{3.4.1}$$

where $I_i(t-1)$ is the inventory level at node i at time step $t-1$ and $Y_{i,i+1}(t-L)$ represents the products dispatched by the upstream node $i+1$ to node i, which is assumed to arrive with a delay of L time steps. Although inventory level is a key variable in supply chain operation, each node i can better monitor the changes in inventory level at time t by using inventory position, $IP_i(t)$, which is given by:

$$IP_i(t) = IP_i(t-1) + Y_{i+1,i}(t) - Y_{i,i-1}(t) \tag{3.4.2}$$

Another reason of using the inventory position $IP_i(t)$ instead of actual inventory $I_i(t)$ is mainly because its calculation in each step does not depend upon the lead time L. Since the minimisation of inventory fluctuations is a key issue for supply chain managers, inventory position can provide useful information about changes in inventory levels. These changes can be analysed easily by choosing the inventory position $IP_i(t)$ as a state variable as it will be discussed later.

As it has been mentioned before we consider the supply chain network as a decentralised control system - where there is no global moderator and decisions are taken locally at each node, e.g., corresponding to managers aiming to hold their stocks at certain levels or to meet expected future demand, following a series of rules which are known as inventory policies. Hence, the amount of orders placed at the upstream level must satisfy certain criteria such as minimising holding costs, avoiding shortages and maximising profits. The resulting decentralised structure of the system is one of the main contributory factors of the bullwhip effect - in a sense it is analogous to the 'string oscillations' observed in automated highway systems due to the lack of proper 'preview information'. Whereas co-operation between supply chain managers (e.g., by sharing information) would certainly help to alleviate the

bullwhip effect, unfortunately most enterprises regard these data as proprietary and are reluctant to share them. Techniques for forcing co-operation between supply chain participants are considered in the next chapter.

We denote by $O_{i,i+1}(t)$ the amount of orders placed by node i to node $i+1$, given by:

$$O_{i,i+1}(t) = k_i(SP_i - IP_i(t)) \tag{3.4.3}$$

where SP_i represents a target set-point (assumed constant) and k_i is the corresponding inventory replenishment gain factor.

Note that the size of the order placed corresponds to an inventory control policy. We consider that inventory managers on each stage are aware of allowing the downstream customer node to order as much as it wants, but managers do not guarantee that the demand order can be fulfilled.

$O_{i,i+1}(t)$ in equation 3.4.3 can be negative if inventory position is higher than the set point. In this case we allow the downstream node to revoke its order. Although we assume that there is no delay on information flow, an order at time t will only be processed at time $t+1$ in similar way that products delivered at time $t+L$ and are only available to a customer at $t+L+1$.

Thus, we need to introduce a new variable representing the amount of order to be processed at time $t+1$ by a node i. This variable is called standing order. Standing orders of note i at time step t, $O_i^\star(t)$, evolve according to the difference equation:

$$O_i^\star(t) = O_{i-1,i}(t) + O_i^\star(t-1) - Y_{i,i-1}(t) \tag{3.4.4}$$

We assume that an order can be accumulated to the next time step if it is not fulfilled, since each customer has only one supplier in our series supply chain. Therefore, the standing order for node i at time t is the sum of the order placed plus any unfulfilled order at time t.

Equation 3.4.4 presumes that if there is enough inventory to satisfy the standing order at $t+1$, all the orders will be delivered. Otherwise, the inventory will be cleared. Similarly, if the downstream node already has too much inventory, the upstream node

will just stop to deliver since return of goods is not taken into consideration. We can summarise above remarks as follows:

$$Y_{i,i-1}(t) = \begin{cases} 0, & O_i^\star(t-1) \leq 0 \\ O_i^\star(t-1), & 0 \leq O_i^\star(t-1) \leq I_i(t-1) \\ I_i(t-1), & 0 \leq I_i(t-1) \leq O_i^\star(t-1) \end{cases} \tag{3.4.5}$$

The ordering and delivery times have been assumed so far continuous although in practice they are discrete. Thus, the supply chain system can be viewed as a linear discrete system. We can represent above equations in z-transform and derive a discrete time model. Hence Inventory balance $I_i(z)$ is given by:

$$I_i(z) = \frac{z}{z-1}(z^{-L}Y_{i+1,1}(z) - Y_{i,i-1}(z)) \tag{3.4.6}$$

inventory position $IP_i(z)$:

$$IP_i(z) = \frac{z}{z-1}(Y_{i+1,i}(z) - Y_{i,i-1}(z)) \tag{3.4.7}$$

orders to be placed $O_{i,i+1}(z)$:

$$O_{i,i+1}(z) = k_i(SP_i(z) - IP_i(z)) \tag{3.4.8}$$

and standing orders $O_i^\star(z)$:

$$O_i^\star(z) = \frac{z}{z-1}(O_{i-1,i}(z) - Y_{i,i-1}(z)) \tag{3.4.9}$$

and:

$$Y_{i,i-1}(z) = \begin{cases} 0, & z^{-1}O_i^\star(z) \leq 0 \\ z^{-1}O_i^\star(z), & 0 \leq z^{-1}O_i^\star(z) \leq z^{-1}I_i(z) \\ z^{-1}I_i(z), & 0 \leq z^{-1}I_i(z) \leq z^{-1}O_i^\star(z) \end{cases} \tag{3.4.10}$$

Figure 3.4 shows the simplified block diagram of the node i derived by equation 3.4.6 - 3.4.10.

Although the above mathematical model seems simple it captures the basic dynamic behaviour of a series supply chain system. A real supply chain can many intermediate nodes and products. In a decentralised system, the inventory dynamics does not really depend on how many customers or suppliers the node has, since all

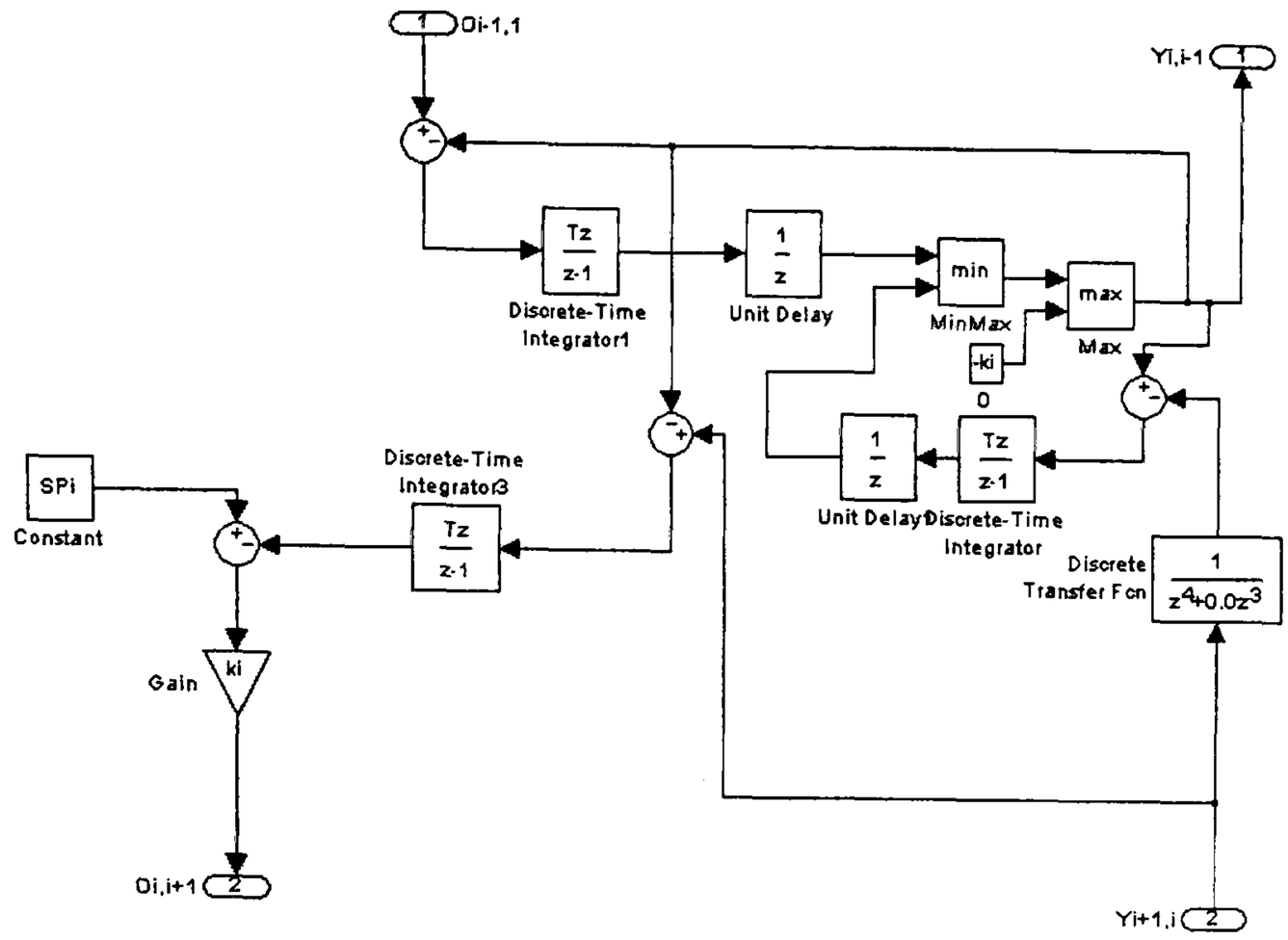

Figure 3.4: The block diagram of node i of the series supply chain

customer demands can be defined into an aggregate demand. Obviously, if every node has sufficient inventory and has the same lead time, the distribution of order would not affect the system dynamic behaviour. We can assume that different suppliers have different inventory levels with different lead times, and investigate the optimal order allocations. If the processing of order and delivery of different products do not interfere with each other, each product can be viewed as a separate supply chain. One may impose constraints that total inventory is limited by storage space and contrive an inventory control policy accordingly, although in a pure pull system, each stage does not hold any inventory and only places orders. Various complications can be introduced and analysed using our basic model. However, it is important to understand the basic dynamic behaviour, before such complications are introduced.

For the purposes of further analysis it is assumed that there is always enough stock at each node to meet the demand, so that $Y_{i,i-1}(t) = O_i^\star(t-1)$. This implies that the amount of goods dispatched to node $i-1$ from the upstream node i at time t is the amount of standing orders of node i at time $t-1$. This is essentially

a linearisation assumption also made in [LWJ+04] which simplifies the subsequent analysis. In addition, since the covariance analysis of the following section does not depend on SP_i, we may set $SP_i = 0$ for simplicity.

3.4.2 Stochastic state space for analysing the bullwhip effect under white noise customer demand profiles

We now consider the series supply chain model depicted in Figure 3.5.

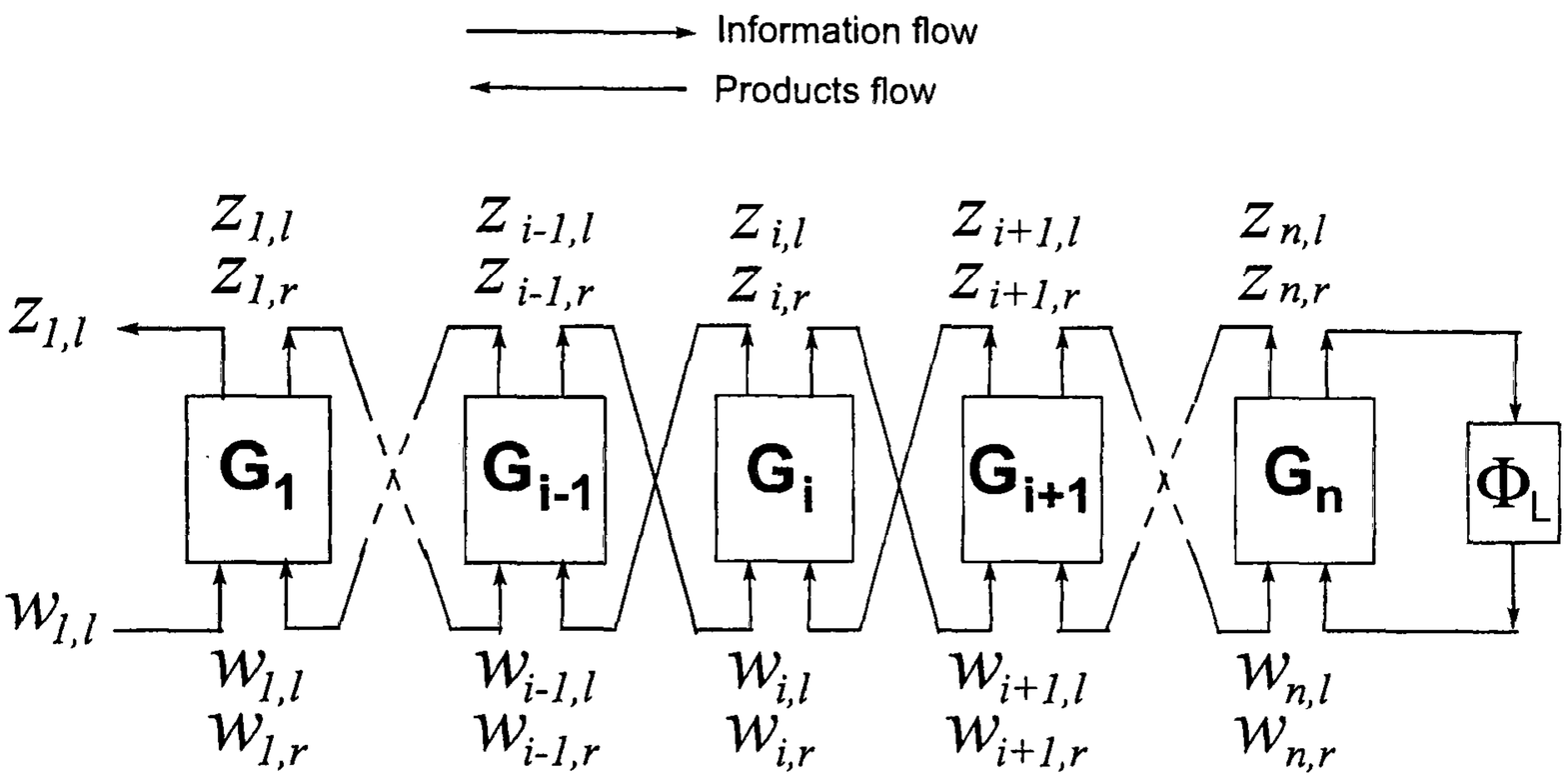

Figure 3.5: The supply chain with defined inputs and outputs in each node

Each stage G_i has two inputs $w_{i,l}$ and $w_{i,r}$ and two outputs $z_{i,l}$ and $z_{i,r}$ (left and right respectively). It can be inferred from the nature of figure's interconnections that $w_{i,l} = z_{i-1,r}$ and $w_{i,r} = z_{i+1,l}$. Index l represents the left part of each node while r represents the right part of inputs and outputs. The terminal node Φ representing the Manufacturer site is assumed to be a simple time delay. Thus the manufacturer always delivers the order he receives with a delay of one time step. If we assume that the input and output of the system are $w_{1,r}$ and $z_{1,l}$, respectively then we can

derive the model equations in form:

$$\dot{x}(t) = Ax(t) + Bu(t) \tag{3.4.11}$$

$$y(t) = Cx(t) + Du(t) \tag{3.4.12}$$

where $y(t) = z_{1,l}$ and $u(t) = w_{1,r}$. The model equations for each separate node can be expressed in state space form as:

$$x_i(t+1) = A_i x_i(t) + B_{i,l} w_{i,l}(t) + B_{i,r} w_{i,r}(t) = A_i x_i(t) + \begin{pmatrix} B_{i,l} & B_{i,r} \end{pmatrix} \begin{pmatrix} w_{i,l}(t) \\ w_{i,r}(t) \end{pmatrix}$$

and

$$\begin{pmatrix} z_{i,l}(t) \\ z_{i,r}(t) \end{pmatrix} = \begin{pmatrix} C_{i,l} \\ C_{i,r} \end{pmatrix} x_i(t) + \begin{pmatrix} D_{i,ll} & D_{i,lr} \\ D_{i,rl} & D_{i,rr} \end{pmatrix} \begin{pmatrix} w_{i,l}(t) \\ w_{i,r}(t) \end{pmatrix}$$

The above state space refers to node i. We can easily obtain the state space for each individual node if we simply substitute the node i with the corresponding node index.

The equivalent state-space model of the Manufacturer node is:

$$x_\phi(t+1) = A_\phi x_\phi(t) + B_\phi z_{i+1,r}(t)$$

and

$$w_{i+1,r}(t) = C_\Phi x_\phi(t)$$

Due to our previous assumption we have $A_\phi = D_\phi = 0$ and $B_\phi = C_\phi = 1$. The state space form of the node i given above can be written in more concrete form as:

$$x_i(t+1) = \begin{pmatrix} 1 & -1 \\ 0 & 0 \end{pmatrix} x_i(t) + \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} u_i(t) \tag{3.4.13}$$

and

$$y_i(t) = \begin{pmatrix} 0 & 1 \\ -k_i & k_i \end{pmatrix} x_i(t) + \begin{pmatrix} 0 & 0 \\ 0 & -k_i \end{pmatrix} u_i(t) \tag{3.4.14}$$

where $y_i(t)$ and $u_i(t)$ are the (two-dimensional) vector outputs and inputs of node i, respectively. The overall state space model for a series supply chain with three and four stages can be found in Appendix B.

As mentioned previously, the inventory level I_i and the amount of goods $Y_{i,i-1}$ dispatched by node i to its downstream stage are both important variables for decision making. By making these decisions, managers can manipulate and control the entire supply chain system. By choosing $IP_i(t-1)$ and $Y_{i,i-1}(t)$ as state space variables, all other variables of the node can be easily calculated.

The state space model given by equation 3.4.13 and 3.4.14 can be written as:

$$\begin{pmatrix} IP_i(t) \\ Y_{i,i-1}(t+1) \end{pmatrix} = \begin{pmatrix} 1 & -1 \\ 0 & 0 \end{pmatrix} \begin{pmatrix} IP_i(t-1) \\ Y_{i,i-1}(t) \end{pmatrix} + \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} O_{i-1,i}(t) \\ Y_{i+1,i}(t) \end{pmatrix}$$

and

$$\begin{pmatrix} Y_{i,i-1}(t) \\ O_{i,i+1}(t) \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ -k_i & k_i \end{pmatrix} \begin{pmatrix} IP_i(t-1) \\ Y_{i,i-1}(t) \end{pmatrix} + \begin{pmatrix} 0 & 0 \\ 0 & -k_i \end{pmatrix} \begin{pmatrix} O_{i-1,i}(t) \\ Y_{i+1,i}(t) \end{pmatrix} + \begin{pmatrix} 0 \\ k_i \end{pmatrix} SP_i(t)$$

i.e., we choose: $x_i(t) = (IP_i(t-1)Y_{i,i-1}(t))'$, $w_{i,l}(t) = O_{i-1,i}(t)$, $w_{i,r}(t) = Y_{i+1,i}(t)$, $z_{i,l}(t) = Y_{i,i-1}(t)$ and $z_{i,r}(t) = O_{i,i+1}(t)$

Note that the assumed inventory replenishment policy is continuous (rather than periodic). We also not consider variations in the set-point levels, i.e., $SP_i(t)$ are assumed to be constant. As these do not affect the covariance analysis performed in the next section, they can be set to zero.

3.4.3 Computation of model's covariance matrix

In this section we outline a method for calculating the covariance matrix of the state-vector $x(t)$ of the overall model developed in the previous section using symbolic computations. In our application, symbolic computations are essential, since we wish to obtain the solution as a function of the gain parameters $\{k_i\}$, which will allow further investigation of the bullwhip effect using our model. We first outline a general solution method based on Kronecker matrix products and vectorisation operations [HJ95]; subsequently, the special structure of the state-space model is exploited to derive a simple recursive solution procedure which can be applied to models of arbitrarily high complexity.

Consider the LTI discrete-time state-space model:

$$x(t+1) = Ax(t) + Be(t) \tag{3.4.15}$$

$$y(t) = Cx(t) + De(t) \tag{3.4.16}$$

where $\{e(t)\}$ denotes a white vector-noise sequence of unit intensity, representing customer demand, assumed to have been applied as input to the model since the infinite past. Then, assuming that A is asymptotically stable (all eigenvalues of A have modulus less than one), the (steady-state) covariance of the state-vector $x(t)$, $E(x(t)x'(t))$, is given by the (unique, positive semi-definite) solution of the discrete Lyapunov equation [DV85]:

$$P - APA' - BB' = 0 \tag{3.4.17}$$

Further, $E(yy') = CPC' + DD'$. In our case, A depends linearly on n parameters $k_1, k_2, \ldots, k_n$ which are assumed constant (but possibly unknown). Hence, the solution of 3.4.17 is the steady-state covariance of $x(t)$ for all t, for all combinations of $\{k_j\}$ for which A is asymptotically stable. It is shown next that this condition is satisfied if and only if the parameter vector $k = (k_1, k_2, \ldots, k_n)$ lies in the hypercube:

$$\mathcal{K}_n = (0,\ 2)^n := \{k \in \mathcal{R}^n\ :\ 0 < k_i < 2,\ i = 1, 2, \ldots, n\}$$

This agrees with a parallel result in [DDLT03].

Lemma 3.4.1. *Consider the $(m+1)-th$ node model 3.4.16 depending on m real gain parameters $k = \{k_1, k_2, \ldots, k_m\}$. Then the system is asymptotically stable if and only if $k \in \mathcal{K}_m$. In particular, if $A = A_{2m+1}$ denotes the 'A'-matrix of the state-space realisation of the system, then the eigenvalues of A are $\{1-k_1, 1-k_2, \ldots, 1-k_m, 0, \ldots, 0\}$, where the multiplicity of the zero eigenvalue is $m+1$.*

Proof. See Appendix A. □

Next, let $A \otimes B$ denote the Kronecker product of two matrices A and B; let also $\text{vec}(A)$ be the operation which stacks the elements of a matrix A in a column vector (sweeping along the rows of A). Applying the $\text{vec}(\cdot)$ operation to equation 3.4.17 and using the identity $\text{vec}(ABC) = (C' \otimes A)\text{vec}(B)$ (see [HJ90] gives:

$$(I_{n^2} - A \otimes A)\text{vec}(P) = \text{vec}(BB')$$

which may be solved as:

$$\text{vec}(P) = (I_{n^2} - A \otimes A)^{-1}\text{vec}(BB') \tag{3.4.18}$$

The following Lemma guarantees that the indicated inverse in equation 3.4.18 exists. The Lemma shows that matrix $I_{n^2} - A \otimes A$ is non-singular for all $k \in \mathcal{K}_n$. This is important as it ensures that the symbolic inverse of the matrix exists and can be expressed uniquely as a function of the k_i's. Of course, the solution of the equation is a valid covariance matrix of the state $x(t)$ only when $k \in \mathcal{K}_n$.

Lemma 3.4.2. *Matrix $I_{n^2} - A \otimes A$ is non-singular for all $k \in \mathcal{K}_n$. In fact, $I_{n^2} - A \otimes A$ is singular if and only if $(1-k_i)(1-k_j) = 1$ for any two indices i and j such that $i \leq i \leq m$ and $1 \leq j \leq m$, where $n = 2m+1$.*

Proof. See Appendix A. □

The covariance matrix obtained in equation 3.4.18 essentially involves the solution of a system of n^2 linear equations in the elements of P, which depend parametrically on the k_i's. Since the solution of the Lyapunov equation is symmetric, however, this

system of equations is redundant (with $n(n-1)/2$ equations being repeated). The solution can be simplified using the following procedure: For a symmetric matrix P let $\overline{\text{vec}}(P)$ denote $\text{vec}(P)$ with all the entries of P below the main diagonal eliminated. Clearly, if $P \in \mathcal{R}^{n\times n}$, then $\overline{\text{vec}}(P) \in \mathcal{R}^r$, where $r = n(n+1)/2$. Define $W \in \mathcal{R}^{n^2\times r}$ so that $\text{vec}(P) = W\overline{\text{vec}}(P)$, e.g. for $n = 2$,

$$W = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

Let also $\mathcal{S} \subset \{1, 2, \ldots, n\}$ be the subset of the $n(n-1)/2$ indices of $\text{vec}(P)$ which are eliminated when constructing $\overline{\text{vec}}(P)$. Then equation 3.4.18 may be written as:

$$V(I_{n^2} - A \otimes A)W\overline{\text{vec}}(P) = V\text{vec}(BB') \tag{3.4.19}$$

where $V \in \mathcal{R}^{r\times n^2}$ denotes the unit matrix with all rows corresponding to indices in $\mathcal{S}$ eliminated. Clearly, multiplication from the right by matrix V in equation 3.4.19 eliminates the $n(n-1)/2$ redundant equations. Further we have:

Lemma 3.4.3. *Matrix $V(I_{n^2} - A \otimes A)W$ is non-singular for all $k \in \mathcal{K}_n$.*

Proof. See Appendix A. □

Thus equation 3.4.19 has the unique solution

$$p = \overline{\text{vec}}(P) = [V(I_{n^2} - A \otimes A)W]^{-1}V\text{vec}(BB')$$

from which P can be recovered as $P = \overline{\text{vec}}^{-1}(p)$.

Example: Using the two methods described in the earlier part of this section the covariance matrices corresponding to the two and three-node models were obtained using the symbolic Matlab toolbox [oTC], as:

$$P_3 = \begin{pmatrix} \frac{1}{k_1(2-k_1)} & 0 & \frac{1}{k_1-2} \\ 0 & 1 & 0 \\ \frac{1}{k_1-2} & 0 & \frac{k_1}{2-k_1} \end{pmatrix} \tag{3.4.20}$$

and

$$P_5 = \begin{pmatrix} \frac{1}{k_1(2-k_1)} & 0 & -\frac{k_1-1}{(k_1-2)k} & \frac{1}{k_1-2} & \frac{(k_1-1)k_2}{(k_1-2)k} \\ 0 & 1 & 0 & 0 & 0 \\ -\frac{k_1-1}{(k_1-2)k} & 0 & \frac{k_1(k+2)}{k_2(2-k_1)(k_2-1)k} & \frac{(k_1-1)k_1}{(k_1-2)k} & \frac{(k+2)k_1}{(k_1-2)(k_2-2)k} \\ \frac{1}{k_1-2} & 0 & \frac{(k_1-1)k_1}{(k_1-2)k} & \frac{k_1}{2-k_1} & -\frac{(k_1-1)k_1k_2}{(k_1-2)k} \\ \frac{(k_1-1)k_2}{(k_1-2)k} & 0 & \frac{(k+2)k_1}{(k_1-2)(k_2-2)k} & \frac{(k_1-1)k_1k_2}{(2-k_1)k} & \frac{(k+2)k_1k_2}{(k_1-2)(2-k_2)k} \end{pmatrix} \tag{3.4.21}$$

respectively, where $k = k_1k_2 - k_2 - k_1$.

A still better method for calculating the covariance matrix of the state-vector is to use the special structure of the state-space model, which leads to a simple recursive updating algorithm. This is outlined in the following result:

Lemma 3.4.4. *Let (A_{2j+1}, B_{2j+1}) denote the $(j+1)-th$ node state-space model, depending on the j parameters $\{k_1, k_2, \ldots, k_j\}$ where $j \geq 1$. Then:*

1. *There is a state-space transformation defined by a permutation matrix Q_j, such that*

$$Q_jA_{2j+1}Q_j := A = \begin{pmatrix} A_{11} & 0 \\ A_{21} & A_{22} \end{pmatrix}$$

and

$$Q_jB_{2j+1} = B_{2j+1} := B$$

in which: (i) $A_{11} = A_{2j-1}$, (ii) A_{21} and A_{22} have rank one, and (iii) B is of the form $[B_1' \; 0_{2j-1}]'$.

2. *The Lyapunov equation $P - APA' - BB' = 0$ has a unique symmetric positive-*

semidefinite solution P for all $(k_1, k_2, \ldots, k_j) \in (0,2)^j$. Let P be partitioned conformably with A, i.e.,

$$P = \begin{pmatrix} P_{11} & P_{12} \\ P_{12}' & P_{22} \end{pmatrix}$$

where $P_{11} = P_{11}' \in \mathcal{R}^{(2j-1)\times(2j-1)}$, $P_{12} \in \mathcal{R}^{(2j-1)\times 2}$ and $P_{22} = P_{22}' \in \mathcal{R}^{2\times 2}$. Then $P_{11} = P_{2j-1}$ where P_{2j-1} is the covariance matrix of the j-th node model, i.e., the unique symmetric solution of the discrete Lyapunov equation:

$$P_{2j-1} - A_{2j-1} P_{2j-1} A_{2j-1}' - B_{2j-1} B_{2j-1}' = 0$$

Further, P_{12} and P_{22} have rank at most one and may obtained from the unique solutions of the linear equations:

$$P_{12} - A_{11} P_{12} A_{22}' = A_{11} P_{11} A_{21}'$$

and

$$\begin{aligned} P_{22} - A_{22} P_{22} A_{22}' &= A_{21} P_{11} A_{21}' + A_{22} P_{12}' A_{21}' \\ &+ A_{21} P_{12} A_{22}' \end{aligned}$$

respectively.

3. *If $(k_1, k_2, \ldots, k_j) \in (0,2)^j$, the Lyapunov equation:*

$$P_{2j+1} - A_{2j+1} P_{2j+1} A_{2j+1}' - B_{2j+1} B_{2j+1}' = 0$$

has a unique symmetric positive semi-definite solution given by:

$$P_{2j+1} = Q_j \begin{pmatrix} P_{2j-1} & P_{12} \\ P_{12}' & P_{22} \end{pmatrix} Q_j$$

Remark 3.4.1. The Lemma 3.4.4 shows that the covariance matrix of the $(j+1)$-th node model may be obtained recursively from the solution of the j-th node model by solving two linear equations of order $2(2j-1)$ and 4, respectively (in fact of order $2j-1$ and 2, taking into account that P_{12} and P_{22} have both rank at most one).

This can be achieved by the vectorisation approach outlined earlier. Hence, the bulk of the computation involving the solution of a $(2j-1) \times (2j-1)$ matrix equation is completely avoided. After P has been assembled from P_{2j-1}, P_{12} and P_{22}, P_{2j+1} may be obtained by reversing the permutation through matrix Q_j.

Proof. See Appendix A. □

Example: In this example we demonstrate the construction of the covariance matrix of the 3-node model using the covariance matrix of the 2-node model by applying the procedure outlined in Lemma 3.4.4. Let A_5 be the A-matrix of the standard state-space realisation of the 3-node system and Q_5 the matrix resulting by permuting the fourth and fifth rows and columns of the 5×5 identity matrix. Then,

$$Q_5 A_5 Q_5 = \begin{pmatrix} A_{11} & O \\ A_{21} & A_{22} \end{pmatrix}$$

where

$$A_{11} = \begin{pmatrix} 1 & -1 & 1 \\ 0 & 0 & 0 \\ -k_1 & k_1 & k_1 \end{pmatrix}, \quad A_{21} = \begin{pmatrix} 0 & 0 & -1 \\ 0 & 0 & k_2 \end{pmatrix} \quad \text{and} \quad A_{22} = \begin{pmatrix} 1 & 1 \\ -k_2 & -k_2 \end{pmatrix}$$

Let

$$P = \begin{pmatrix} P_{11} & P_{12} \\ P_{12}' & P_{22} \end{pmatrix}$$

be the solution of the Lyapunov equation

$$P - Q_5 A_5 Q_5 P Q_5 A' Q_5 = Q_5 B_5 B_5' Q_5 = B_5 B_5'$$

partitioned conformably with A. Then $P_{11} = P_3$, the covariance matrix of the standard 2-node state-space model given in equation (3.4.20), while P_{12} and P_{22} are the unique solutions of the matrix equations:

$$P_{12} = A_{11} P_3 A_{21}' + A_{11} P_{12} A_{22}' \tag{3.4.22}$$

and

$$P_{22} = A_{22} P_{22} A_{22}' + A_{21} P_3 A_{21}' + A_{22} P_{12}' A_{21}' + A_{21} P_{12} A_{22}' \tag{3.4.23}$$

respectively. Consider first equation 3.4.22. Since P_{12} has rank one, we can write its first and second columns as α and $\lambda\alpha$, respectively. Equation (3.4.22) can now be written as:

$$\begin{pmatrix} \alpha & \lambda\alpha \end{pmatrix} = (1+\lambda)\begin{pmatrix} A_{11}\alpha & -k_2 A_{11}\alpha \end{pmatrix} + \begin{pmatrix} s & -k_2 s \end{pmatrix}$$

where

$$s = \begin{pmatrix} \frac{k_1-1}{k_1-2} & 0 & -\frac{k_1(k_1-1)}{k_1-2} \end{pmatrix}'$$

leading to the two equations

$$\begin{aligned} \alpha &= (1+\lambda)A_{11}\alpha + s \\ \lambda\alpha &= -k_2[(1+\lambda)A_{11}\alpha + s] \end{aligned}$$

Substituting the first equation into the second gives $\lambda\alpha = -k_2\alpha$ and hence $\lambda = -k_2$ since $\alpha \neq 0$ (for $\alpha = 0$ implies that $s = 0$). Thus

$$\alpha = (1-k_2)A_{11}\alpha + s \quad \Rightarrow \quad \alpha = [I-(1-k_2)A_{11}]^{-1}s$$

Since A_{11} has rank one it can be factored as $A_{11} = pq'$, where p and q are column vectors, e.g., by choosing

$$p = \begin{pmatrix} 1 \\ 0 \\ -k_1 \end{pmatrix} \quad \text{and} \quad q = \begin{pmatrix} 1 \\ -1 \\ 1 \end{pmatrix}$$

Using the matrix inversion Lemma (see [HJ90]),

$$(I-(1-k_2)A_{11})^{-1} = (I-(1-k_2)pq')^{-1} = I+\frac{1-k_2}{1-(1-k_2)q'p}pq' = I+\frac{1-k_2}{k_1+k_2-k_1k_2}A_{11}$$

and hence, after some algebra,

$$\alpha = \left(I+\frac{1-k_2}{k_1+k_2-k_1k_2}A_{11}\right)s = \begin{pmatrix} \frac{k_1-1}{(k_1-2)(k_1+k_2-k_1k_2} \\ 0 \\ \frac{k_1(k_1-1)}{(2-k_1)(k_1+k_2-k_1k_2)} \end{pmatrix}$$

which implies that

$$P_{12} = \begin{pmatrix} \frac{k_1-1}{(k_1-2)(k_1+k_2-k_1k_2)} & \frac{k_2(k_1-1)}{(2-k_1)(k_1+k_2-k_1k_2)} \\ 0 & 0 \\ \frac{k_1(k_1-1)}{(2-k_1)(k_1+k_2-k_1k_2)} & \frac{k_1k_2(k_1-1)}{(k_1-2)(k_1+k_2-k_1k_2)} \end{pmatrix}$$

Next consider equation 3.4.23. Writing

$$P_{22} = \begin{pmatrix} x & y \\ y & z \end{pmatrix}$$

equation 3.4.23 can be written as

$$\begin{pmatrix} x & y \\ y & z \end{pmatrix} - \begin{pmatrix} x+2y+z & -k_2(x+2y+z) \\ -k_2(x+2y+z) & k_2^2(x+2y+z) \end{pmatrix} - \gamma \begin{pmatrix} 1 & -k_2 \\ -k_2 & k_2^2 \end{pmatrix} = 0$$

where we have defined

$$\gamma = \frac{k_1(2-k_1-k_2+k_1k_2)}{(k_1-2)(k_1k_2-k_1-k_2)}$$

Hence

$$\begin{pmatrix} x & y \\ y & z \end{pmatrix} = (x+2y+z+\gamma) \begin{pmatrix} 1 & -k_2 \\ -k_2 & k_2^2 \end{pmatrix}$$

Setting $\nu = x+2y+z+\gamma$ gives $x = \nu$, $y = -k_2\nu$ and $z = k_2^2\nu$. Thus

$$\nu = \nu - 2k_2\nu + k_2^2\nu + \gamma \quad \Rightarrow \quad \nu = \frac{\gamma}{k_2(2-k_2)}$$

and hence

$$P_{22} = \begin{pmatrix} \frac{k_1(2-k_1-k_2+k_1k_2)}{k_2(2-k_2)(k_1-2)(k_1k_2-k_1-k_2)} & \frac{k_1(2-k_1-k_2+k_1k_2)}{(2-k_1)(2-k_1)(k_1k_2-k_1-k_2)} \\ \frac{k_1(2-k_1-k_2+k_1k_2)}{(2-k_1)(2-k_1)(k_1k_2-k_1-k_2)} & \frac{k_1k_2(2-k_1-k_2+k_1k_2)}{(2-k_2)(k_1-2)(k_1k_2-k_1-k_2)} \end{pmatrix}$$

Constructing P from its blocks $P_{11} = P_3$, P_{12} and P_{22} and applying permutation Q_5 from the left and the right gives P_5 in the form given in equation 3.4.21.

3.4.4 Characterisation of Bullwhip effect

The covariance analysis carried out in the previous section allows us to analyse the effect of the inventory replenishment policies on the bullwhip effect. Recall that end-

customer demand has been modelled as a white-noise sequence. Hence, the variance of the demand signal at any node of the chain may be calculated easily from the covariance matrix. Consider as an example a three-node supply chain model. The orders placed by the second node (on the manufacturer) correspond to signal $z_{2,r}$ and we can write:

$$z_{2,r}(t) = C_{2,r}x_2(t) + D_{22,r}w_{2,r}(t) = C_{2,r}x_2(t) - k_2x_\Phi(t)$$

or

$$z_{2,r}(t) = Cx(t)$$

where

$$C = \begin{pmatrix} 0 & 0 & -k_2 & k_2 & -k_2 \end{pmatrix}$$

and

$$x'(t) = \begin{pmatrix} x'_1 & x'_2 & x_\Phi \end{pmatrix}$$

Thus the demand amplification factor can be obtained from the variance of z_{2r}, σ^2 which is given as:

$$\sigma^2 = E(z_{2,r}^2) = CP_5C' = \frac{k_1k_2(2 + k_1k_2 - k_1 - k_2)}{(2 - k_1)(2 - k_2)(k_1 + k_2 - k_1k_2)} \tag{3.4.24}$$

To find the regions in the (k_1, k_2) plane where demand amplification and demand attenuation occurs, this expression was set to one, and the resulting equation was solved to give k_2 as a function of k_1. This gives two solutions:

$$k_2 = f(k_1) = \frac{2 - 5k_1 + 2k_1^2 \pm \sqrt{4 - 12k_1 + 13k_1^2 - 4k_1^3}}{2(k_1 - 1)^2} \tag{3.4.25}$$

which are valid for $k_1 \neq 1$. It can be easily seen that the positive square root should be chosen, as with this choice, k_1 values in the interval $0 \leq k_1 \leq 2$ are mapped to k_2 values inside the same interval. When $k_1 = 1$ equation 3.4.24 (with $\sigma = 1$) shows that $k_2 = 1$. Strictly, equation 3.4.25 does not define k_2 when $k_1 = 1$, although it can be easily verified by applying de L'Hospital's rule (twice) that defining $k_2 = 1$ in this case makes the function $k_2 = f(k_1)$ continuous and differentiable at $k_1 = 1$.

The resulting curve is plotted in Figure 3.6, and indicates the boundary between the demand-amplification and demand-attenuation regions. As expected, aggressive replenishment policies (i.e., large values of k_1 and k_2) reinforce the bullwhip effect.

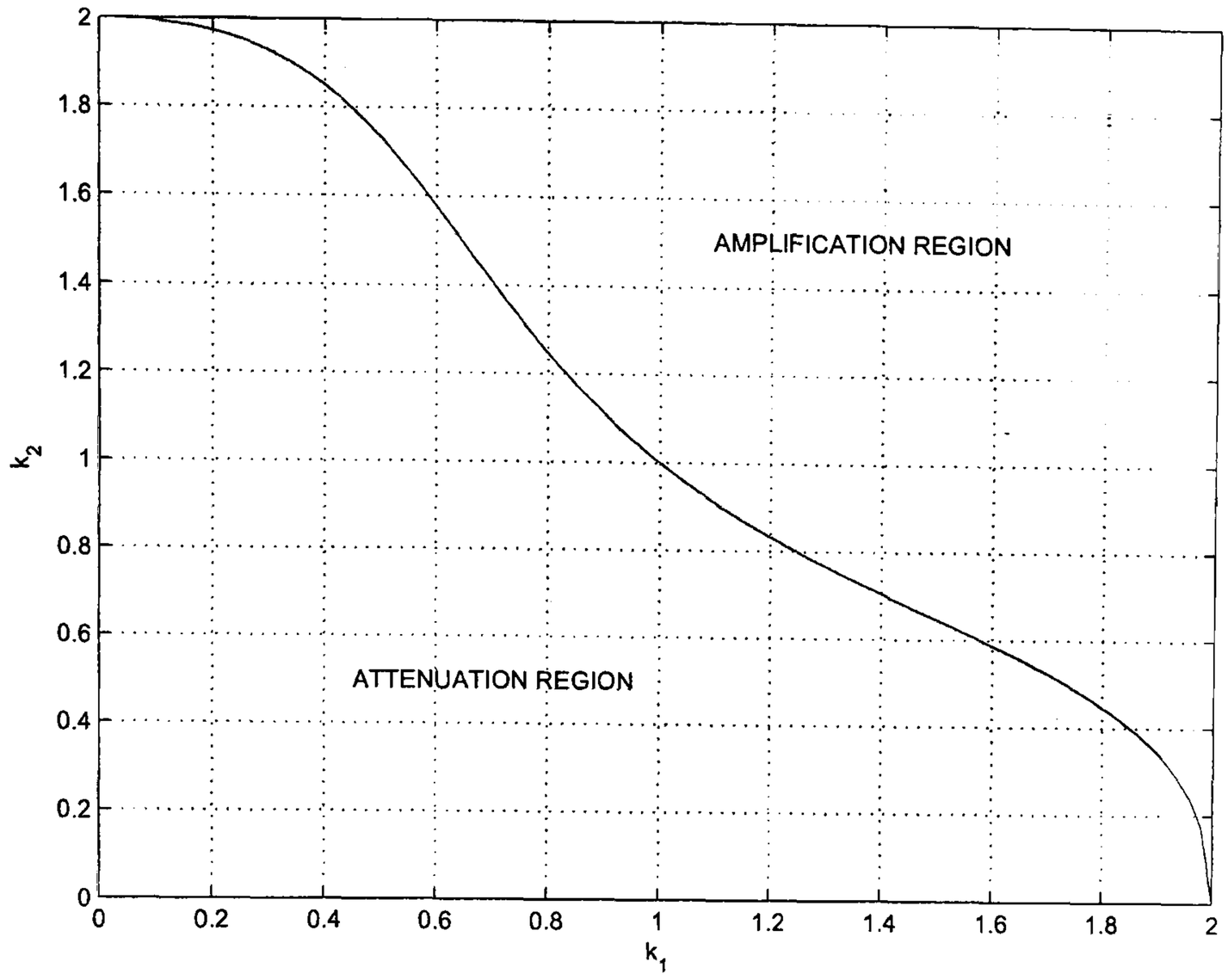

Figure 3.6: Boundary between demand amplification and attenuation regions

Example: In this example we illustrate the validity of the above expression with the results of a simple simulation based on a three-node supply-chain model. The first two nodes of the system (see Figure 3.3) representing the Distributor and Retailer were modelled in Simulink using equations 3.4.6, 3.4.7, 3.4.8 and 3.4.9, are shown in Figure 3.7. The Manufacturer is modelled as a simple unit delay, i.e., it is assumed that he is capable to meet any order placed on him with a delay of one time-period.

End-customer demand was modelled as a normally-distributed white-noise sequence with mean $\mu = 100$ and variance $\sigma^2 = 1$. To ensure that the the linearity assumptions are satisfied throughout the non-linear simulation (i.e., that each node of the chain has sufficient inventory to meet downstream orders), inventories for both nodes were initialised as $I_1(0) = I_2(0) = 1000$, while both set-points were

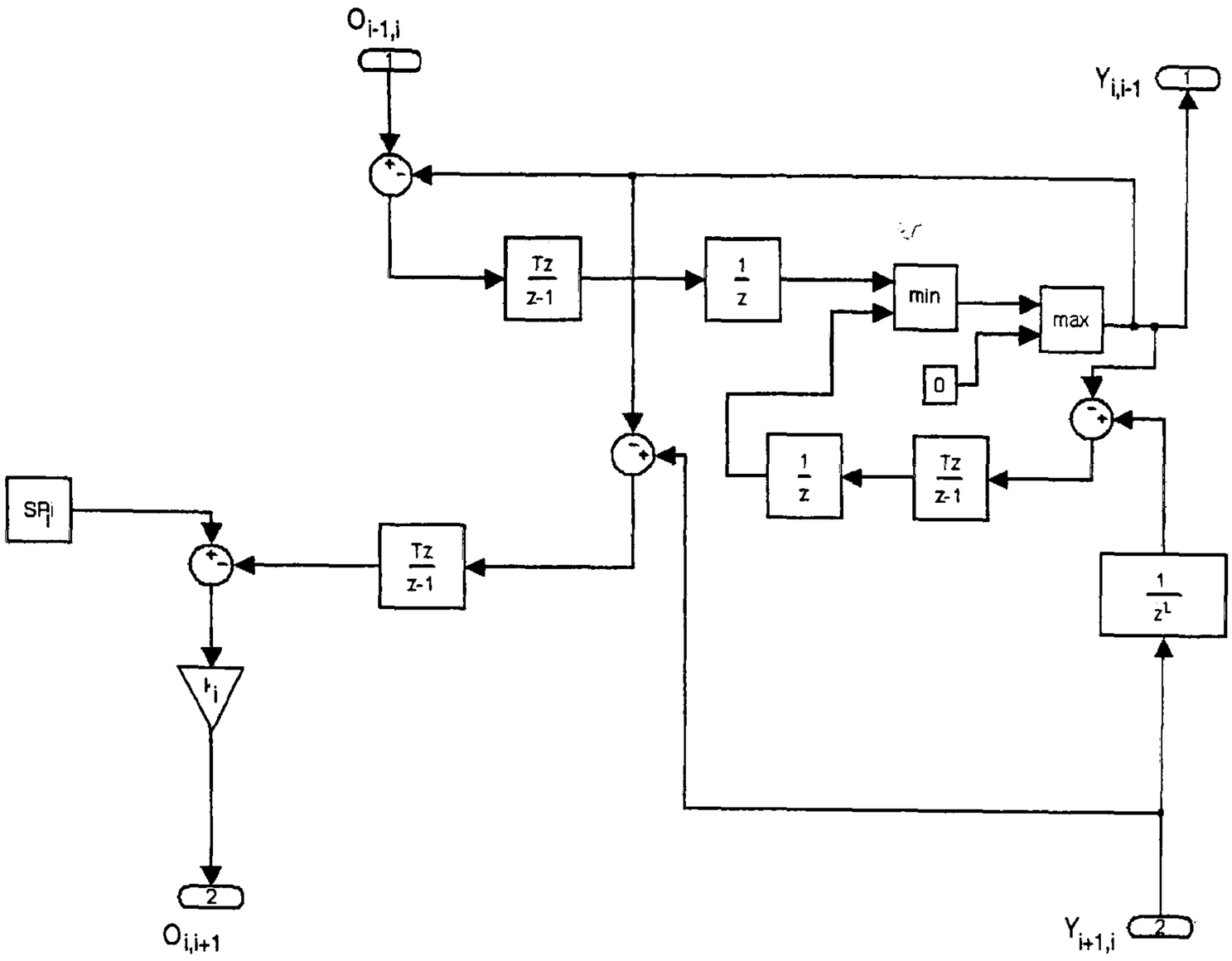

Figure 3.7: Detailed model of each supply chain node

set as $SP_1 = SP_2 = 100$. Stock replenishment gains were set as $k_1 = 0.9$ and $k_2 = 1.6$, respectively. The simulation was then carried out over $N = 10000$ time steps. After removing the first few samples of the time series corresponding to $z_{2,r}$ (representing demand placed on the Manufacturer) to eliminate any effects of initial transients, estimates of the mean and standard deviation of $z_{2,r}$ were obtained as $\hat{\mu}_{z_{2,r}} = 99.96$ and $\hat{\sigma}_{z_{2,r}} = 1.707$. These compare favourably with the theoretical values $\mu_{z_{2,r}} = 100$ and $\sigma_{z_{2,r}} = 1.704$ obtained via equation 3.4.24. The theoretical and empirical probability density distributions of $z_{2,r}$ are shown in Figure 3.8.

Next consider the $n = 4$-node model. Symbolic computations in this case give the variance of $z_{3,r}$ (demand on manufacturer) as:

$$\sigma^2 = \frac{\prod_i k_i\{\zeta - \theta + \prod_i k_i(14 - \gamma - \prod_i k_i)\}}{\xi\tau\upsilon(k_1 - 2)(k_2 - 2)(k_3 - 2)}$$

where:

$\zeta = \sum_i (4k_i - 2k_i^2)$,

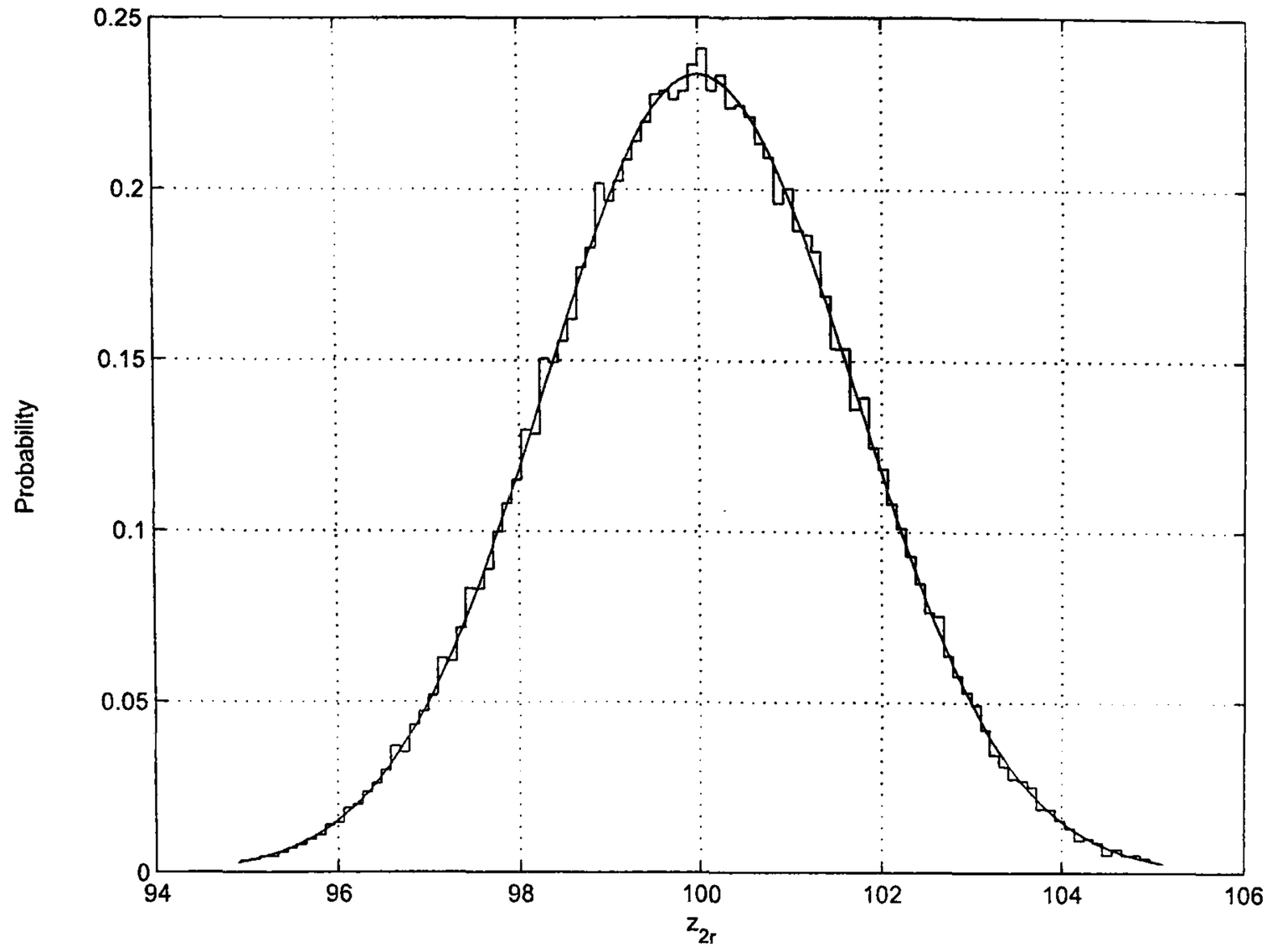

Figure 3.8: Theoretical and empirical distributions of $z_{2,r}$

$$\theta = 8\sum_{i\neq j} k_i k_j + \sum_{i\neq j} k_i^2 k_j(3-k_j),$$

$$\gamma = 5\sum_{i} k_i + 2\sum_{i\neq j} k_i k_j,$$

$$\xi = (k_1 k_2 - k_1 - k_2),$$

$$\tau = (k_1 k_3 - k_1 - k_3),$$

$$\upsilon = (k_2 k_3 - k_3 - k_2)$$

Note that all sums and products are taken over the three parameters (k_1, k_2, k_3). To characterise the bullwhip effect in this case we set $\sigma^2 = 1$ and solve the resulting equation (symbolically) to obtain a function of the form $k_3 = f(k_1, k_2)$ which characterises the surface boundary between the gain-amplification and gain-attenuation regions in (k_1, k_2, k_3) parameter space. It can be verified numerically that for each pair (k_1, k_2) in the square $\mathcal{K} = [0\ 2] \times [0\ 2]$ there is exactly one real root of the equation $\sigma^2 = 1$ which lies in the interval $0 \leq k_3 \leq 2$. The function $k_3 = f(k_1, k_2)$ is plotted in Figure 3.9. Due to the high complexity of the expressions involved, the explicit form of this function is not included here. In practice, it is preferable to generate a table of the surface over a grid of (k_1, k_2) values and use interpolation techniques if higher accuracy is required.

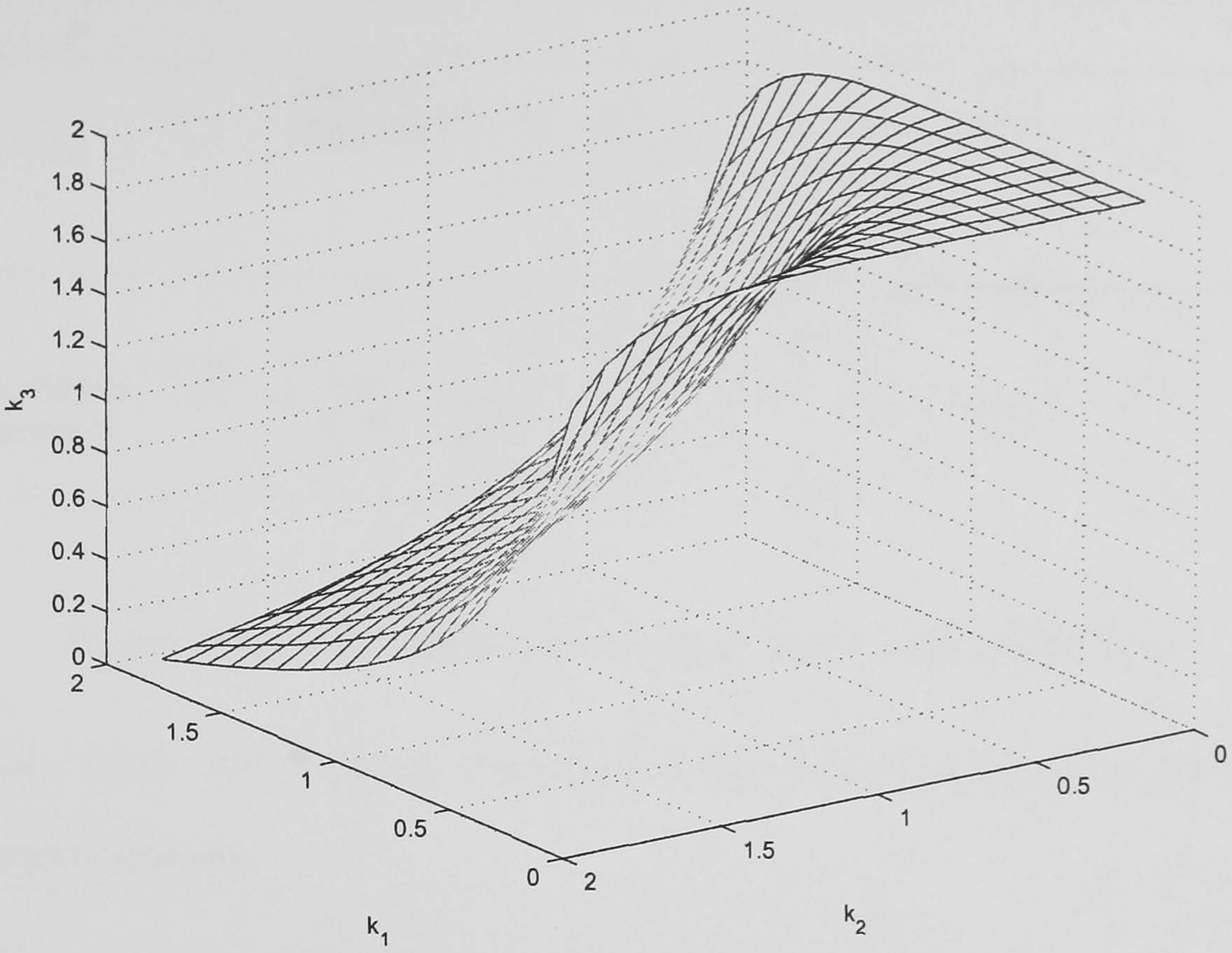

Figure 3.9: Boundary between demand amplification and attenuation regions: 4-node model

3.4.5 Stochastic state space for analysing the bullwhip effect under autoregressive customer demand profiles

In previous section a simple stochastic multi-node supply chain state-space model is developed and its properties analysed in the steady-state, under white noise end-customer demand-profiles. Although the white-noise demand profile offers the advantage of simplicity and modelling analysing convenience, it is unrealistic for real supply chains (as it ignores, for example, trends, seasonal variations or more complex patterns). In this part of the thesis we extend our study and we consider autoregressive (AR) customer demand profiles.

We consider again the series supply chain depicted in Figure 3.4.5 consisting of three stages (e.g., Manufacturer, Distributor and Retailer) and we assume that the customer demand $w(k)$ is driven through an (AR) filter such as:

$$w(k) = \frac{(1-\alpha)z^{-1}}{1-\alpha z^{-1}} e(k) \tag{3.4.26}$$

where α is the demand autocorrelation coefficient with $|\alpha| < 1$ and $e(k)$ is a white

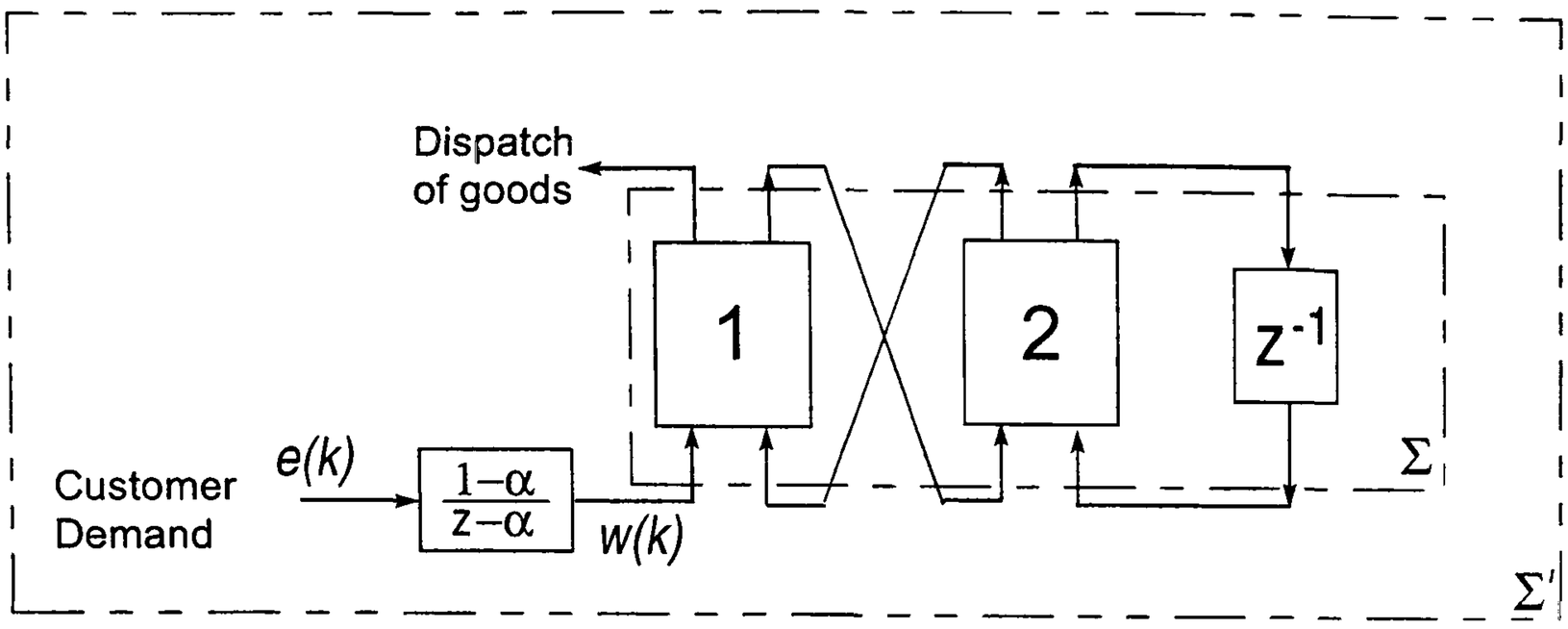

Figure 3.10: The three-node supply chain with (AR) filter

noise signal. Thus, the external customer demand pattern faced by Retailer is an autoregressive process:

$$\begin{aligned}
(1-\alpha z^{-1})w(k) &= (1-\alpha)z^{-1}e(k) \Rightarrow \\
w(k)-\alpha w(k-1) &= (1-\alpha)e(k-1) \Rightarrow \\
w(k) &= \alpha w(k-1)+(1-\alpha)e(k-1) \Rightarrow \\
w(k+1) &= \alpha w(k)+(1-\alpha)e(k)
\end{aligned}$$

Then the model equations for the filter H are given:

$$x_f(k+1) = A_f x_f(k) + B_f e(k) \tag{3.4.27}$$

$$y_f(k) = C_f x_f(k) + D_f e(k) \tag{3.4.28}$$

and the state space of the filter is:

$$\begin{pmatrix} x_f(k+1) \\ w(k+1) \end{pmatrix} = \begin{pmatrix} A & B \\ 0 & \alpha \end{pmatrix} \begin{pmatrix} x_f(k) \\ w(k) \end{pmatrix} + \begin{pmatrix} 0 \\ 1-\alpha \end{pmatrix} e(k)$$

where $x_f(k)$ represents the state of the AR filter, $y_f(k)$ is the output of the filter $w(k)$, $A_f = \alpha$, $B_f = (1-\alpha)$, $C_f = 1$ and $D_f = 0$.

Before study further the impact of the (AR) filter to the supply chain model it is useful to understand better the behaviour of the smoothing filter H. Such filters

are low-pass filters and are used mainly in systems to remove unwanted noise, from a relatively slowly-varying signal and to create correlated data in multiple steps. By considering again the filter equation 3.4.26, the magnitude of the filter $|H|$ is:

$$\begin{aligned} |H(z)| &= \left|\frac{(1-\alpha)z^{-1}}{1-\alpha z^{-1}}\right| \\ &= \left|\frac{(1-\alpha)}{1-\alpha(cos\omega + jsin\omega)}\right| \end{aligned}$$

and the magnitude of the corresponding spectral function $|H(\omega)|$ is:

$$\begin{aligned} |H(\omega)| &= \left|\frac{(1-\alpha)}{1-\alpha(cos\omega + jsin\omega)}\right| \\ &= \left|\frac{(1-\alpha)}{(1-\alpha cos\omega) + j\alpha sin\omega}\right| \\ &= \frac{(1-\alpha)}{\sqrt{(1-\alpha cos\omega)^2 + \alpha^2 sin^2\omega}} \\ &= \frac{(1-\alpha)}{\sqrt{(1+\alpha^2 - 2\alpha cos\omega)}} \end{aligned}$$

Figure reffreqfilter shows the frequency response of the filter for three different α values, 0.2, 0.5 and 0.8.

Let us now denote as $\underline{x}(k+1) = A\underline{x}(k) + Bw(k)$ the state space equation of the previous section supply chain system Σ driven now by $w(k)$. Then, the overall state space form of the new supply chain system Σ' with the (AR) filter H is:

$$\begin{pmatrix} \underline{x}(k+1) \\ x_f(k+1) \end{pmatrix} = \begin{pmatrix} A & BC_f \\ 0 & A_f \end{pmatrix} \begin{pmatrix} \underline{x}(k) \\ x_f(k) \end{pmatrix} + \begin{pmatrix} 0 \\ B_f \end{pmatrix} e(k)$$

If we choose $x_f(k) = w(k)$ and $y(k) = z_{i,l}$ the output of the system Σ', then we can derive the state space form for the three-stage supply chain as:

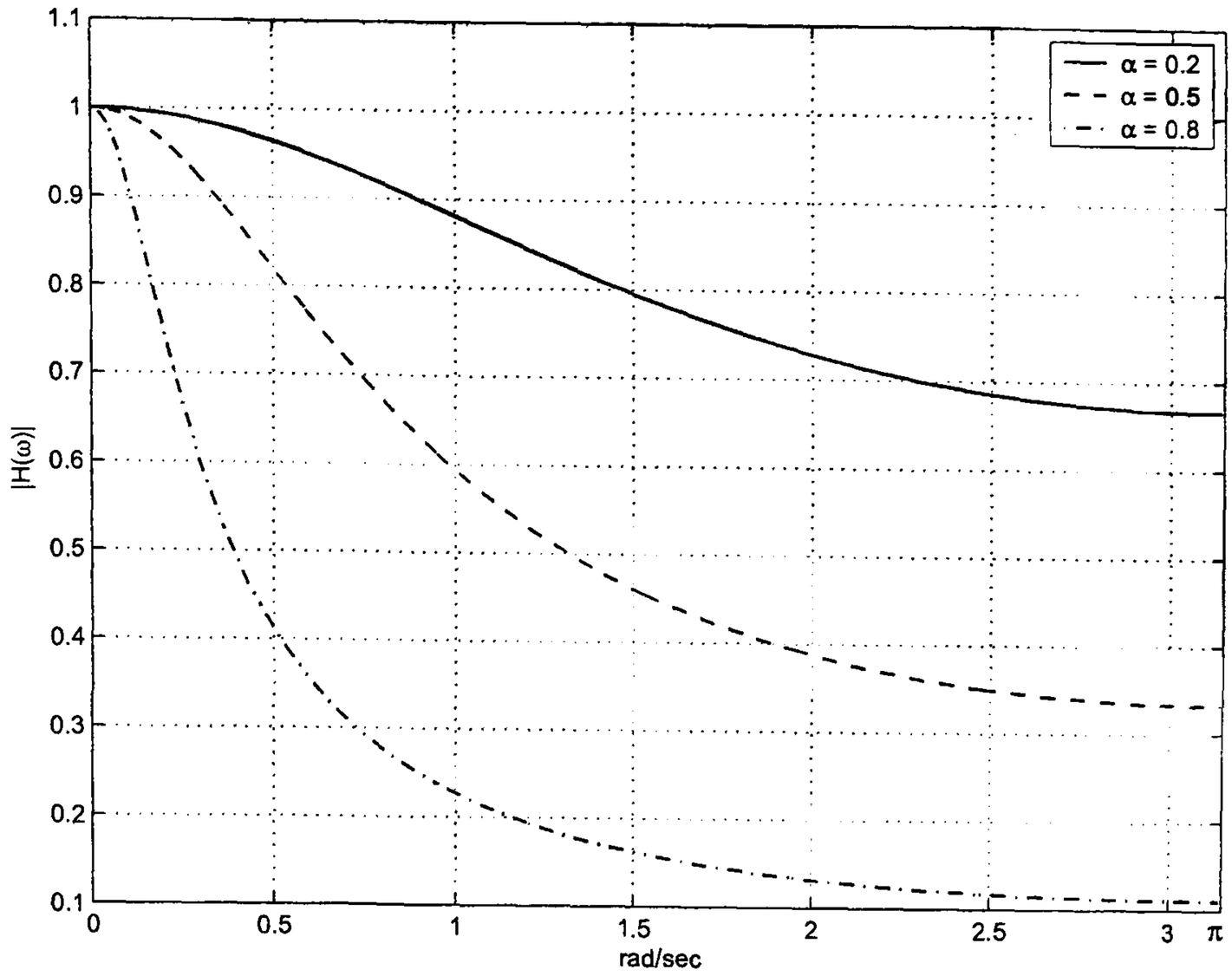

Figure 3.11: Frequency response of the (AR) filter

$$\begin{pmatrix} x_1(k+1) \\ x_2(k+1) \\ x_\phi(k+1) \\ w(k+1) \end{pmatrix} = \begin{pmatrix} 1 & -1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & -1 & 0 & 0 \\ -k_1 & k_1 & 0 & -k_1 & 0 & 0 \\ 0 & 0 & -k_2 & k_2 & -k_2 & 0 \\ 0 & 0 & 0 & 0 & 0 & \alpha \end{pmatrix} \begin{pmatrix} x_1(k) \\ x_2(k) \\ x_\phi(k) \\ w(k) \end{pmatrix} + \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1-\alpha \end{pmatrix} e(k)$$

and

$$y(k) = \begin{pmatrix} 0 & 1 & 0 & 0 & 0 & 0 \end{pmatrix} \begin{pmatrix} x_1(k) \\ x_2(k) \\ x_\phi(k) \\ w(k) \end{pmatrix}$$

The above state space form does not include the set-points SP_1 and SP_2. If we wish to monitor the changes on set-point levels we can write:

$$\begin{pmatrix} x_1(k+1) \\ x_2(k+1) \\ x_\phi(k+1) \\ w(k+1) \\ SP_1(k+1) \\ SP_2(k+1) \end{pmatrix} = \begin{pmatrix} 1 & -1 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & -1 & 0 & 0 & 0 & 0 \\ -k_1 & k_1 & 0 & -k_1 & 0 & 0 & k_1 & 0 \\ 0 & 0 & -k_2 & k_2 & -k_2 & 0 & 0 & k_2 \\ 0 & 0 & 0 & 0 & 0 & \alpha & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 \end{pmatrix} \begin{pmatrix} x_1(k) \\ x_2(k) \\ x_\phi(k) \\ w(k) \\ SP_1(k) \\ SP_2(k) \end{pmatrix} + \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1-\alpha \\ 0 \\ 0 \end{pmatrix} e(k)$$

The characterisation of the bullwhip effect involves again the computation of the demand amplification factor which can be obtained from the symbolic computation of the variance $z_{2,r}/w_{1,l}$:

$$E(z_{2,r}/w_{1,l}) = \frac{k_1 k_2\{\alpha k_1 k_2 \zeta - \alpha\theta - (\xi - 2\alpha^2 + 2)\}}{(k_1-2)(k_2-2)(\alpha k_2 + 1 - \alpha)(\alpha k_1 + 1 - \alpha)\xi} \tag{3.4.29}$$

where:

$\zeta = [\alpha k_1 k_2 - (2\alpha + 1)(k_1 + k_2) + 5\alpha + 4]$,

$\theta = [(3\alpha + 2)(k_1 + k_2) - (k_1^2 + k_2^2)(\alpha + 1)]$,

$\xi = [k_1 k_2 - k_1 - k_2]$

which are valid for $k_1 \neq 1$, $k_2 \neq 1$ and $|\alpha| < 1$.

The regions in the (k_1, k_2) where demand amplification and demand attenuations occurs, we must set again Equation 3.4.29 to 1 and solve the resulting equation to obtain a function of the form $k_2 = f(k_1, \alpha)$. It can be verified numerically that there exists one real root k_2 which lies in the interval $0 \leq k_2 \leq 2$. Figure 3.4.5 depicts the plot of the function $k_2 = f(k_1, \alpha)$ for three different values of α ($\alpha = 0.2, \alpha = 0.5, \alpha = 0.8$). It can be inferred from Figure 3.4.5 that higher values of α increase slightly the demand attenuation region while for small values the boundary between demand amplification and demand attenuation is the same obtained without the (AR) filter. As expected, smoothing customer demand profiles can lead to a slightly increment of k_1 and k_2 without the danger of demand amplification due to bullwhip effect. Another observation derived from the use of the (AR) filter involves the special case: $\alpha = 0$ in which the input signal is unbiased from the (AR) filter.

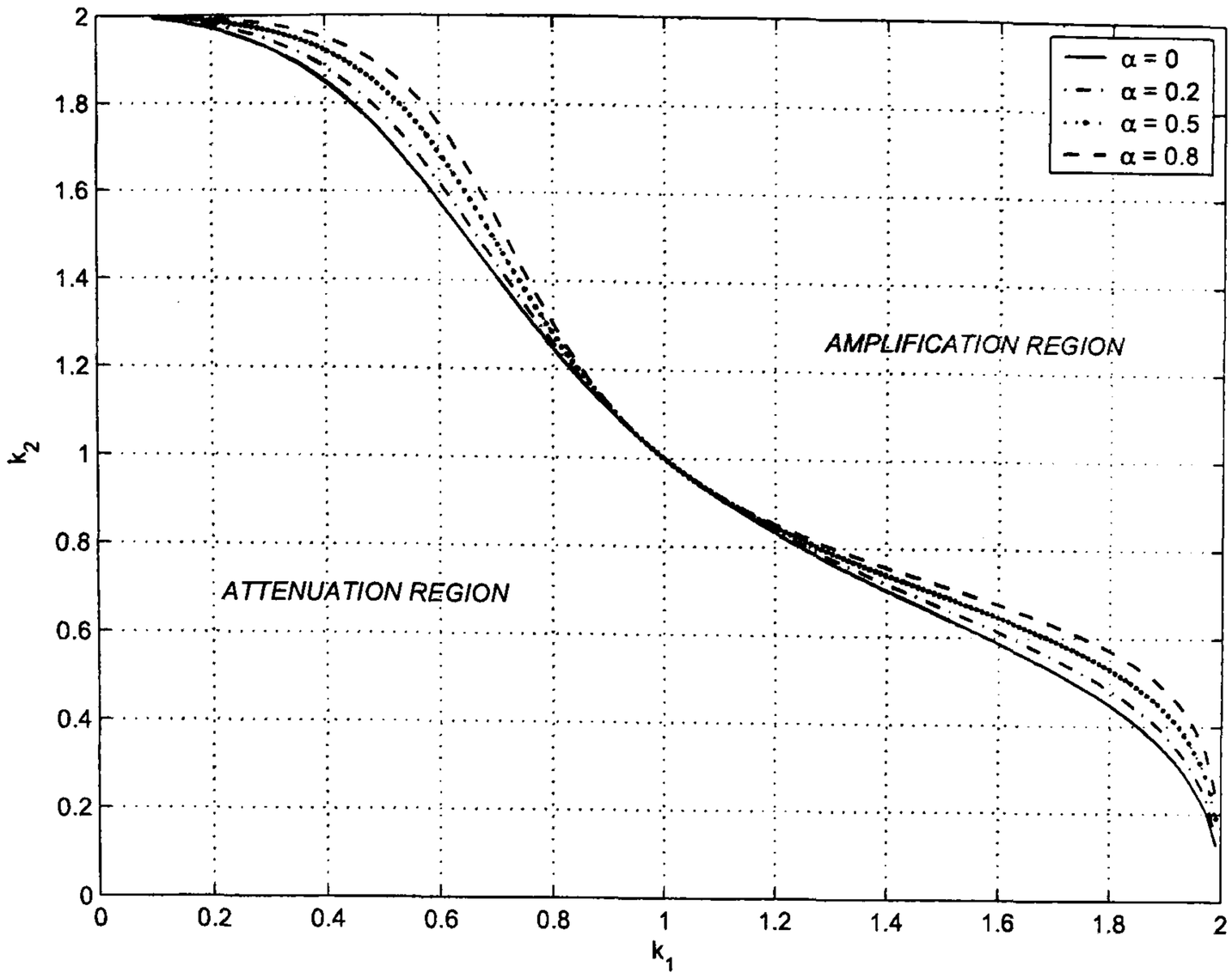

Figure 3.12: Boundary between demand amplification and attenuation regions with the (AR) filter

3.4.6 Customer service level

The models described so far have assumed that no back-orders (or backlogs) are allowed and that all demand must be set. Although this assumption has been considered mainly for the difficulties appearing in analysing the supply chain model, it is also a reasonable view when shortages - due to back-ordering - are very expensive. There are, however circumstances where planned backlogs are beneficial. An example comes when the cost of keeping an item in stock is higher than the profit of selling it, especially when a company holds stocks of the complete product range. On the other hand this has a weighty impact on the relationship between customer and retailer. Customers who experience shortages are likely to divert at least some future retailers to more reliable ones.

The dissatisfaction level of customer service is proportional to the time (order cycle time [Bal04]) needed for the ordered products to reach customer site. In other words, the more customers wait, the more look dissatisfied about the service. It is worth remarking that customers are willing to wait, and retailers are committed to

meet their demands. Retailers frequently set and claim to meet a customer service level often expressed as a percentage of given size delivered within a time frame. In this section we measure the size of the back-orders according to order cycle time when retailers face shortages. Our analysis is based on the series supply chain depicted in Figure 3.3. We remove the assumption that there is always enough stock at Retailer to meet the demand, so that $Y_{1,0}(t) \neq O_1^{\star}(t)$.

Simulation model

In order to determine the service level we use the MATLAB/SIMULINK tool. The structure of the Retailer node is given again in Figure 3.4. We consider the following example that measures the order cycle times and the total logistic costs for Retailer under different simulation parameters. Customer service level is defined as the percentage of goods demanded that are met from Retailer's stock. Logistic costs are associated with the *Holding Cost (HC)*, *Ordering Costs (OC)* and *Shortage Costs (SC)*. Holding costs is the cost of holding one good in stock for one period of time, Ordering Cost (OR) is the cost of placing a routine order for the good and might include telephone costs, receiving, use of equipment, expediting and quality checks and Shortage cost which is the cost of having a shortage and not being able to meet demand from stock.

Example: We consider a three-node series supply chain (i.e., Retailer, Distributor and Manufacturer). Customer demand is assumed as a normally distributed signal with mean $\mu = 20$ and variance $\sigma = 3$. Initial inventories (I_0) of both Retailer and Distributor are set to 15, 20, and 30. We proceed to three different values of Retailer's inventory replenishment gain factor k_1: 0.8, 1 and 1.2, while the corresponding Distributor's gain is set at a fixed value of 1.

The Holding cost each period time is calculated as £.2 per good while the cost for ordering purposes is £.05 per good per order. Costs due to backorders are not calculated numerically for the reason that they are often intangible (e.g., additional clerical costs or loss of future sales). Thus, we calculate instead the number of

Table 3.1: Simulation results of the supply chain model with backorders

k_1	k_2	Retailer	Distributor	1st Year HC	1st Year OC	1st Year SC	2nd Year HC	2nd Year OC	2nd Year SC	3rd Year HC	3rd Year OC	3rd Year SC	Total Cost	Service Level
0.8	1	15	15	675	169	Low	674	168	None	674	168	None	2528	45%
0.8	1	20	20	900	225	Low	898	225	None	898	225	None	3371	60%
0.8	1	30	30	1352	337	Low	1348	337	None	1348	337	None	5059	90%
1	1	15	15	732	183	High	729	182	High	729	183	High	2738	50%
1	1	20	20	976	244	High	972	243	High	972	244	High	3651	67%
1	1	30	30	1477	364	None	1461	365	None	1460	365	None	5492	100%
1.2	1	15	15	732	220	High	729	219	High	729	220	High	2849	50%
1.2	1	20	20	976	293	High	972	292	High	972	293	High	3798	67%
1.2	1	30	30	1708	367	None	1480	433	None	1460	438	None	5886	97%

backorders occurred annually. In those cases where the stock is insufficient for more than 80% of period time we refer the Shortage cost as High while for backorders less than 20% the Shortage cost is considered as Low.

Simulation results

Table 4.1 summarises the simulation results for a total simulation period time of 3 years. The Total cost is the sum of Holding and Ordering costs at the end of the simulation process. Table 4.1 shows also the corresponding Service Level for each scenario. The results have been rounded to the nearest integer.

The results indicate that the Service Level is better when k_1 is either 1 or 1.2 (both values settings have given similar results). More specifically the Service Level has been improved about 11% when both initial inventories of Retailer and Distributor I_0 are set to 15.

Regarding the Total costs for different replenishment policies we recall that for all the initial inventory settings when the gain factor was first set to $k_1 = 1$, simulation results have shown that Total costs have been reduced by 7.67% and 7.88% for $I_0 = 20$ and $I_0 = 30$, respectively when k_1 was changed to 0.8. In cases when Retailer follows a replenishment policy with $k_1 = 1.2$, Total costs have been increased approximately by 4% and 7.1% for $I_0 = 20$ and $I_0 = 30$, respectively.

Another useful observation has been derived for the differences in Total costs when both Retailer and Distributor use different initial inventory points by keeping the gain factor invariable. For initial inventory levels of 15 goods Total costs have

been increased by 33% and 100% when those levels have been augmented to initial values of 20 and 30, respectively. These differences may be considerable indeed when $k_1 = 1$ or $k_1 = 1.2$ and when initial inventory position is not set to 30, due to High Shortage costs.

It is also clear that aggressive ordering policies described by setting the gain factor $k_1 = 1.2$, do not guarantee necessarily better Service Level. On the contrast costs have been increased significantly. This remarkable outcome has been also marked in previous section where is noted that the selection of $k_1 = 1.2$ and $k_2 = 1$ leads to bullwhip effect Figure 3.6 which has tremendous impact on increment of costs.

Figure 3.13 and figure 3.14 depict the number of delay days between orders placed by customers and corresponding delivered goods. Both initial inventory values for Retailer and Distributor are set to $I_0 = 30$. As it can be shown for $k_1 = 1.2$ some orders "wait" even more than 30 days to be fulfilled while the majority of orders delivered in the first 5 days. In case of $k_1 = 1$ the maximum period of waiting is 7 days while the main volume of goods are delivered in 4th and 5th day. These figures corroborate the above remark that higher gain factor does not lead necessarily to better Service Level.

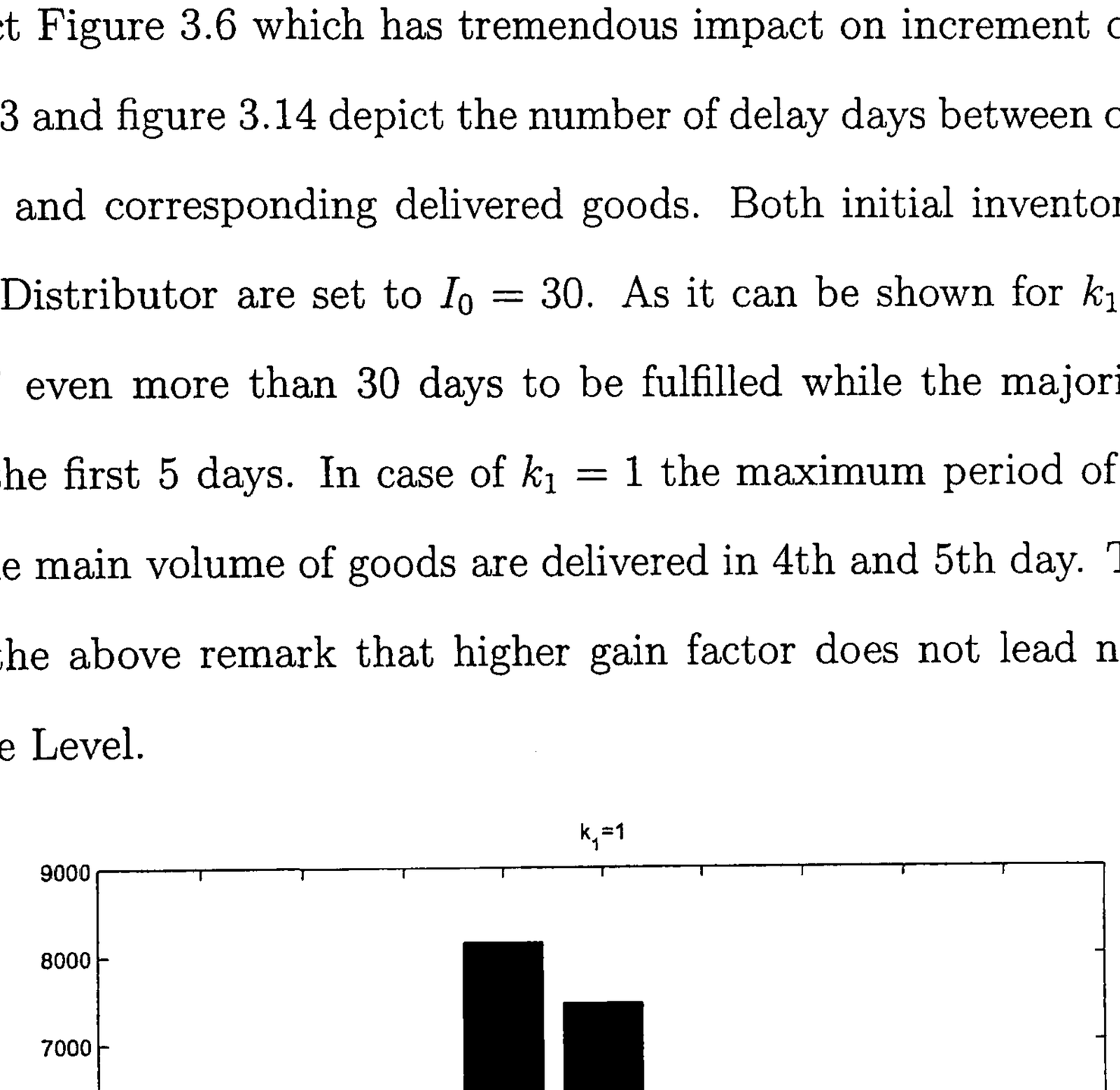

Figure 3.13: Number of delay days between orders and deliveries when $k_1 = 1$

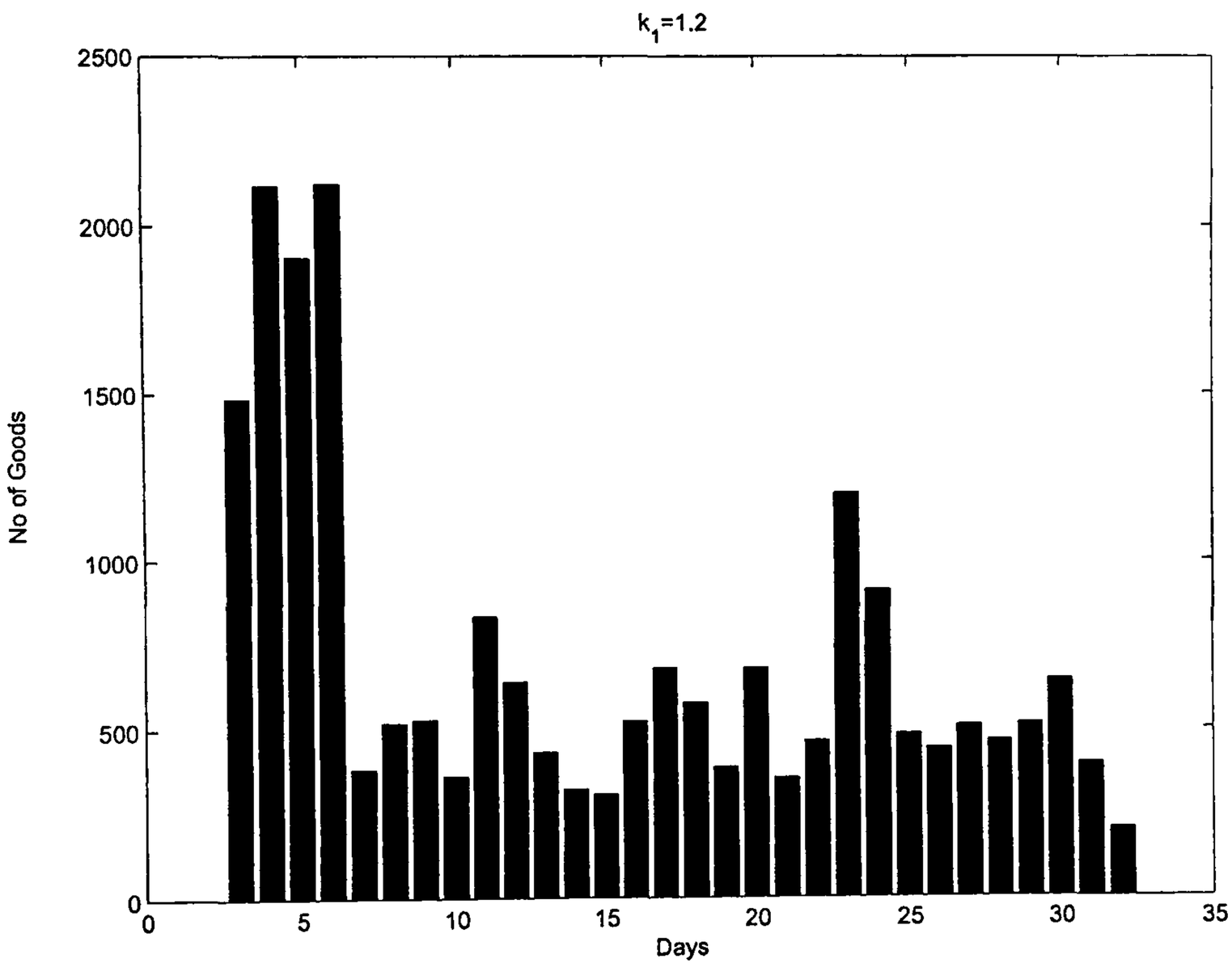

Figure 3.14: Number of delay days between orders and deliveries when $k_1 = 1.2$

Chapter 4

Analysis of optimal policies, information-sharing and estimation methods in series supply chain

4.1 Introduction

In previous chapter we have seen that bullwhip effect is a critical issue in the area of supply chain management. Aggressive replenishment policies of the downstream participants lead to demand amplification in the procurement volumes of upstream nodes. As a result, upstream participants affected by demand fluctuations have seen excessive inventories, poor product forecasts, insufficient or excessive capacities, poor customer service due to unavailable products or long backlogs, and uncertain production planning. Various techniques and methods have been proposed by researchers and practitioners in order to cope with changing demands and dampen fluctuations. In this chapter we focus on information-sharing techniques between neighbouring participants and estimation schemes followed by upstream members in a series supply chain.

The three-node model adopted from previous chapter is subsequently analysed in detail under information-sharing and the optimal policy is derived, which minimises inventory fluctuations (and inventory mean) under a probabilistic constraint related to downstream demand. It is shown that this policy can never lead to demand amplification in the chain, as long as the gain parameter k_i of the downstream node

i lies in the stability region.

In cases where information-sharing is infeasible the only available information to a particular node arises only from its interaction with neighbouring nodes. In this section local estimation schemes are investigated under which the parameters of the adjacent node can be estimated from (local) historical data. The main results and conclusions are illustrated via numerous examples and simulations.

4.2 Analysis of information-sharing and optimal policies

In this chapter we specialise our system to a three node model. We assume linear dynamics (i.e., that all inventories are sufficiently high to meet downstream demand with no back-orders). The manufacturer (node 3) is still modelled as a unit delay, i.e., he delivers the requested products with a delay of one time period. Assume further that customer demand is normally distributed as $e(t) = O_{0,1}(t) \sim N(\mu, \sigma^2)$. Provided that the system is stable ($0 < k_1 < 2$ and $0 < k_2 < 2$) all signals in the limit are stationary. The expected values of the state variables can be found using the state space model, which is of the form:

$$x(t+1) = Ax(t) + Be(t) + F(SP)$$

where SP is the (deterministic) vector of set-points $SP = (SP_1 \ SP_2)'$ (assumed constant). Thus under stationary conditions,

$$x(t) = (I - A)^{-1}Be(t) + (I - A)^{-1}F(SP)$$

and hence

$$E[x(t)] = \mu(I - A)^{-1}B + (I - A)^{-1}F(SP)$$

Note that the indicated matrix inverse exists as the spectral radius of A is less than one as long as $0 < k_1 < 2$ and $0 < k_2 < 2$. Thus, the five state variables are

distributed as:

$$\begin{aligned}
IP_1(t) &\sim N\left(SP_1 - \frac{\mu}{k_1}, \frac{\sigma^2 k_1}{2-k_1}\right) \\
Y_{1,0}(t) &\sim N(\mu, \sigma^2) \\
IP_2(t) &\sim N\left(SP_2 - \frac{\mu}{k_2}, \frac{\sigma^2 k_1(k_1k_2 - k_1 - k_2 + 2)}{k_2(2-k_1)(2-k_2)(k_1+k_2-k_1k_2)}\right) \\
Y_{2,1}(t) &\sim N\left(\mu, \frac{\sigma^2 k_1}{2-k_1}\right) \\
O_{2,3}(t) &\sim N\left(\mu, \frac{\sigma^2 k_1 k_2(k_1k_2 - k_1 - k_2 + 2)}{(2-k_1)(2-k_2)(k_1+k_2-k_1k_2)}\right)
\end{aligned}$$

Further on noting that

$$IP_2 - O_{1,2} = k_1 IP_1 - k_1 Y_{1,0} + IP_2 + k_1 Y_{2,1}$$

we obtain

$$\mathrm{Var}(IP_2 - O_{1,2}) = CP_5C', \quad C = \begin{pmatrix} k_1 & -k_1 & 1 & k_1 & 0 \end{pmatrix}$$

and hence,

$$IP_2 - O_{1,2} \sim N\left(SP_2 - \frac{\mu(k_2+1)}{k_2}, \frac{\sigma^2 k_1(2 - 5k_2k_1 + k_2^2k_1 - k_2^3 + k_2^3k_1 - k_1 + 3k_2 + 4k_2k_1^2 - 2k_2^2k_1^2)}{k_2(2-k_2)(k_1+k_2-k_2k_1)(2-k_1)}\right)$$

Note that the mean of IP_i $(i = 1, 2)$ does not track the set-point SP_i exactly, but with a steady-state error μ/k_i characteristic of type zero feedback systems. As expected, the information pattern is asymmetric, i.e., node 2 (Distributor) is affected by the inventory policy of node 1 (Retailer) but not vice versa. Suppose now that the Retailer makes his policy gain-factor k_1 known. In this case the Distributor can make use of this information to minimise his own costs, typically related to excessive inventory levels. Although this objective is situation-specific (e.g., due to possible existence of capacity constraints, depreciation effects, etc) it is reasonable to assume that the objective of the Distributor is to minimise both his average inventory and his inventory fluctuations. Note that in our model the Distributor is always capable of controlling his average inventory-level through his choice of SP_2, which can be used

to shift $E(IP_2)$ to any required level. In fact, a more general cost function could be formulated, involving the integral of an appropriate inventory cost-function, weighted by the distribution of IP_2.

An additional requirement is that the Distributor should have enough inventory to meet (fluctuating) downstream demand, at least for most orders placed on him. This is in order to ensure the smooth operation of the chain, to which he has an interest as a participant. One way of modelling this requirement is to include explicit penalty-terms in the Distributor's 'objective function', reflecting real or virtual costs (e.g., penalty terms for not fulfilling a contract, loss of sales due to Customer dissatisfaction, etc). Here we impose a probabilistic constraint for fulfilling orders, i.e., we require that $\text{Prob}[IP_2 \leq O_{1,2}] \leq \delta$ for some (small) parameter δ.

Let $\Phi(z)$ denote the probability density function of the normal distribution $N(0,1)$, i.e.:

$$\Phi(z) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{z} e^{-\xi^2/2} d\xi$$

Then, using the distribution of $IP_2 - O_{1,2}$ above, the 'order-fulfilling' constraint takes the form:

$$SP_2 - \frac{\mu(k_2+1)}{k_2} + \sigma\Phi^{-1}(\delta)\sqrt{\frac{k_1(2-5k_2k_1+k_2^2k_1-k_2^3+k_2^3k_1-k_1+3k_2+4k_2k_1^2-2k_2^2k_1^2)}{k_2(2-k_2)(k_1+k_2-k_2k_1)(2-k_1)}} \geq 0 \tag{4.2.1}$$

Thus, the optimisation problem faced by the Distributor is to choose his inventory replenishment policy parameters, k_2 and IP_2, to minimise his inventory costs subject to the constraint of equation 4.2.1. (Note that parameter k_1 is not under the control of the Distributor and has been assumed to be fixed and known). Rather than attempting to solve this constrained optimisation for a general inventory cost function, we have chosen the following (more revealing) procedure: For any given k_1 in the interval $0 < k_1 < 2$ we seek the optimal choice of k_2, $k_2^\star = f^\star(k_1)$ say, which minimises the variance of IP_2; subsequently we minimise the mean of IP_2 subject to constraint 4.2.1. Note that once the optimal policy $k_2^\star = f^\star(k_1)$ has been determined,

we need to set:

$$SP_2^\star = \frac{\mu(k_2^\star + 1)}{k_2^\star} - \sigma\Phi^{-1}(\delta)\Xi$$

where:

$$\Xi = \sqrt{\frac{k_1(2 - 5k_2^\star k_1 + (k_2^\star)^2 k_1 - (k_2^\star)^3 + (k_2^\star)^3 k_1 - k_1 + 3k_2^\star + 4k_2^\star k_1^2 - 2(k_2^\star)^2 k_1^2)}{k_2^\star(2 - k_2^\star)(k_1 + k_2^\star - k_2^\star k_1)(2 - k_1)}}$$

resulting in the constrained minimum $E[IP_2] = SP_2^\star - \frac{\mu}{k_2^\star}$. It has been assumed that the Customer demand parameters μ and σ are known or can be estimated accurately from the data.

Due to the high complexity of the expressions involved, we rely (in part) on Matlab's symbolic toolbox [oTC] to obtain an analytic expression for the optimal policy $k_2^\star = f^\star(k_1)$. The following procedure was followed: First, the variance of IP_2 was differentiated with respect to k_2 and the resulting expression was set to zero. This was then solved to express k_1 as a function of k_2, resulting in three (apparently complex) solutions:

$$k_2 = \frac{1}{6}\frac{\sqrt[3]{l} + 16/\sqrt[3]{l} + 6k_1 - 8}{k_1 - 1}$$

$$k_2 = -\frac{1}{12}\frac{\sqrt[3]{l} + 16/\sqrt[3]{l} - 12k_1 + 16 - i\sqrt{3}(\sqrt[3]{l} - 16/\sqrt[3]{l})}{k_1 - 1}$$

$$k_2 = -\frac{1}{12}\frac{\sqrt[3]{l} + 16/\sqrt[3]{l} - 12k_1 + 16 + i\sqrt{3}(\sqrt[3]{l} - 16/\sqrt[3]{l})}{k_1 - 1}$$

where

$$l = 108k_1^2 - 216k_1 + 64 + 12\sqrt{3k_1(k_1 - 2)(27k_1^2 - 54k_1 + 32)}$$

We investigate each of the following three solutions in turn. Consider first the term $m(k_1) = 27k_1^2 - 54k_1 + 32$ appearing inside the square root defining l. It is easy to see that this term is always positive for every value of k_1 (attaining its minimum

value $m(1) = 5$). Thus, the term inside the square root defining l is always negative in the interval $0 < k_1 < 2$. Hence l is complex and can be written in terms of its real and imaginary parts as:

$$l = 108k_1^2 - 216k_1 + 64 + 12i\sqrt{3k_1(2-k_1)(27k_1^2 - 54k_1 + 32)}$$

Thus parameter l can be written in polar form as $l = r\exp(i\phi)$ where

$$r = \sqrt{(108k_1^2 - 216k_1 + 64)^2 + 432k_1(2-k_1)(27k_1^2 - 54k_1 + 32)} = 64$$

and

$$\phi = \tan^{-1}\left(\frac{3\sqrt{3k_1(2-k_1)(27k_1^2 - 54k_1 + 32)}}{27k_1^2 - 54k_1 + 16}\right) \tag{4.2.2}$$

To avoid any confusion arising from the fact that $\tan^{-1}(\cdot)$ is a multi-function, we stress that ϕ is assumed to take values in the interval $0 \leq \phi < \pi$ (since the imaginary part of l is positive on the interval $0 < k_1 < 2$, while the real part of l takes positive, zero and negative values in this interval). On noting that:

$$l^{1/3} = \{4\exp(i\phi/3), 4\exp(i(\phi + 2\pi)/3), 4\exp(i(\phi - 2\pi)/3)\}$$

and substituting into the three expressions for k_2, it can be easily seen after some algebra, that each expression reduces to the same set of solutions, given by the three (real-valued) functions:

$$k_2 = \left\{\frac{4\cos(\phi/3) + 3k_1 - 4}{3(k_1 - 1)}, \frac{4\cos(\phi/3 - 2\pi/3) + 3k_1 - 4}{3(k_1 - 1)}, \frac{4\cos(\phi/3 + 2\pi/3) + 3k_1 - 4}{3(k_1 - 1)}\right\}$$

It may now be easily verified that the minimizing solution corresponds to the second function, so that

$$k_2^\star = f^\star(k_1) = \frac{4\cos(\phi/3 - 2\pi/3) + 3k_1 - 4}{3(k_1 - 1)}$$

where $\phi = \phi(k_1)$ is defined in equation 4.2.2. Again $f^\star(1)$ is not formally defined by this equation, so we set $f^\star(1) = 1$ to make the function continuous and differentiable at $k_1 = 1$. A plot of $k_2^\star = f^\star(k_1)$ (along with the boundary between

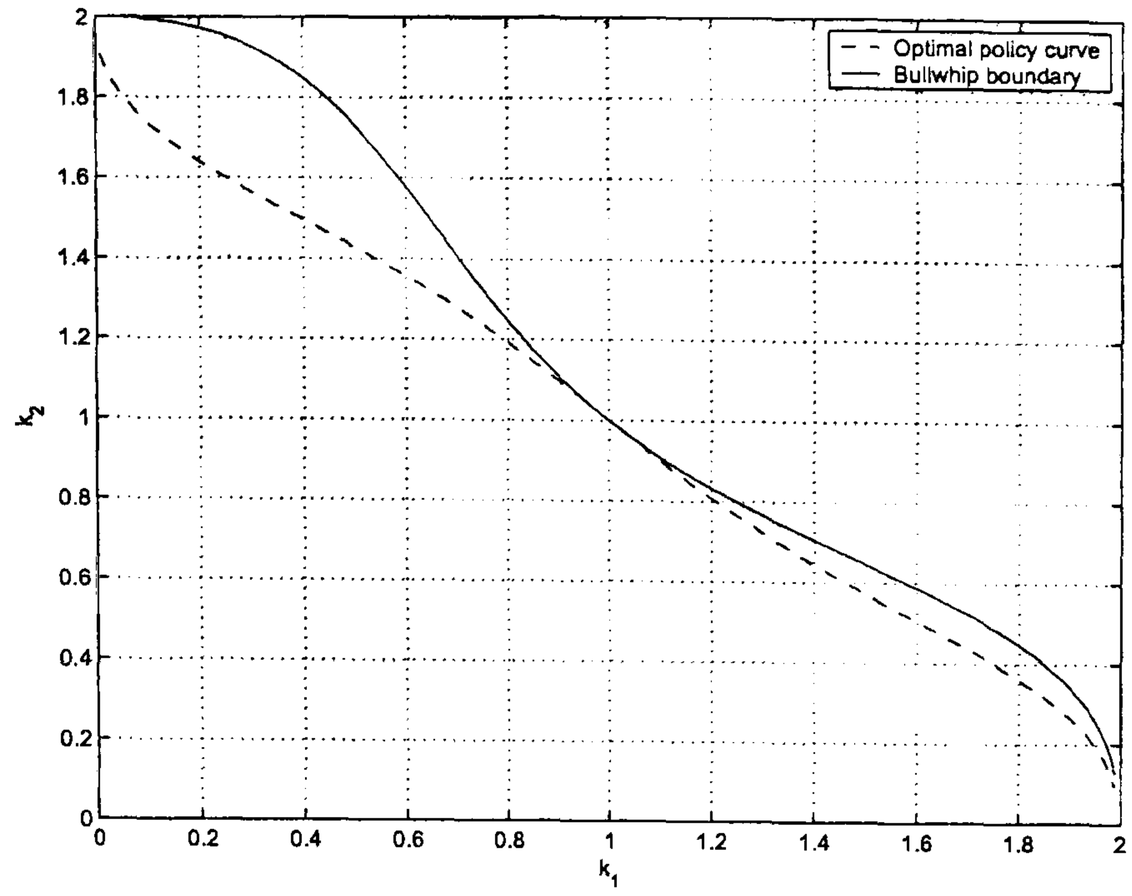

Figure 4.1: Optimal policy $k_2^\star = f^\star(k_1)$ and boundary between amplification and attenuation regions

the attenuation and amplification regions) is shown in Figure 4.1. An important observation is that the optimal curve lies entirely in the attenuation region. Thus, under information-sharing (disclosure of policy parameter k_1 to the Distributor), a 'selfish' policy by the Distributor (resulting from his attempt to minimise his own inventory costs) can not give rise to the bullwhip effect. Of course this conclusion should be qualified by the assumptions of the model (e.g., linearity, white-noise customer demand profile, no demand forecasting, etc).

The minimum variance $\mathrm{Var}^\star(IP_2)$ can be obtained by substituting the optimal policy $k_2^\star = f^\star(k_1)$ into the $(3,3)$ element of P_5. A plot of $\mathrm{Var}^\star(IP_2)$ versus k_1 reveals that $\mathrm{Var}^\star(IP_2)$ is a monotonically increasing function of k_1. The optimal curve $k_2^\star = f^\star(k_1)$ (which is a monotonically decreasing function) starts at point $(2,0)$ (where $\mathrm{Var}^\star(IP_2) = 0$), passes through the point $(1, k_2^\star(1) = 1)$ (where $\mathrm{Var}^\star(IP_2) = \sigma^2$) and approaches zero as $k_1 \to 2$ (where $\mathrm{Var}^\star(IP_2) \to \infty$ as this corresponds to the edge of the stability region).

Example: The results of the optimal policy $k_2^\star = f^\star(k_1)$ for three values of $k_1 = 0.5, 1, 1.5$ are summarised in Table 4.1. The parameters for the demand distribution were chosen as $\mu = 10$ and $\sigma = 1$, while parameter δ was set to $\delta = 0.05$. The distributions of IP_2 and $IP_2 - O_{1,2}$ for the three values of k_1 are

Table 4.1: Summary of optimal policy results

k_1	$k_2^\star$	$\mathrm{E}[IP_2]$	$\mathrm{Var}[IP_2]$	$\mathrm{E}[IP_2 - O_{1,2}]$	$\mathrm{Var}[IP_2 - O_{1,2}]$
0.50	1.43	11.41	0.26	1.41	0.73
1.00	1.00	12.33	1.00	2.33	2.00
1.50	0.57	14.23	2.38	4.23	6.61

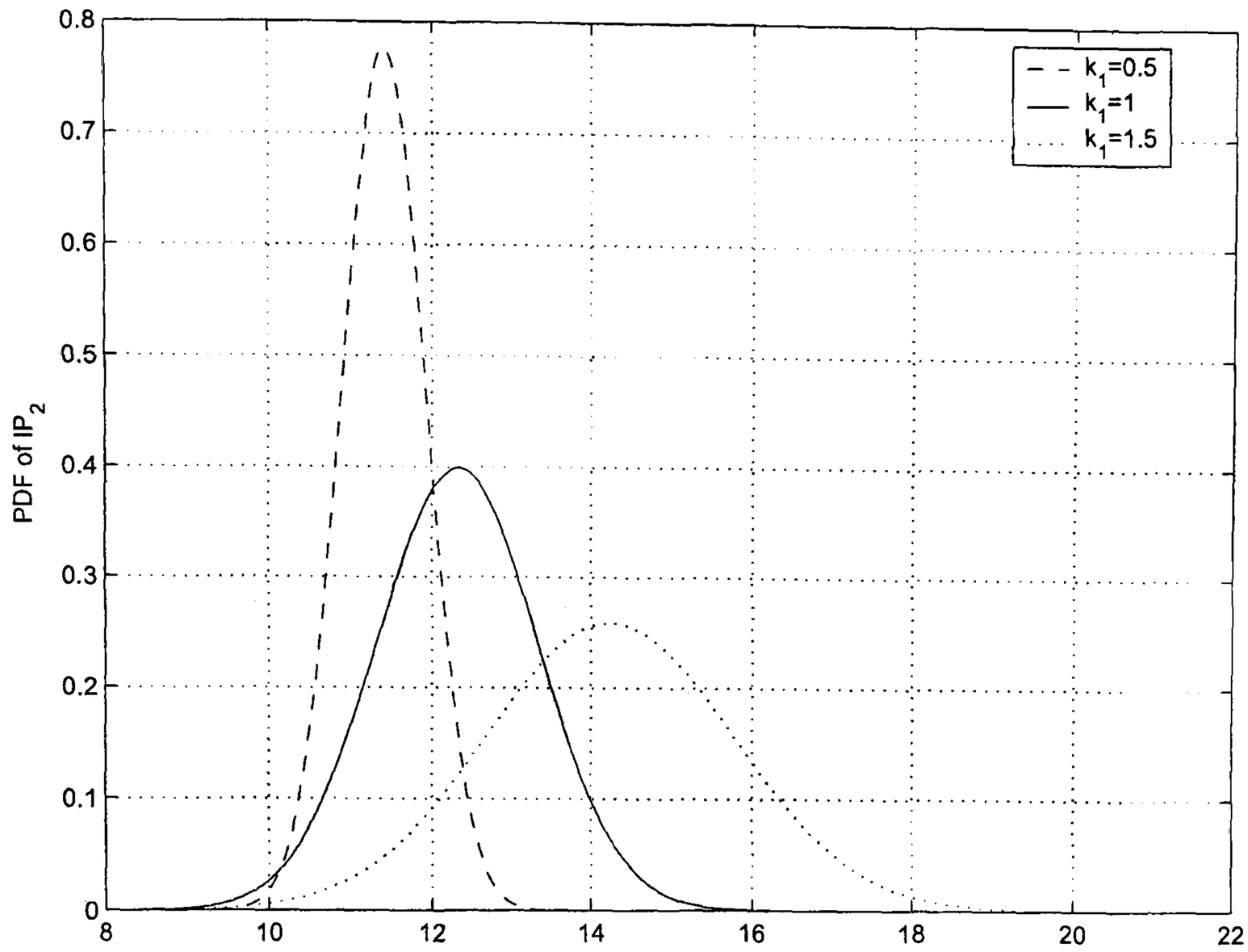

Figure 4.2: Probability density function of IP_2

shown in Figure 4.2 and Figure 4.3, respectively. Note that all three distributions of $IP_2 - O_{1,2}$ suggest that inventory IP_2 is insufficient to meet downstream demand $O_{1,2}$ with probability 0.05, as set by parameter δ.

4.3 Analysis of estimation schemes

We continue our analysis of the three node model by removing the assumption that policy parameter k_1 (corresponding to the Retailer's proportional replenishment policy) is communicated to the Distributor. A natural question arising in this case is whether k_1 can be estimated by the Distributor (node 2). Naturally, the data on

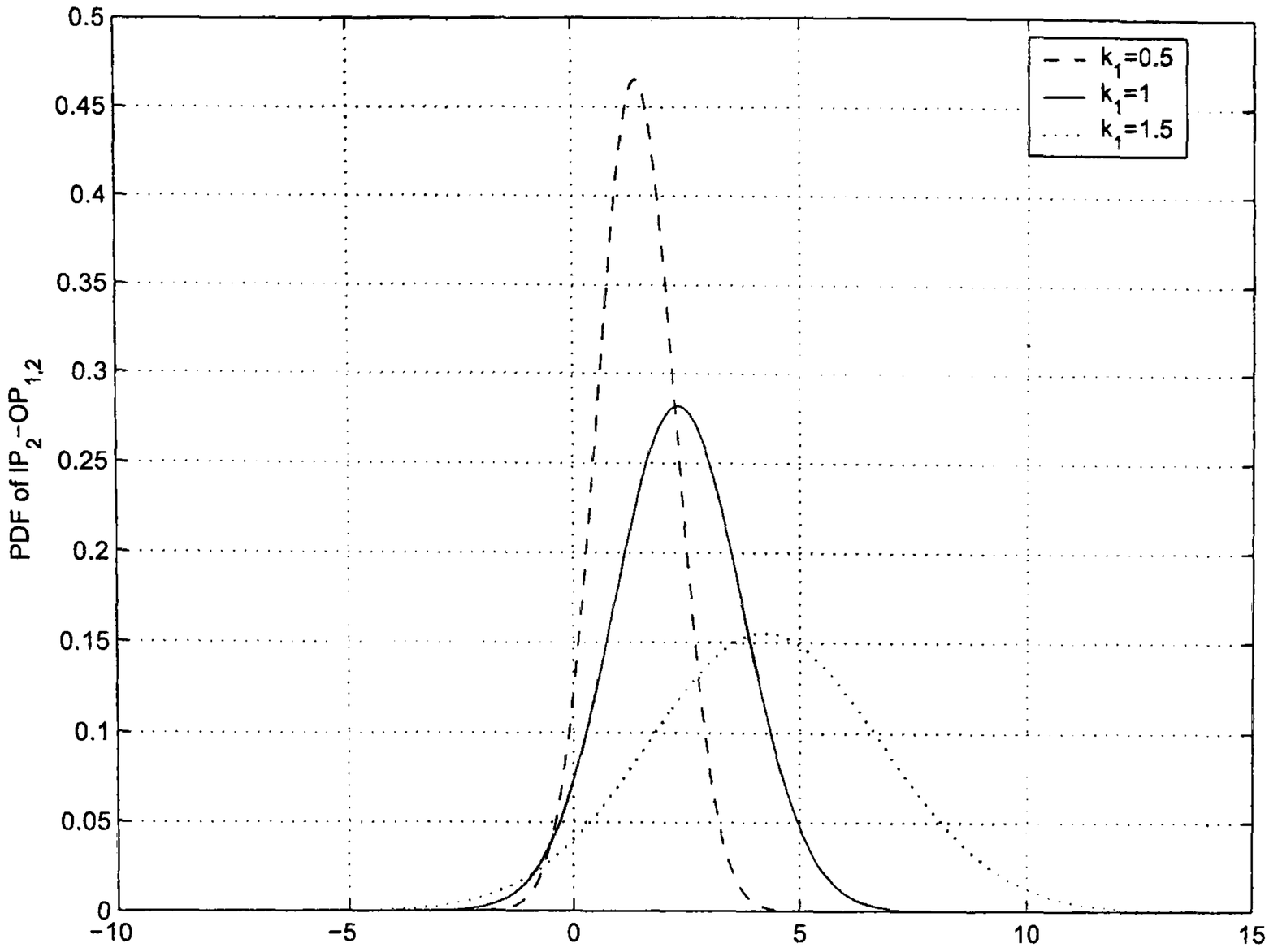

Figure 4.3: Probability density function of $IP_2 - O_{1,2}$

which the estimation should be based are restricted only to the input/output and state variables local to node 2.

4.3.1 Estimation method 1: Use of partial information derived by covariance

Since $E(Y_{2,1}) = \mu$, the mean customer demand (μ) can be estimated from $Y_{2,1}(t)$, which is an output signal of node 2 (e.g., an unbiased estimate $\hat{\mu}$ of μ can be obtained asymptotically). Consider next the part of the covariance matrix P_5 corresponding to the state variables of node 2; this is the diagonal block of P_5 corresponding to the third and fourth rows and columns, i.e.:

$$\mathrm{Cov}(IP_2, Y_{2,1}) = \sigma^2 \begin{pmatrix} \frac{k_1(2-k_1-k_2+k_1k_2)}{k_2(2-k_1)(2-k_2)(k_1+k_2-k_1k_2)} & \frac{k_1(k_1-1)}{(2-k_1)(k_1+k_2-k_1k_2)} \\ \frac{k_1(k_1-1)}{(2-k_1)(k_1+k_2-k_1k_2)} & \frac{k_1}{2-k_1} \end{pmatrix} := \begin{pmatrix} P_{11} & P_{12} \\ P_{12} & P_{22} \end{pmatrix}$$

One way of estimating k_1 and σ is to define:

$$\alpha = \frac{P_{12}}{P_{22}} = \frac{k_1 - 1}{k_1 + k_2 - k_1 k_2} \Rightarrow k_1 = \frac{1 + \alpha k_2}{1 + \alpha k_2 - \alpha}$$

and note that:

$$\sigma^2 = \frac{P_{22}(2-k_1)}{k_1}$$

Now, using the data $\{IP_2(t), Y_{2,1}(t)\}$ and noting that parameter k_2 is known, we can obtain estimates for $P_{11} = \mathrm{Var}(IP_2)$, $P_{22} = \mathrm{Var}(Y_{2,1})$ and $P_{12} = E[(IP_2 - E(IP_2))(Y_{21} - E(Y_{2,1})]$, say $\hat{P}_{11}$, $\hat{P}_{22}$ and $\hat{P}_{12}$ respectively, and use them to estimate k_1 and σ via equations:

$$\hat{\alpha} = \frac{\hat{P}_{12}}{\hat{P}_{22}}, \quad \hat{k}_1 = \frac{1+\hat{\alpha}k_2}{1+\hat{\alpha}(k_2-1)}, \quad \hat{\sigma}^2 = \frac{\hat{P}_{22}(2-\hat{k}_1)}{\hat{k}_1}$$

This estimation scheme will produce asymptotically unbiased estimates for k_1 and σ^2 and can implemented efficiently via the recursions (in n):

$$\begin{aligned}
\hat{P}_{22}(n) &= \frac{\hat{P}_{22}(n-1)}{1+(Y_{2,1}(n)-\overline{Y}_{2,1}(n))^2\hat{P}_{22}(n-1)} \\
\hat{\alpha}(n) &= \hat{\alpha}(n-1) + \hat{P}_{22}(n)(Y_{2,1}(n)-\overline{Y}_{1,2}(n))[(IP_2(n)-\overline{IP}_2(n)) - \\
&\qquad (Y_{2,1}(n)-\overline{Y}_{2,1}(n))\hat{\alpha}(n-1)] \\
\hat{k}_1(n) &= \frac{1+\hat{\alpha}(n)k_2}{1+\hat{\alpha}(n)(k_2-1)} \\
\hat{\sigma}^2(n) &= \frac{\hat{P}_{22}(n)(2-\hat{k}_1(n))}{\hat{k}_1(n)}
\end{aligned}$$

where $\overline{IP}_2(n)$ and $\overline{Y}_{2,1}(n)$ denote running estimates of the means of IP_2 and $Y_{2,1}$ respectively. The recursion can be initialised from arbitrary initial conditions $\hat{P}_{22}(0) > 0$ and $\hat{\alpha}(0)$.

4.3.2 Estimation method 2: Structured covariance approximation

A limitation of the first method is that it does not take full advantage of the available information structure (e.g., the information contained in $\mathrm{Var}(IP_2)$ is ignored). A superior approach is to formulate the estimation problem as a structured-covariance approximation, e.g.,

Table 4.2: Estimated and true parameters (method 1)

	$E[IP_1]$	$E[IP_2]$	$E[Y_{1,0}]$	$\mathrm{E}[Y_{2,1}]$	$E[Y_{3,2}]$	$\mathrm{Var}[Y_{3,2}]$	α	σ^2	k_1
Estimated	13.35	13.35	9.99	9.99	9.98	15.94	0.63	1.09	1.48
True	13.33	13.33	10.00	10.00	10.00	15.00	0.66	1.00	1.50

$$\min_{k_1\in(0,2),\sigma>0} \left\| W \circ \left(\hat{P} - \sigma^2 \begin{pmatrix} \frac{k_1(2-k_1-k_2+k_1k_2)}{k_2(2-k_1)(2-k_2)(k_1+k_2-k_1k_2)} & \frac{k_1(k_1-1)}{(2-k_1)(k_1+k_2-k_1k_2)} \\ \frac{k_1(k_1-1)}{(2-k_1)(k_1+k_2-k_1k_2)} & \frac{k_1}{2-k_1} \end{pmatrix} \right) \right\|_F^2$$

in which $\hat{P}$ denotes the estimated covariance matrix (constructed from the data). The choice of Frobenious-norm makes the problem easily transformable into a scalar sum-of-squares type non-linear optimisation, while W is a weighting matrix which can be used to emphasise/de-emphasise different matrix elements in the approximation (here '$\circ$' denotes the Hadamard product, i.e., element by element product, of two matrices [HJ95]). For example, choosing $W_{11} = W_{22} = 1$ and $W_{12} = W_{21} = \frac{1}{2}$ results in the objective function:

$$\left(\hat{P}_{11} - \frac{\sigma^2 k_1(2-k_1-k_2+k_1k_2)}{k_2(2-k_1)(2-k_2)(k_1+k_2-k_1k_2)}\right)^2 + \left(\hat{P}_{12} - \frac{\sigma^2 k_1(k_1-1)}{(2-k_1)(k_1+k_2-k_1k_2)}\right)^2 + \left(\hat{P}_{22} - \frac{\sigma^2 k_1}{2-k_1}\right)^2$$

which can be easily minimised (over $k_1 \in (0,2)$ and $\sigma^2 > 0$) via gridding or local search methods.

Example: We illustrate the estimation scheme by means of a simulation example. Assume that $O_{1,2} \sim N(\mu, \sigma^2)$ with $\mu = 10$ and $\sigma^2 = 1$. We simulate the 3-node chain with parameters $k_1 = k_2 = 1.5$, $SP_1 = SP_2 = 20$ and $IP_1(0) = IP_2(0) = 20$ for $n = 1000$ time-steps. Parameter k_1 is assumed unknown to node 2 (Distributor) and is estimated using the first method described earlier. The results of the estimation are summarised in Table 4.2.

Applying the second estimation method described in this section (structured covariance approximation) produced a (slightly) more accurate estimate $\hat{k}_1 = 1.51$. The minimisation was carried over k_1 using the estimated variance of the end-customer demand signal $\hat{\sigma}^2 = 1.09$. The graph of the cost function which is minimised is shown in Figure 4.4. The minimum was found to be insensitive to the choice of norm (Frobenious or maximum singular value) and weighting function W.

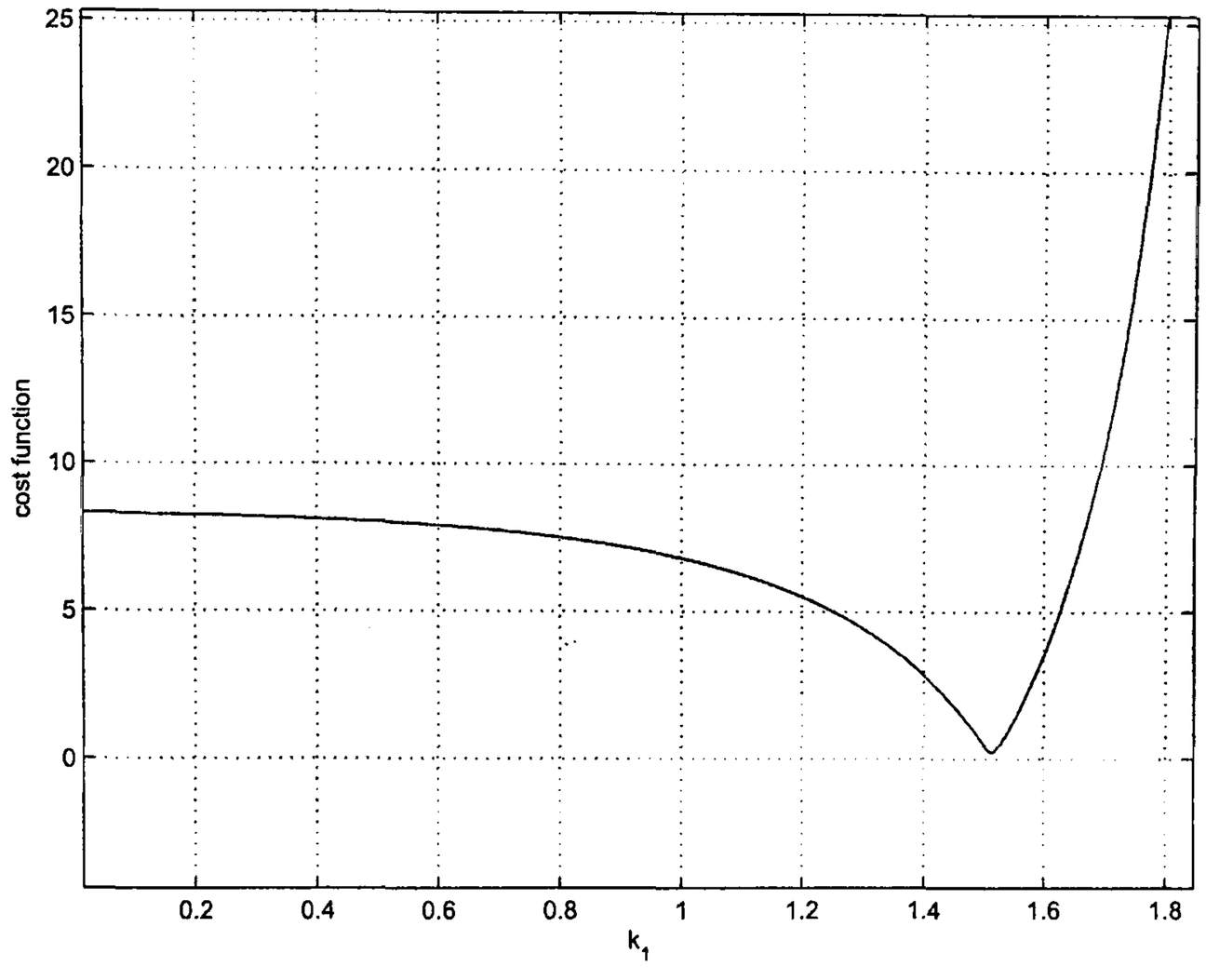

Figure 4.4: Cost function in covariance structured approximation (method 2)

The main advantage of using the (computationally more demanding) covariance structured approximation method (method 2) for estimating k_1 is illustrated in Figure 4.5. This shows how the estimates of k_1 for the two schemes vary with the length of the data records (note that only method 1 is truly recursive). It can be seen from Figure 4.5 that the estimates based on method 2 converge much faster to the true parameter value $k_1 = 1.5$. This was consistently observed in all simulations and is not surprising as the full structure of the covariance matrix is used.

Once an accurate estimate of k_1 has been obtained, node 2 can switch to the optimal policy $k_2^\star$ and $IP_2^\star$, thus minimising its average inventory level and its inventory fluctuations. To assess the 'value of information' when policy parameter k_1 is disclosed to the Distributor, it is necessary to carry out a statistical analysis of

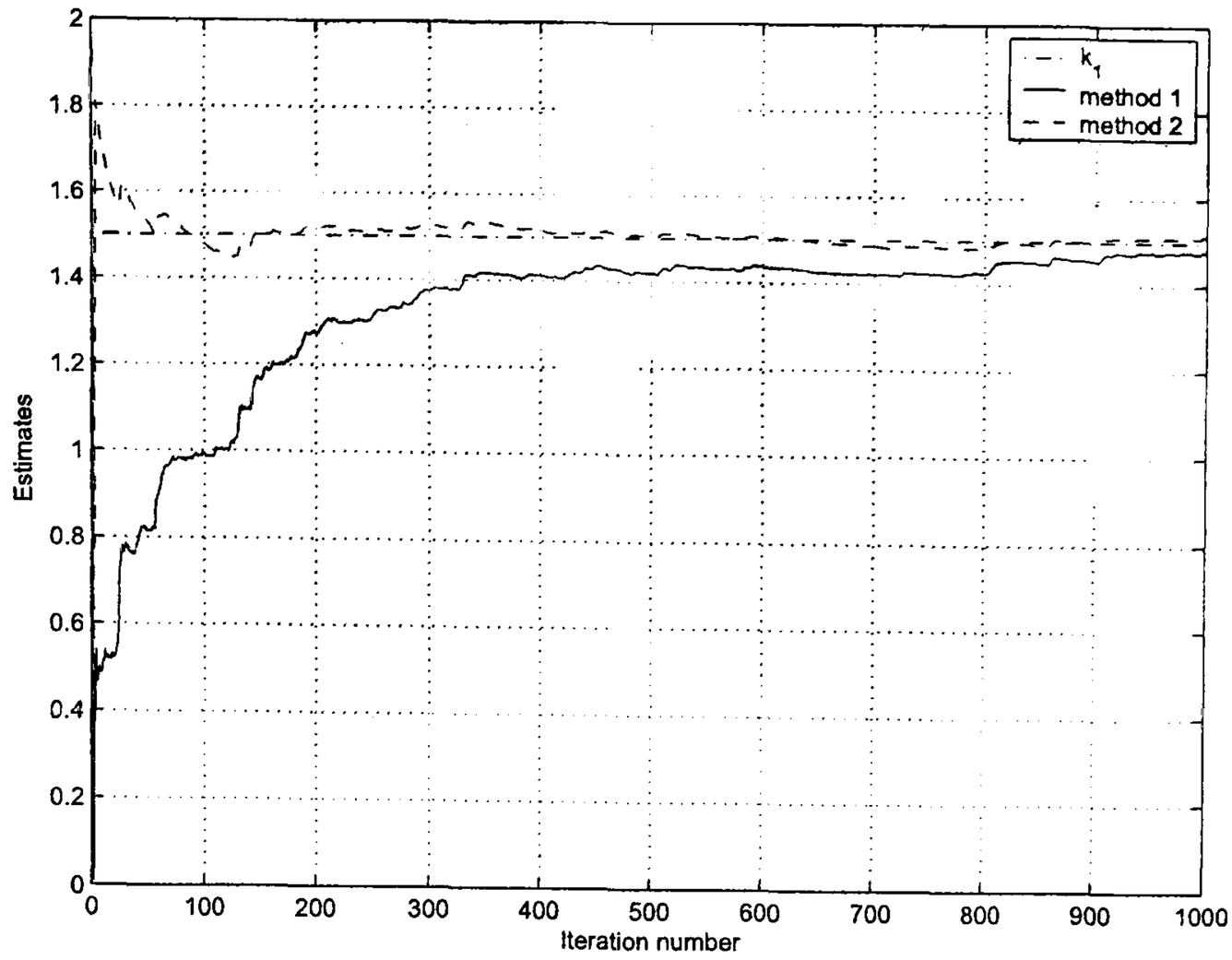

Figure 4.5: Estimates of k_1 using method 1 and 2 as functions of data length

the estimation schemes presented above in order to determine the properties of the estimate (e.g., variance, confidence intervals, rate of convergence etc) and how these depend on data lengths. This analysis is not undertaken here and will be addressed in future work.

4.3.3 Estimation method 3: Use of covariance matrix structure and its properties

Another interesting approach towards better estimation schemes involves the use of covariance matrix structure and its properties. The following study is focused on the computation of inverse covariance matrix and the estimation of the inverse gain factor $\check{k_1} = \frac{1}{k_1}$. By considering as $\acute{P_5}$ the part of the covariance matrix P_5 corresponding to the state variable of node 2, we have:

$$\acute{P_5} = \begin{pmatrix} \frac{k_1(2-k_1-k_2+k_1k_2)}{k_2(2-k_1)(2-k_2)(k_1+k_2-k_1k_2)} & \frac{k_1(k_1-1)}{(2-k_1)(k_1+k_2-k_1k_2)} \\ \frac{k_1(k_1-1)}{(2-k_1)(k_1+k_2-k_1k_2)} & \frac{k_1}{2-k_1} \end{pmatrix} := \begin{pmatrix} \acute{P_{533}} & \acute{P_{534}} \\ \acute{P_{543}} & \acute{P_{544}} \end{pmatrix}$$

The inverse covariance matrix $\acute{P_5}^{-1}$ is given by:

$$\acute{P_5}^{-1} = \begin{pmatrix} \frac{k_2(2-k_2)(k_1k_2-k_1-k_2)^2}{k_1^2} & \frac{(k_1-1)(k_1k_2-k_1-k_2)k_2(2-k_2)}{k_1^2} \\ \\ \frac{(k_1-1)(k_1k_2-k_1-k_2)k_2(2-k_2)}{k_1^2} & \_\frac{(2+k_1k_2-k_1-k_2)(k_1k_2-k_1-k_2)}{k_1^2} \end{pmatrix} := \begin{pmatrix} \acute{P}_{533}^{-1} & \acute{P}_{534}^{-1} \\ \acute{P}_{543}^{-1} & \acute{P}_{544}^{-1} \end{pmatrix}$$

$\acute{P}_{533}^{-1}$ can be rewritten as:

$$\begin{aligned} \acute{P}_{533}^{-1} &= k_2(2-k_2)\left[\frac{k_1k_2-k_1-k_2}{k_1}\right]^2 \\ &= k_2(2-k_2)\left[k_2-1-\frac{k_2}{k_1}\right]^2 \\ &= k_2(2-k_2)\left[(k_2-1)^2-2(k_2-1)\frac{k_2}{k_1}+\frac{k_2^2}{k_1^2}\right] \end{aligned}$$

By replacing $\check{k_1} = \frac{1}{k_1}$, $\acute{P}_{533}^{-1}$ can be written:

$$\begin{aligned} \acute{P}_{533}^{-1} &= k_2(2-k_2)\left[(k_2-1)^2-2(k_2-1)k_2\check{k_1}+k_2^2\check{k_1}^2\right] \\ &= k_2(2-k_2)(k_2-1)^2-2k_2^2(2-k_2)(k_2-1)\check{k_1}+k_2^3(2-k_2)\check{k_1}^2 \end{aligned}$$

It can be inferred from above expression that $\acute{P}_{533}^{-1}$ is a quadratic polynomial in $\check{k_1}$ with quadratic coefficient $\alpha_{533} = k_2^3(2-k_2)$, linear coefficient $\beta_{533} = -2k_2^2(2-k_2)(k_2-1)$ and constant coefficient $\gamma_{533} = k_2(2-k_2)(k_2-1)^2$. Thus, the resulting quadratic equation can be expressed as: $\acute{P}_{533}^{-1} = \alpha_{533}\check{k_1}^2 + \beta_{533}\check{k_1} + \gamma_{533}$.

Similarly, $\acute{P}_{534}^{-1}$ and $\acute{P}_{543}^{-1}$ are given by:

$$
\begin{aligned}
P_{534}^{'-1} = P_{543}^{'-1} &= \frac{(k_1-1)(k_1k_2-k_1-k_2)k_2(2-k_2)}{k_1^2} \\
&= (\frac{k_1-1}{k_1})(\frac{k_1k_2-k_1-k_2}{k_1})k_2(2-k_2) \\
&= (1-\frac{1}{k_1})(k_2-1-k_2\frac{1}{k_1})k_2(2-k_2) \\
&= (1-\check{k}_1)(k_2-1-k_2\check{k}_1)k_2(2-k_2) \\
&= k_2(2-k_2)\left[k_2-1-2k_2\check{k}_1+\check{k}_1+k_2\check{k}_1^2\right] \\
&= k_2(2-k_2)\left[(k_2-1)+(1-2k_2)\check{k}_1+k_2\check{k}_1^2\right] \\
&= k_2(2-k_2)(k_2-1)+k_2(2-k_2)(1-2k_2)\check{k}_1+k_2^2(2-k_2)\check{k}_1^2
\end{aligned}
$$

Again, $P_{534}^{'-1}$ and $P_{543}^{'-1}$ are quadratic polynomials in $\check{k}_1$ with coefficients $\alpha_{534} = \alpha_{543} = k_2^2(2-k_2)$, $\beta_{534} = \beta_{543} = k_2(2-k_2)(1-2k_2)$ and $\gamma_{534} = \gamma_{543} = k_2(2-k_2)(k_2-1)$. Therefore, the resulting quadratic equations can be expressed as: $P_{534}^{'-1} = \alpha_{534}\check{k}_1^2 + \beta_{534}\check{k}_1 + \gamma_{534}$ and $P_{543}^{'-1} = \alpha_{543}\check{k}_1^2 + \beta_{543}\check{k}_1 + \gamma_{543}$.

Finally, $P_{544}^{'-1}$ can be derived as:

$$
\begin{aligned}
P_{544}^{'-1} &= -\frac{(2+k_1k_2-k_1k_2)(k_1k_2-k_1-k_2)}{k_1^2} \\
&= (\frac{2+k_1k_2-k_1-k_2}{k_1^2})(-\frac{k_1k_2-k_1-k_2}{k_1}) \\
&= (2\check{k}_1+k_2-1-k_2\check{k}_1)(-k_2+1+k_2\check{k}_1) \\
&= \left[k_2-1+\check{k}_1(2-k_2)\right]\left[-(k_2-1)+k_2\check{k}_1\right] \\
&= -(k_2-1)^2+\left[-(2-k_2)(k_2-1)+k_2(k_2-1)\right]\check{k}_1+k_2(2-k_2)\check{k}_1^2 \\
&= -(k_2-1)^2+(k_2-1)(2k_2-2)\check{k}_1+k_2(2-k_2)\check{k}_1^2
\end{aligned}
$$

$\acute{P}_{544}^{-1}$ can also be written in the following quadratic form:

$$\acute{P}_{544}^{-1} = \alpha_{544}\check{k}_1^2 + \beta_{544}\check{k}_1 + \gamma_{544}$$

where $\alpha_{544} = k_2(2-k_2)$, $\beta_{544} = (k_2-1)(2-k_2)$ and $\gamma_{544} = -(k_2-1)^2$.

The above results can be summarised in the following Proposition 4.3.1.

Proposition 4.3.1. *The inverse of partial covariance matrix* $\acute{P}_5$*, denoted as* $\acute{P}_5{}^{-1}$*, is quadratic in* $\check{k}_1$*. In particular,* $\acute{P}_5{}^{-1} = \mathbf{A}\check{k}_1{}^2 + \mathbf{B}\check{k}_1 + \mathbf{\Gamma}$ *where:*

$$\mathbf{A} = \begin{bmatrix} \alpha_{533} & \alpha_{534} \\ \alpha_{543} & \alpha_{544} \end{bmatrix} = \begin{bmatrix} k_2^3(2-k_2) & k_2^2(2-k_2) \\ k_2^2(2-k_2) & k_2(2-k_2) \end{bmatrix}$$

$$\mathbf{B} = \begin{bmatrix} \beta_{533} & \beta_{534} \\ \beta_{543} & \beta_{544} \end{bmatrix} = \begin{bmatrix} -2k_2^2(2-k_2)(k_2-1) & k_2(2-k_2)(1-2k_2) \\ k_2(2-k_2)(1-2k_2) & (k_2-1)(2-k_2) \end{bmatrix}$$

and

$$\mathbf{\Gamma} = \begin{bmatrix} \gamma_{533} & \gamma_{534} \\ \gamma_{543} & \gamma_{544} \end{bmatrix} = \begin{bmatrix} k_2(2-k_2)(k_2-1)^2 & k_2(2-k_2)(k_2-1) \\ k_2(2-k_2)(k_2-1) & -(k_2-1)^2 \end{bmatrix}$$

where $\check{k}_1 = \frac{1}{k_1}$.

The result of Proposition 4.3.1 is that we can formulate the estimation problem as a structured inverse covariance approximation which can be expressed by the minimisation of Frobenious norm of the inverse covariance matrices. As it is mentioned previously the main goal of the participant in node 2 is to estimate the inverse factor of downstream node, $1/k_1$. Since k_2 is a known parameter we can consider the constant matrix $\mathbf{\Gamma}$ as a known quantity. In addition to this we can define as $\mathcal{D}$, a new constant matrix given by: $\mathcal{D} = (\mathbf{\Gamma} - \check{\acute{P}}_5^{-1})$ where $\check{\acute{P}}_5^{-1}$ is the estimation of the inverse matrix $\acute{P}_5{}^{-1}$. Since $\acute{P}_5{}^{-1} = \mathbf{A}\check{k}_1{}^2 + \mathbf{B}\check{k}_1 + \mathbf{\Gamma}$, we have:

$$\begin{aligned} \check{\acute{P}}_5^{-1} - \acute{P}_5{}^{-1} &= \mathbf{A}\check{k}_1{}^2 + \mathbf{B}\check{k}_1 + (\mathbf{\Gamma} - \check{\acute{P}}_5^{-1}) \\ \check{\acute{P}}_5^{-1} - \acute{P}_5{}^{-1} &= \mathbf{A}\check{k}_1{}^2 + \mathbf{B}\check{k}_1 + \mathcal{D} \end{aligned}$$

This observation leads to the following Lemma 4.3.2:

Lemma 4.3.2. *Define the following optimisation problem:*

$$\gamma_{\circ} = \min_{\check{k_1} \in [\frac{1}{2}, \infty)} \| \check{\acute{P}}_5^{-1} - \acute{P}_5^{-1} \|_F^2$$

The optimal solution is given by the following expression:

$$\gamma_{\circ} = \min_{\check{k_1} \in [\frac{1}{2}, \infty)} \{\alpha_0 + \alpha_1 \check{k_1} + \alpha_2 \check{k_1}^2 + \alpha_3 \check{k_1}^3 + \alpha_4 \check{k_1}^4\}.$$

where

$$\begin{aligned} \alpha_0 &= trace(\mathcal{D}\mathcal{D}'), \\ \alpha_1 &= -trace(\mathcal{D}\mathbf{B}' + \mathbf{B}\mathcal{D}'), \\ \alpha_2 &= trace(\mathbf{B}\mathbf{B}' - \mathcal{D}\mathbf{A}' - \mathbf{A}\mathcal{D}'), \\ \alpha_3 &= trace(\mathbf{B}\mathbf{A}' + \mathbf{A}\mathbf{B}') \\ \alpha_4 &= trace(\mathbf{A}\mathbf{A}') \end{aligned}$$

Proof. See Appendix A. □

Let us now consider each of this coefficient.

$$
\begin{aligned}
\alpha_0 &= [\mathbf{\Phi_{11}} - k_2(2-k_2)(k_2-1)^2]^2 + 2[\mathbf{\Phi_{12}} - k_2(2-k_2)(k_2-1)]^2 + [\mathbf{\Phi_{22}} + (k_2-1)^2]^2 \\
\alpha_1 &= 4[\mathbf{\Phi_{11}} - k_2(2-k_2)(k_2-1)^2]k_2^2(2-k_2)(k_2-1) + \\
&\quad 4[\mathbf{\Phi_{12}} - k_2(2-k_2)(k_2-1)]k_2(2-k_2)(2k_2-1) - 4[\mathbf{\Phi_{22}} + (k_2-1)^2](k_2-1)^2 \\
\alpha_2 &= 4k_2^2(2-k_2)^2(k_2-1)^2 + 2k_2^2(2-k_2)^2(2k_2-1)2 - \\
&\quad 2[\mathbf{\Phi_{11}} - k_2(2-k_2)(k_2-1)^2]k_2^3(2-k_1) - 4[\mathbf{\Phi_{12}} - k_2(2-k_2)(k_2-1)]k_2^2(2-k_2) + \\
&\quad 4(k_2-1)^4 - 2[\mathbf{\Phi_{22}} - (k_2-1)^2]k_2(2-k_2) \\
\alpha_3 &= -4k_2^5(2-k_2)^2(k_2-1) - 4k_2^3(2-k_2)^2(2k_2-1) + 4k_2(2-k_2)(k_2-1)^2 \\
\alpha_4 &= k_2^6(2-k_2)^2 - 2k_2^4(2-k_2)^2 + k_2^2(2-k_2)^2
\end{aligned}
$$

Next we consider the polynomial $p(\check{k_1}) = \alpha_0 + \alpha_1\check{k_1} + \alpha_2\check{k_1}^2 + \alpha_3\check{k_1}^3 + \alpha_4\check{k_1}^4$. The minimisation problem now involves simply the computation of the first derivative $p'(\check{k_1})$ which must be set equal to 0. Hence,

$$p'(\check{k_1}) = \alpha_1 + 2\alpha_2\check{k_1} + 3\alpha_3\check{k_1}^2 + 4\alpha_4\check{k_1}^3 = 0 \tag{4.3.1}$$

The solution of the above equation gives 3 roots from which we must choose the real root in the interval $\frac{1}{2} \leq \frac{1}{k_1} < \infty$, since $0 \leq k_1 \leq 2$.

Example: We demonstrate the structured inverse covariance estimation method with the aid of a simulation example. We consider the three node supply chain with the same parameters we used in previous examples for estimation. Thus, we set $k_1 = k_2 = 1.5$, $SP_1 = SP_2 = 20$, $IP_1(0) = IP_2(0) = 10$. We run the simulation for $n = 1000$ time-steps. As in previous approximation techniques k_1 is assumed as unknown parameter to node 2 (Distributor).

The solution of the cubic equation has given as expected 3 roots; 2 complex $r_1 = 0.2855 + 0.3884i$ and $r_2 = 0.2855 - 0.3884i$ and 1 real root $r_3 = 0.6657$ which corresponds to the minimum value $\check{k_1} = \frac{1}{k_1}$ of the cubic equation 4.3.1. $\frac{1}{r_3}$ gives the estimated parameter $\check{k_1}$ and it is equal to 1.5021 a value which is very close to

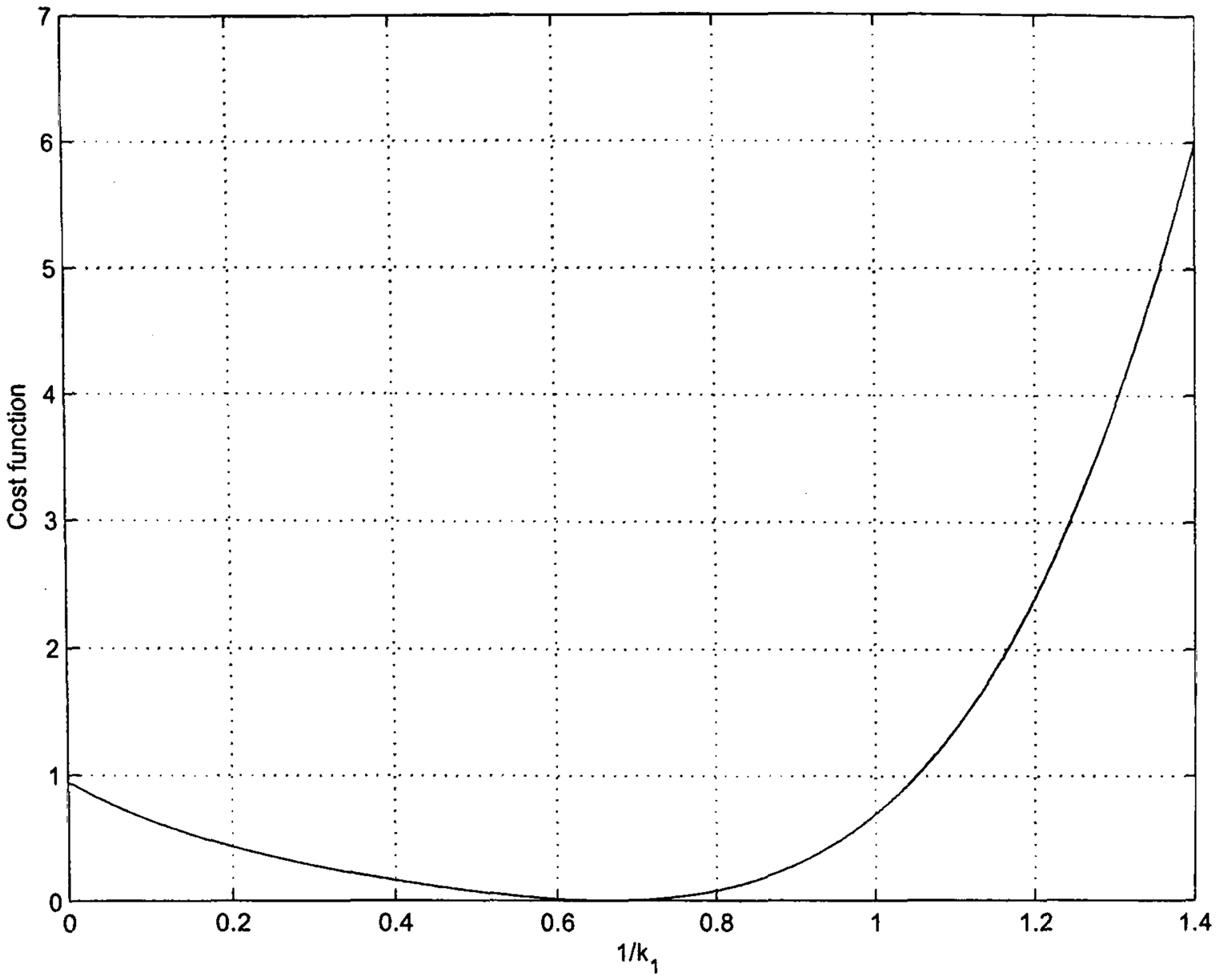

Figure 4.6: Cost function in inverse covariance structured estimation

the real simulation parameter $k_1 = 1.5$. Figure 4.6 illustrates the minimised cost function in inverse covariance structured estimation based on data of the previous example where $\breve{k_1} = 1.5021$.

Chapter 5

Modelling supply chains using Coloured Petri nets

Most of simulation tools are designed as interactive tools to be used by a human planner not as real time decision-making tools, which are directly linked to control system to dispatch tasks. Simulation tools aid human planner to make a right decision by providing information. However, human planner should be able to interpret and modify the plan in order to achieve better supply chain performances.

A different aspect of analysis based on the same type of supply chain studied in previous chapters can be also achieved by using timed Hierarchical Coloured Petri Nets (HCPN). This approach considers supply chain as an event-driven system and captures the process of flow of goods and information within the series supply chain. A HCPN model has been constructed to study the bullwhip effect in supply chains where individual parts using different inventory policies, while metrics have been defined for analysis of inventory balance and flow process. The proposed HCPN model allows to study the bullwhip effect even in cases when intermediate participants do not have sufficient inventories to fulfill downstream orders. In contrast to linear model developed in previous chapter HCPN provide useful framework to study the impact of backlogged orders in overall supply chain performance in non-continuous inventory replenishment policies and known forecasting methods followed by supply chain participants.

The dynamics of a supply chain system are modelled by firing rules, which define the flow of the tokens. In this chapter CPN-Tools [fCPN] are used for the design of

decision-making processes and simulation results are presented to highlight the main issues arising in real systems and to provide insights for future work on modelling and simulation of supply chains.

5.1 Description of the supply chain model

We consider again a series five-node supply chain depicted in Figure 5.1 consists of a Manufacturer, Distributor, Supplier, Retailer, and an end Customer site. We also assume that there is a single participant in each node. The main characteristics and dynamics of the series supply chain is based on the model presented in the chapter 3. Note that since the customer site is the actual output and input of the supply chain system the HCPN model consists of four individual stages. The direction of the flow of information (orders) and products in supply chain is shown in Figure 5.1. We assume that each intermediate participant makes decisions locally and place the amount of orders to the upstream level following an inventory control policy. Meanwhile, dispatch of products to downstream levels performed by upstream stages depends also on several rules and conditions (i.e., availability). We assume at the beginning of process that participants follow the same inventory control and dispatching policy although the model permits the use of different rules on the different stages. We assume also that there is no delay for the information transmission and processing between participants, in contrast with the delivery of finished products, where a lead-time delay is associated in each dispatching process.

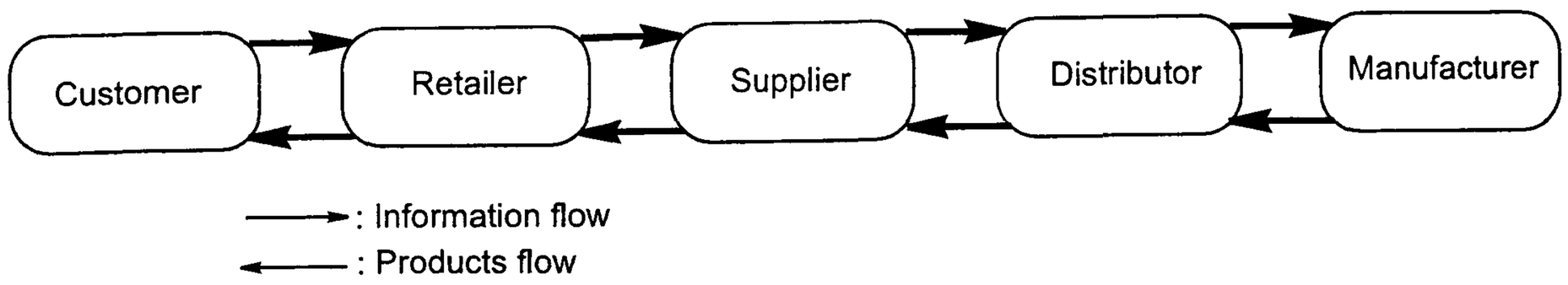

Figure 5.1: A series five-node supply chain

The attainment of all these processes can be implemented if we simply perceive that each action such as placing an order, delivery of product, or even inventory decision making is a discrete event with dynamic properties. Since the number of reachable states in supply chain is typically very large we need to describe clearly all the activities associated with events. This description helps also the user to understand better the structure, rules and functions used in Petri network.

Every time an order placed in node participant, inventory manager checks whether the amount of products held in stock is sufficient to fulfill the order requirements. We assume that there is an initial stock in all four stages of the supply chain. Incoming orders are served always directly from the stock which contains the current stock plus the amount of products that might have been received from the upstream level. If the inventory stores enough products, the same amount of ordering is sent to the upstream level, since manager tries to maintain the inventory to a target level. If the stock level is below the amount of products have been ordered then the completion of the total amount of order is not possible. In this case the managers can follow different inventory control policies. In our model we assume initially that manager orders the amount of products have not been delivered to the downstream level plus the amount of products that will be sufficient to fulfil the next order (backlogged orders).

A manager also decides the amount of products will be delivered to the downstream level. When a participant receives an order, manager checks if the inventory stores the needed products. If the stock has sufficient amount of products, manager dispatches the total amount has been instructed to deliver. Otherwise, if the stock is not sufficient, manager can either wait for needed product to be received and then send as a whole, or dispatch the incomplete order, and later the overdue additional quantity. In current work we consider that all the stages deliver even incomplete orders. This situation is more realistic in supply chains with delivery of strongly standardised products. Later we will see that this effect is seized in the model by cost functions discussed in chapter 2, in which both stock and flow of

products and information are accumulated.

The demand behaviour of the final customer shows fluctuations and has a normal distribution process with a given variance and mean. Thus, adjustment effects can be observed and examined with permanent changes of demand. On the other hand adjustment-conditioned deviations from the normal distribution process can be implemented very easily and are recognised immediately. This helps the modeler to see the effect of demand variance in all the individual stages of supply chain, and to monitor better the inventory levels after a single event has been occurred. In addition, the model permits other arbitrary demand processes (i.e. seasonal fluctuations) by modification of input data. The determination of the order quantities which affect decision making in all four stages of the supply chain leading to bullwhip effect, is easy and we can use metrics and graphical representation of backlogged orders to illustrate this phenomenon.

5.2 Description of the Hierarchical Coloured Petri Net

5.2.1 Prime page *Supply chain*

Figure 5.2 shows the HCPN model of the four-level supply chain. This abstract single prime page called *Supply chain* illustrates the highest network level and provides an overview of the supply chain network. (Note that the name of each page is displayed on the top left side). Figure 5.2 also shows how HCPN has been hierarchically constructed into four modules (subnets): The *manufacturer (M)*, the *Distributor (S1)*, the (intermediate) *supplier (S2)*, and the *Retailer (S3)*. The subnets of the model are also referred to as "pages", while each submodel node represents a page of the HCPN model. *Supply chain* page has four transitions which all are substitution transitions. The Customer site is represented in the prime page by two different socket places: *Cg* which is associated with the goods received by customers and

Cd which represents customer demand. *Cg* and *Cd* are the model output and input, respectively. There are also four socket places surrounding all substitution transitions. For instance, page *Supplier* has the following socket places as it can been inferred by Figure 5.2: *23g* (goods dispatched to *Retailer*), *12g* (goods delivered by *Distributor*), *32d* (demand order received from *Retailer*), and *21d* (demand order placed to Distributor). The hierarchical CP-net model is common for all three different ordering policies (aggresive ordering (AO), moving average (MA) and exponential smoothing (ES) forecasting techniques) considered in this chapter.

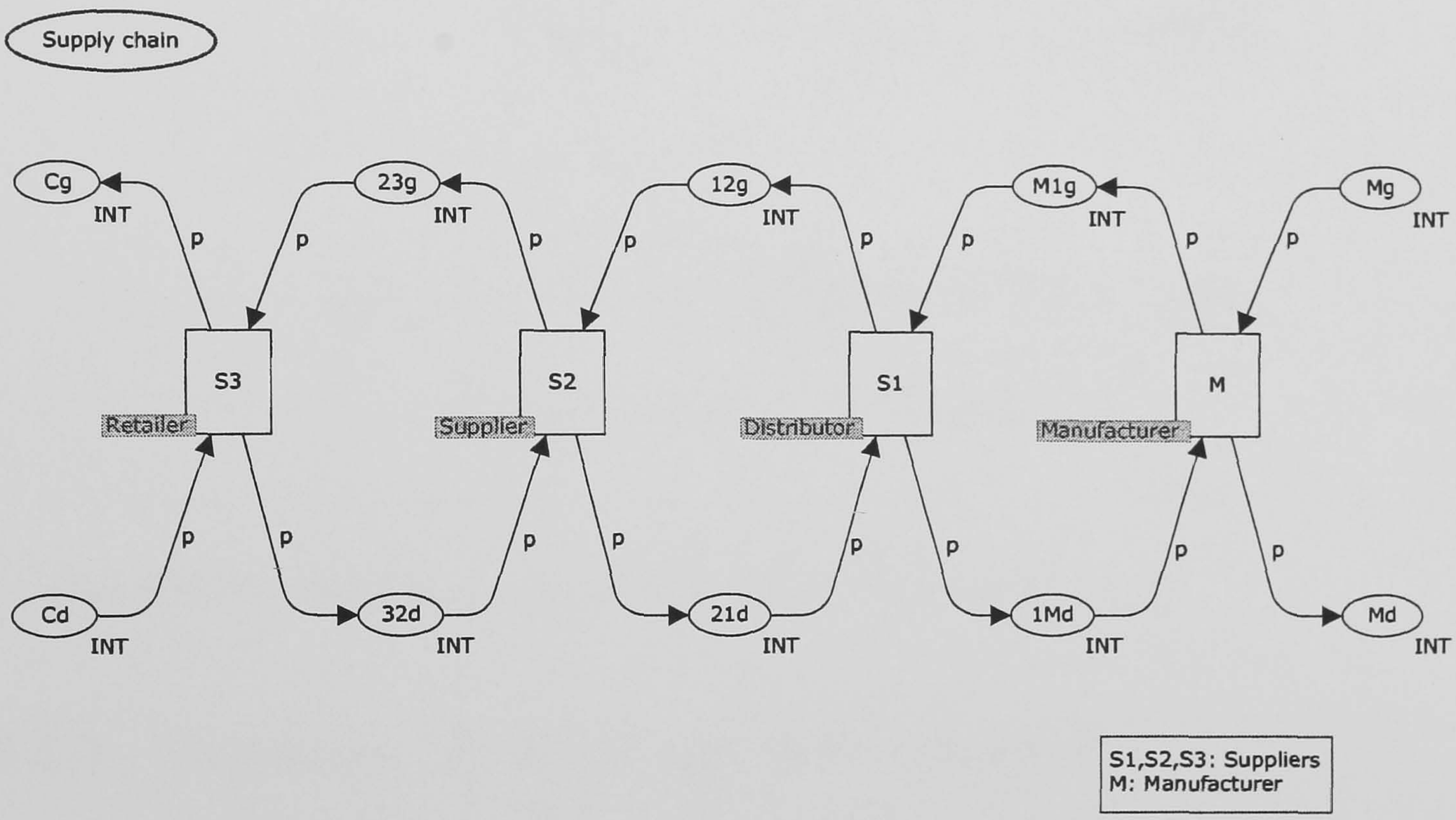

Figure 5.2: HCPN describes an overview of supply chain with four different nodes

The three modules *Retailer*, *Supplier* and *Distributor*, are structurally identical for each case - and their description is based on the description of *Supplier* module given later on, while *Manufacturer* has a different structure and it is described severally. Tokens in HCPN model are of type integers and are associated with the amount of orders and products flow within the supply chain. We can use more data types or more token colours such as names of products, staff or transportation means (could be represented by strings), or even prices and barcodes. However, for the purposes of analysing the impact of bullwhip effect and different replenishment

policies and forecasting techniques, are sufficient parameters for this analysis.

Figures 5.3 - 5.6 depict the four subnets: *Retailer*, *Supplier*, *Distributor*, *Manufacturer*.

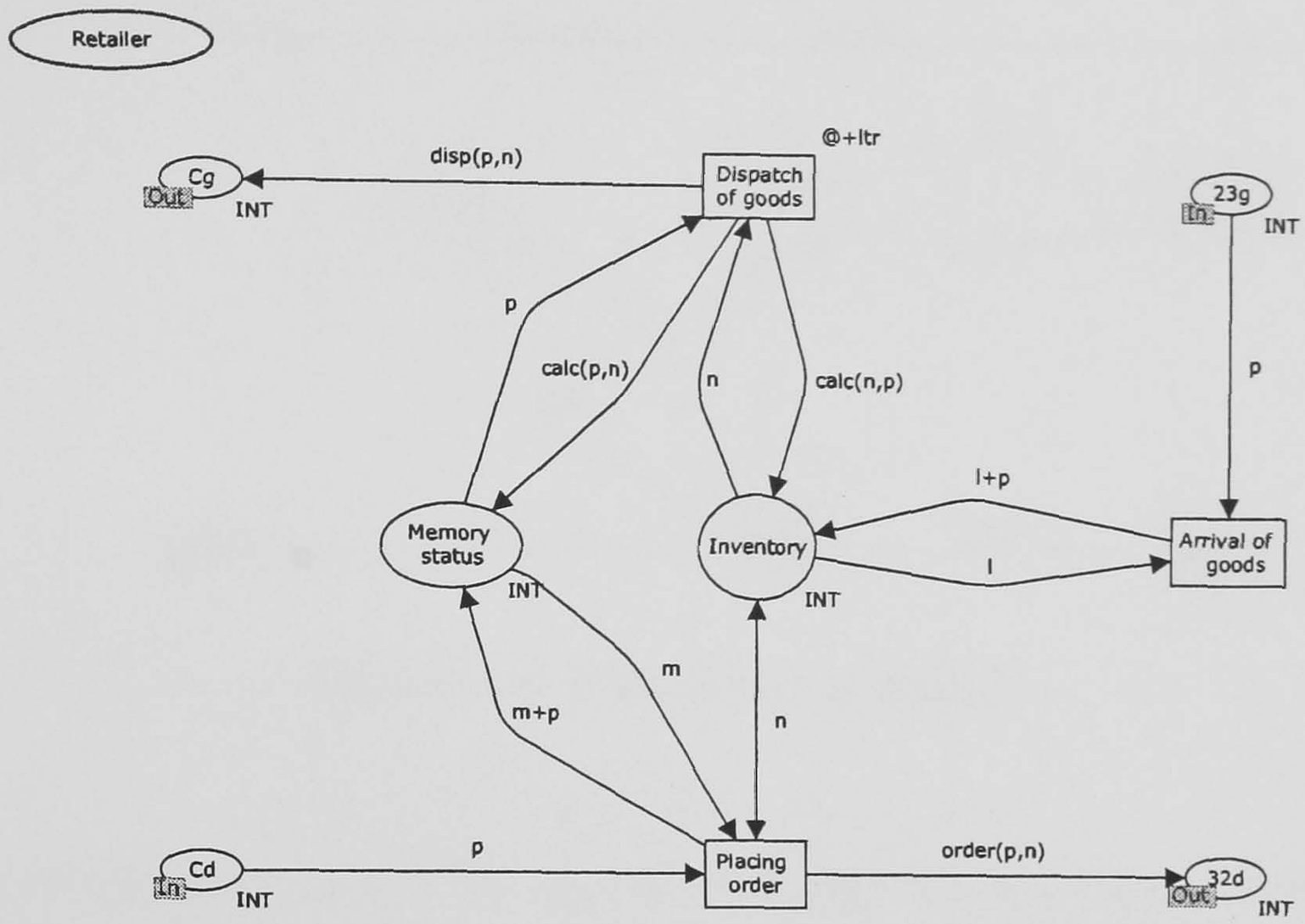

Figure 5.3: The subnet Retailer

Using HCPN definition, Figures 5.2 - 5.6 are represented in 5.7.

5.2.2 Sub-pages *Retailer* and *Manufacturer*

Aggressive ordering (AO)

Figure 5.3 shows the subnet of the module *Retailer* containing socket places *Cg*, *Cd*, *32d*, *23g*, places *Inventory* and *Memory status*, and three transitions *Dispatch of goods*, *Arrival of goods* and *Placing order*. Transition *Placing order* examines whether the existing stock is sufficient for the complete satisfaction of the quantity requested by the customer. If this is the case, the appropriate order quantity is placed as order to the upstream node (Supplier) and a copy of the quantity is stored in place *Memory status* which plays the role of a buffer. If the existing stock held by the Retailer for a complete supply of goods is not sufficient, then the Retailer orders the amount which is accumulated between current stock and the last order plus

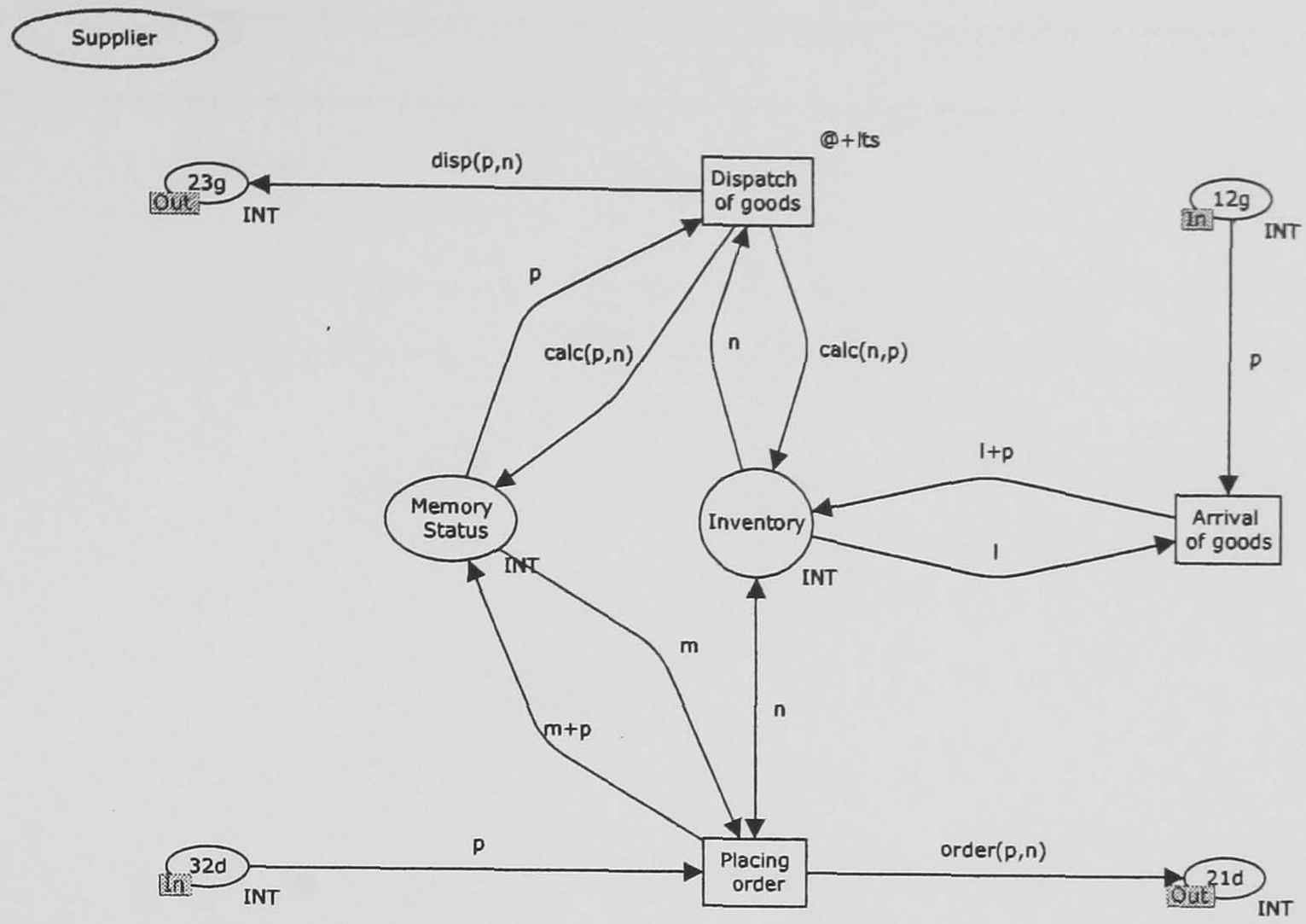

Figure 5.4: The subnet Supplier

the amount of the last order. This type of ordering policy is based on the retailer's estimate that the next order placed by Customer will follow the last demand pattern.

In transition *Dispatch of goods* the ordered products are dispatched to the downstream node (Customer), after being controlled by places *Memory status* and *Inventory*. In the case where the stock is sufficient to fulfil the last order, the Retailer dispatches the full stock. Transition *Arrival of goods* is associated with goods reception which are later stored on the inventory (warehouse). It is assumed that there is no delay in this process (i.e., goods received by the upstream node are delivered immediately to the warehouse). We also assume also that there is no delay on control sequences taking place before the dispatch of goods to the downstream node and on decisions related to the amount of products to be ordered. This assumption is based on the fact that decision-making and dispatch of goods at every node is performed much faster than all the other running activities in the supply chain. In contrast, we indicate by *ltr* the lead time between dispatch and delivery of products from Retailer to Customer.

Manufacturer page shown in Figure 5.6, has exactly the same structure as the Retailer page related to the process of receiving order by an upstream node and

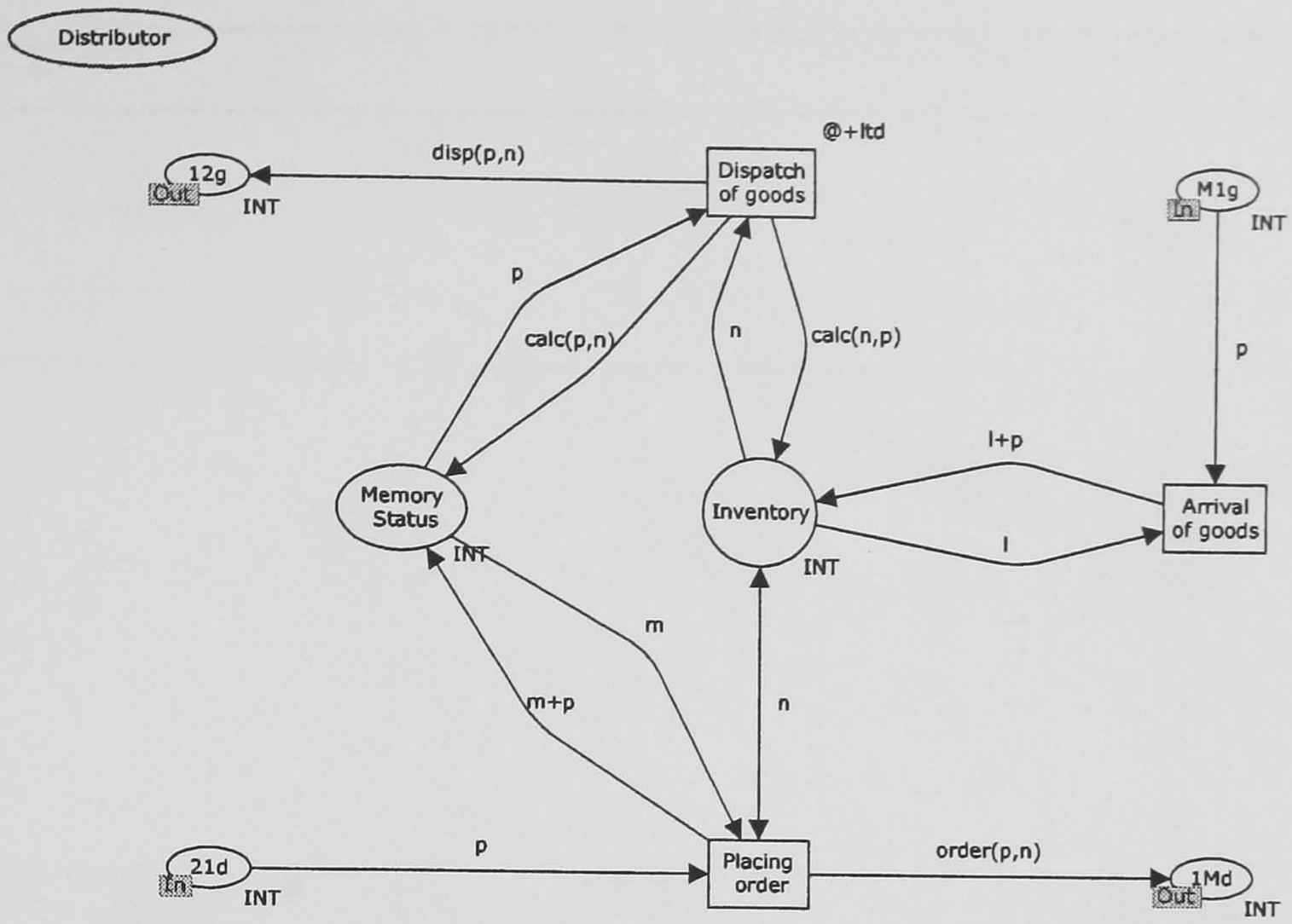

Figure 5.5: The subnet Distributor

dispatching the goods to a downstream node. However, manufacturers in general have different policies of receiving and ordering raw materials, since they usually establish contracts monthly or even annually with raw materials suppliers. For the simplification of our model we assume that the manufacturer receives a settled bulk of raw materials if a current order received by the distributor exceeds the amount of this quantity. Otherwise, the manufacturer places exactly the same amount of order for the purposes of maintaining the inventory level. Place *Md* receives the order demand by the Distributor and forwards the amount of order to transition *Entrance of goods* which, in turn, checks if this exceeds the amount of bulk ordering. Place *Mg* models the raw material suppliers' activity and we assume that the time needed for raw materials to be transformed to finished products is *ltrm*. Hence, Manufacturing site can be considered as a push logistics system where manufacturing process takes place when it is triggered off by downstream demand. Next we discuss the declarations (inscriptions, functions and variables) used in all pages of HCPN.

Declarations for the HCPN supply chain model

As we have seen in Chapter 2 declarations in CPN are used for the quantitative

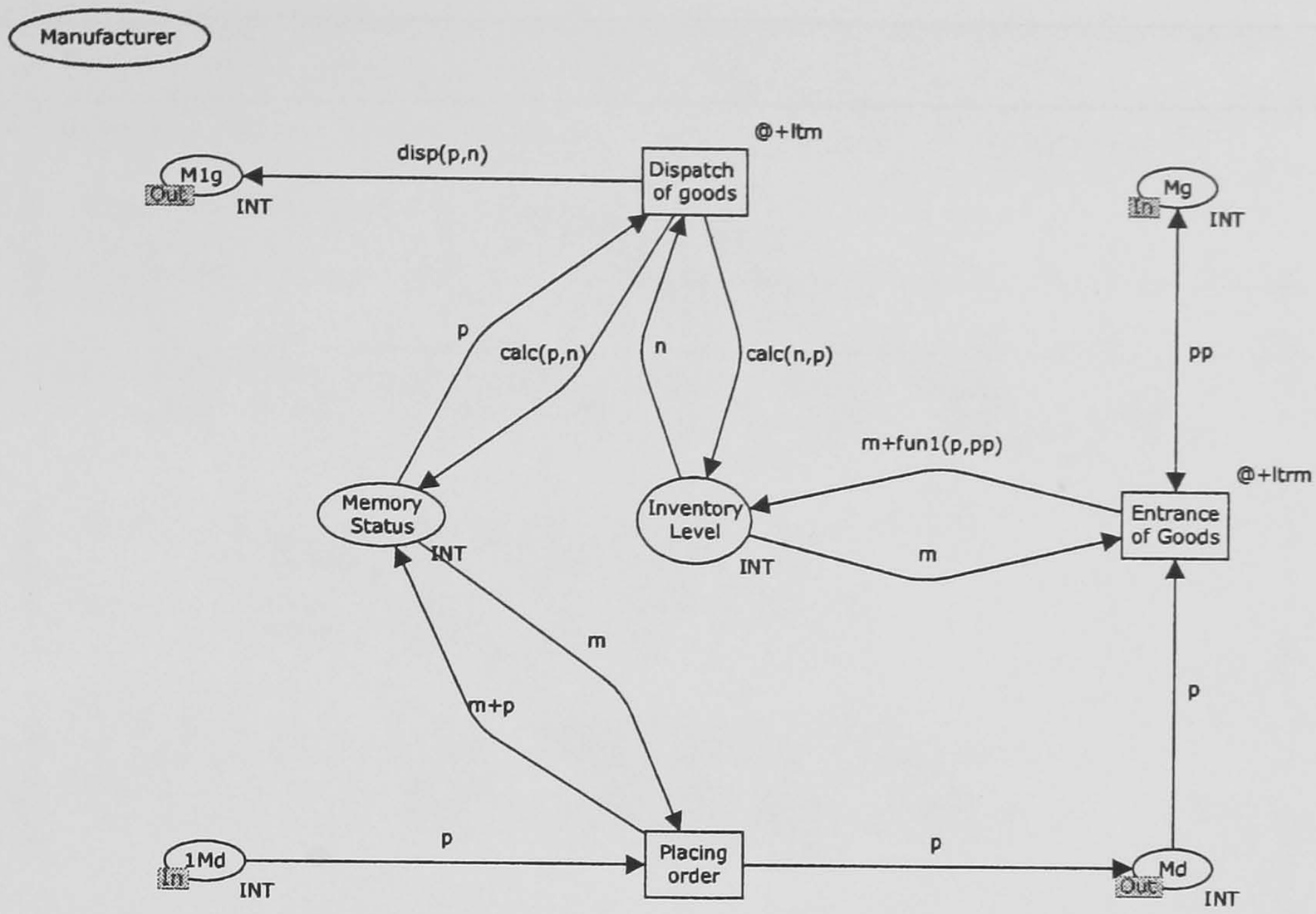

Figure 5.6: The subnet Manufacturer

and qualitative analysis of a model. To facilitate our analysis, we need to associate the flow of goods and orders by using assigned variables. Moreover, all activities performed at all stages of the supply chain (such as dispatching of goods, control sequences and order placing) must clearly defined via a set of rules. This set can be encapsulated in well-defined functions. CPN Tools for simulation and modelling of CPN use CPN ML, which is obtained by extending Standard ML. All functions and variables used in supply chain HCPN model are written in ML code which extracts all data from the CPN model. Declarations are shown in Figure 5.8.

As can be inferred from the declaration box, in order to model the supply chain HCPN with CPN Tools we need to use a standard timed colour set ($INT = int\ timed$). Variable p in all pages represents the amount of orders and goods transferred within the supply chain and is defined as an integer. Variable integer n is used to indicate the amount of stock held at each node. Variable m updates the memory status each time a new order is placed, while l is used in a similar way to update the inventory each time new products are received. In the Manufacturer node, variable pp is associated with the bulk of raw materials arriving from suppliers Mg each time such a command is issued by the manufacturer. Before carrying out a HCPN simulation,

(i) S = {Supply chain, Manufacturer, Distributor, Supplier, Retailer}

(ii) SN = {M@Supply chain, S1@Supply chain, S2@Supply chain, S3@Supply chain}

(iii) SA(t) = { Manufacturer if t= M@Supply_chain; Distributor if t= S1@Supply_chain; Supplier if t= S2@Supply_chain; Retailer if t= S3@Supply_chain }

(iv) PN = {Mg@M, 1Md@M, Md@M, M1g@M, M1g@S1, 21d@S1, 1Md@S1, 12g@S1, 12g@S2, 32d@S2, 23g@S2, 21d@S2, Cd@S3, 23g@S3, Cg@S3, 32d@S3}

(v) PT(p) = { in if p {Mg@M, 1Md@M, M1g@S1, 21d@S1, 12g@S2, 32d@S2, Cd@S3, 23g@S3}; out if p {Md@M, M1g@M, 1Md@S1, 12g@S1, 23g@S2, 21d@S2, Cg@S3, 32d@S3} }

(vi) PA(t) = { {(Md@Supply chain, Md@M), (Mg@Supplychain, Mg@M), (M1g@Supply chain, M1g@M), (1Md@Supply chain, 1Md@M)}, {M1g@Supply chain, M1g@S1), (1Md@Supply chain, 1Md@S1), (12g@Supply chain, 12g@S1), (21d@Supply chain, 21d@S1)} ..... }

(vii) PP = 1`Supply chain

Figure 5.7: HCPN from Figures 5.2 - 5.6 represented as a many-tuple

we assume that customer demand Cd follows a normal distribution with mean m and standard deviation s. Both mean and standard deviation are real numbers (100.0 and 8.0, respectively). The normal distribution is implemented in CPN ML code with function $normal(m, s)$. Since CPN Tools can not treat real numbers in the simulation process, we round each random value to an integer by using the function $distr(m, s) = floor(normal(m, s))$.

Function $order(i, j)$ describes the inventory control policy and models the process of placing an order. Variables i and j are input assignments and represent a received order and the inventory, respectively. In case there is sufficient stock, an amount i is send as order to the upstream node. In cases where current inventory level j is below the received order j, then the accumulated value $i - j$ (back-orders) plus the estimate of the next order i is placed as order to the upstream node. Function $calc(i, j)$ models the control sequence between memory status and current stock and is used to calculate the balance of inventory and memory status when a new immediate shipment is due to take place. The exact amount of products dispatched

```
val ltrm= 1;
val ltm = 4;
val ltd  = 3;
val lts  = 2;
val ltr  = 1;
val sigma = 8.0;
val mean = 100.0;
colset INT = int timed;
type REAL = real;
var q, w, p, pp, m, n, l: INT
normal(mean,sigma) : real;
fun order(i,j) = if(i<=j) then i else 2*1-j;
fun calc(i,j) = if(i>=j) then i-j else 0;
fun disp(i,j) = if(i>=j) then j else i;
fun fun1(i,j) = if(i>=j) then i+j else i;
fun distr(mean,sigma) = floor(normal(mean,sigma));
```

Figure 5.8: The declaration box of supply chain HCPN

to the downstream node is modelled by function $disp(i, j)$.

In order to understand better how the functions $disp(i, j)$ and $calc(i, j,$ we consider as initial condition that the amount of inventory is n and the amount of order is p, (memory m on initial condition is cleared and hence $m = 0$). We consider the simple scenario when $p <= n$. When token p, arrives as orders from place *Cd*, transition *Placing order* removes this token from *Cd* and adds momentarily (no time inscription has been given to *Placing order*) a token of the same value to place *Memory status* and place *32d*. Then *Memory status* updates its tokens by now carrying $m+p = 0+p = p$ tokens and sends this token to transition *Dispatch of goods*. In the meantime, place *Inventory* also sends its token to the same transition. Then transition *Dispatch of goods* is being activated and retailer can dispatch the ordered products p to the customer. Since $p <= n$ the output of the function $disp(i, j)$ will be assigned to p. Function calc(i,j) associated with place *Memory status* now updates again its token and its output is now set to 0. This means that currently no product delivery is pending. In a similar way function calc(i,j) associated with place *Inventory*, updates its token to $n - p$ and hence current stock contains $n - p$ products.

Function $calc(p, n)$ should not be confused with $calc(n, p)$. Although these two

functions operate in a similar way and have the same output assignments, their input assignments are different. We have also used variables *ltr, lts, ltd, ltm* as lead times for dispatch of goods by Retailer, Supplier, Distributor, and Manufacturer, respectively. Note that we need also to introduce guard functions in transition *Dispatch of goods* to avoid delivery of zero products.

Moving average (MA) forecasting technique

In the moving average technique, the demand forecast is calculated as the average of the last recorded observations and is given by:

$$\hat{D}_t = \frac{1}{N} \sum_{n=1}^{N} D_{t-n+1} \tag{5.2.1}$$

where $\hat{D}_t$ is the next demand forecast while D_t, N indicate the amount and number of observations, respectively. Figure 5.9 shows the corresponding subnet *Retailer* which models the moving average procedure. Recall that system is "driven" by a stochastic end-customer demand profile applied at the end of the chain.

It can be inferred from Figure 5.9 that we have introduced three new places *Update*, *Counter* and C and a new transition *Aver*. Place *Update* is a memory buffer that stores the sum of past observations (e.g., updates the past information) while *Counter*, C and *Aver* are used to calculate the sum (indicated by the variable b) and the number of past observations (variable z). The guard equality, $y = z$, on transition *Aver* ensures that data are treated in chronological order. Function $aver(i, j)$ calculates the next demand forecast $\hat{D}_t$ following equation 5.2.2. Note that $\hat{D}_t$ must be rounded to an integer since CPN-Tools can not treat real numbers. Hence, two new lines have been added to the declaration box:

```
fun conv(i) = Real.fromInt((i));
fun aver(i,j) = round(conv(i)/conv(j));
```

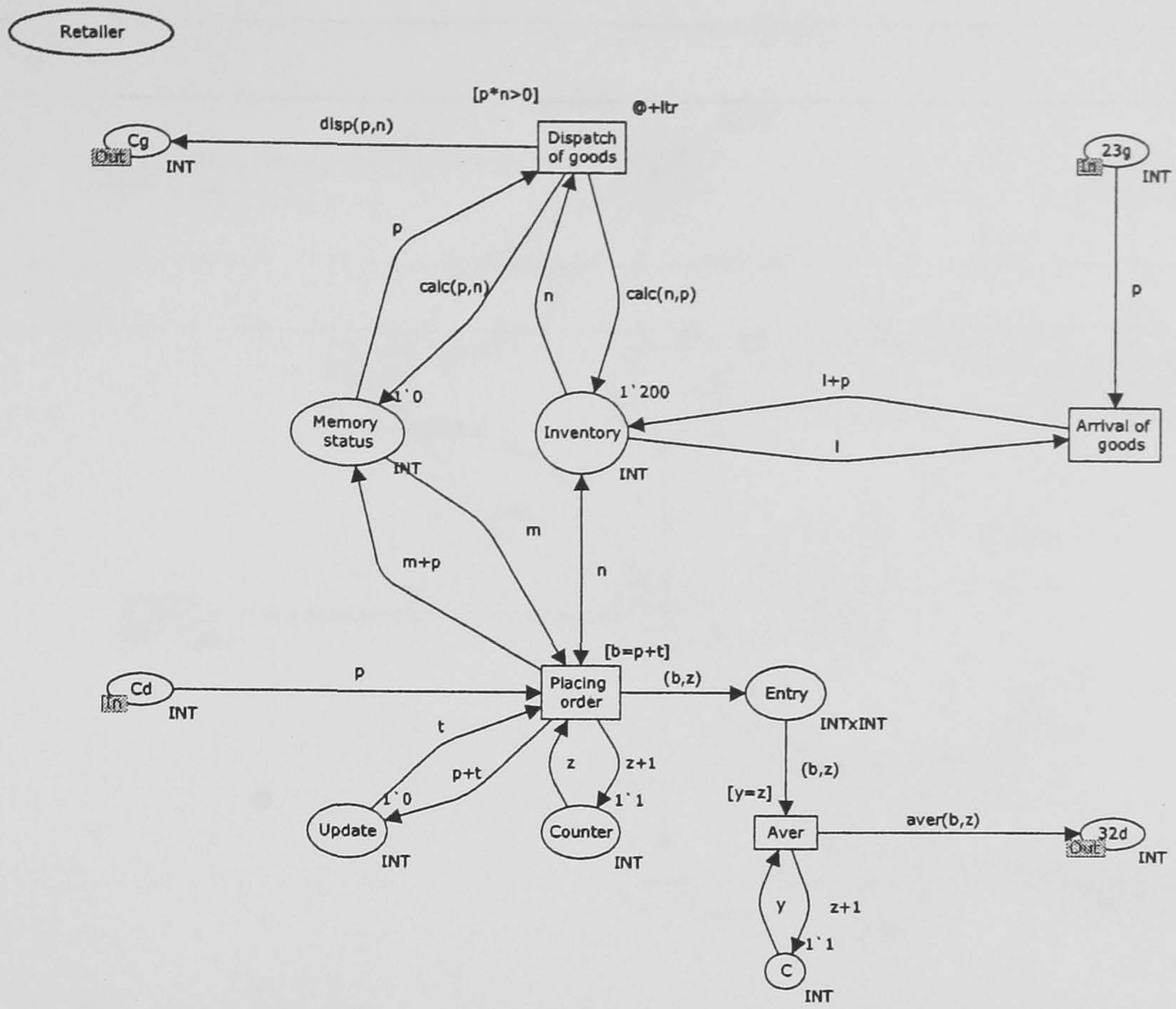

Figure 5.9: The subnet *Retailer* for (MA) techniques

Exponential smoothing (ES) forecasting technique

In cases where we wish to use previous demand forecasts and past observations we can benefit by using the exponential smoothing technique which is given by:

$$\hat{D}_{t+1} = \alpha D_t + (1-\alpha)\hat{D}_t \qquad 0 < \alpha < 1 \tag{5.2.2}$$

where α is the smoothing coefficient. This coefficient controls the weight placed on to the most recent data. Figure 5.10 shows the corresponding subnet *Retailer* which models the exponential smoothing method.

Places Ind and B are used as counter integers and guard equality $z = g$ on transition *estim*, to guarantee, similarly to the moving average model in Figure 5.9, that new data do not overtake older. Place *proest* forwards the amount of last order observation (variable p) to be processed by transition *estim*. Place A updates the last order (αD_t indicated by variable f) and function $expsmo(p, f)$ calculates the demand forecasts. Again, all demand forecasts must be rounded to integers. In order to model the smoothing forecast technique we have added the following lines

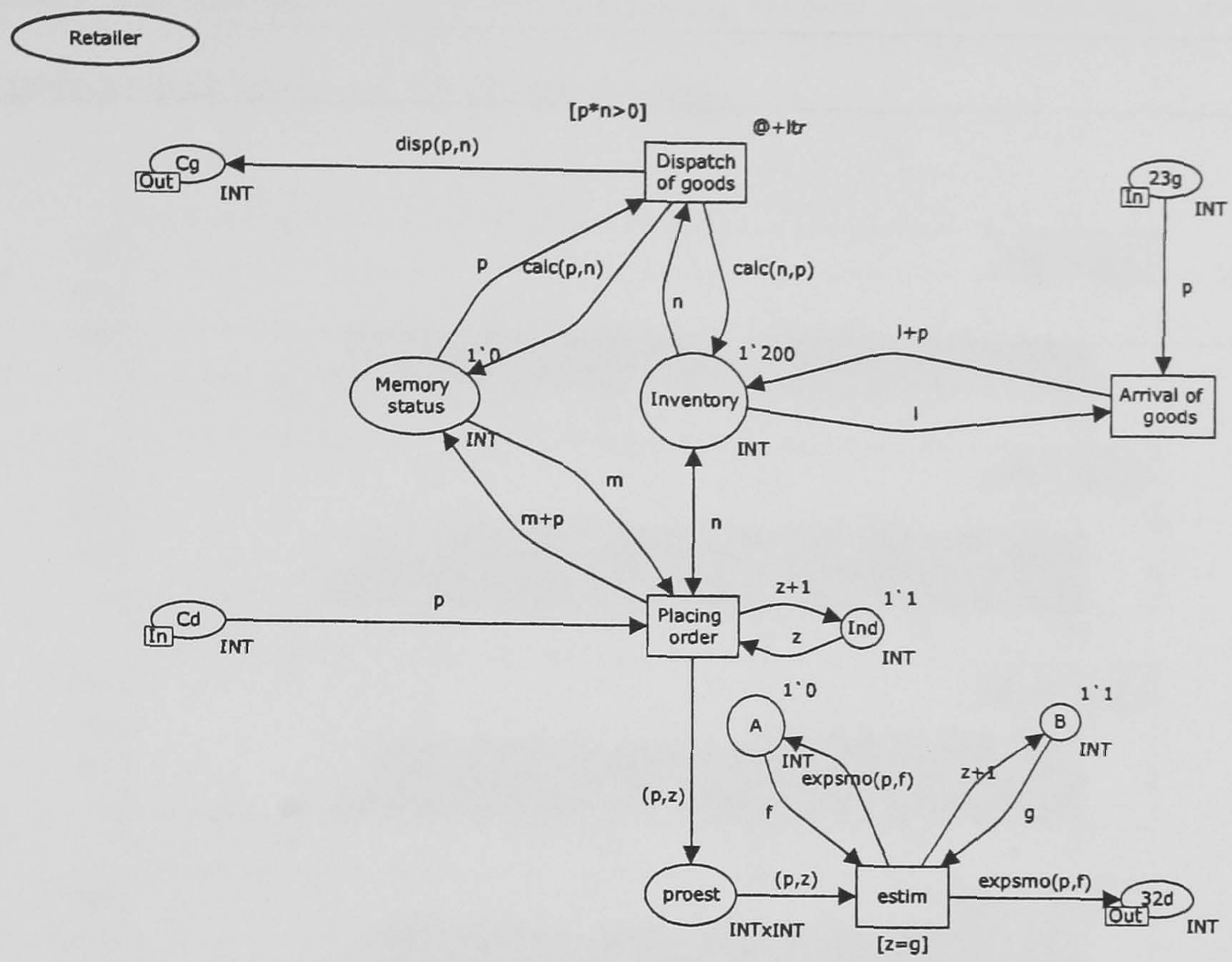

Figure 5.10: The subnet *Retailer* for (ES) techniques

to the declaration box:

val alpha = 0.9;

fun ReaInt(i) = Real.fromInt((i));

fun expsmo(i,j) = round((alpha*ReaInt(i) +

(1-alpha)*ReaInt(j)));

5.3 Simulation results and performance analysis

The simulation results show for each ordering and forecasting technique the changes on inventories at each time-period, the corresponding backorders and a customers' satisfaction metric. For simplicity we assume constant lead times for the distribution of goods throughout the supply chain. The initial inventory in each node for the (AO) case is set to 100 and 200 for (MA) and (ES). To make our model more realistic, we let participants order in batches but we also consider simulation periods where no orders are placed. Note also that each participant may not dispatch the ordered

goods instantly due to administrative delays or laches in the dispatch section. The simulation period has been set to 60 time-steps.

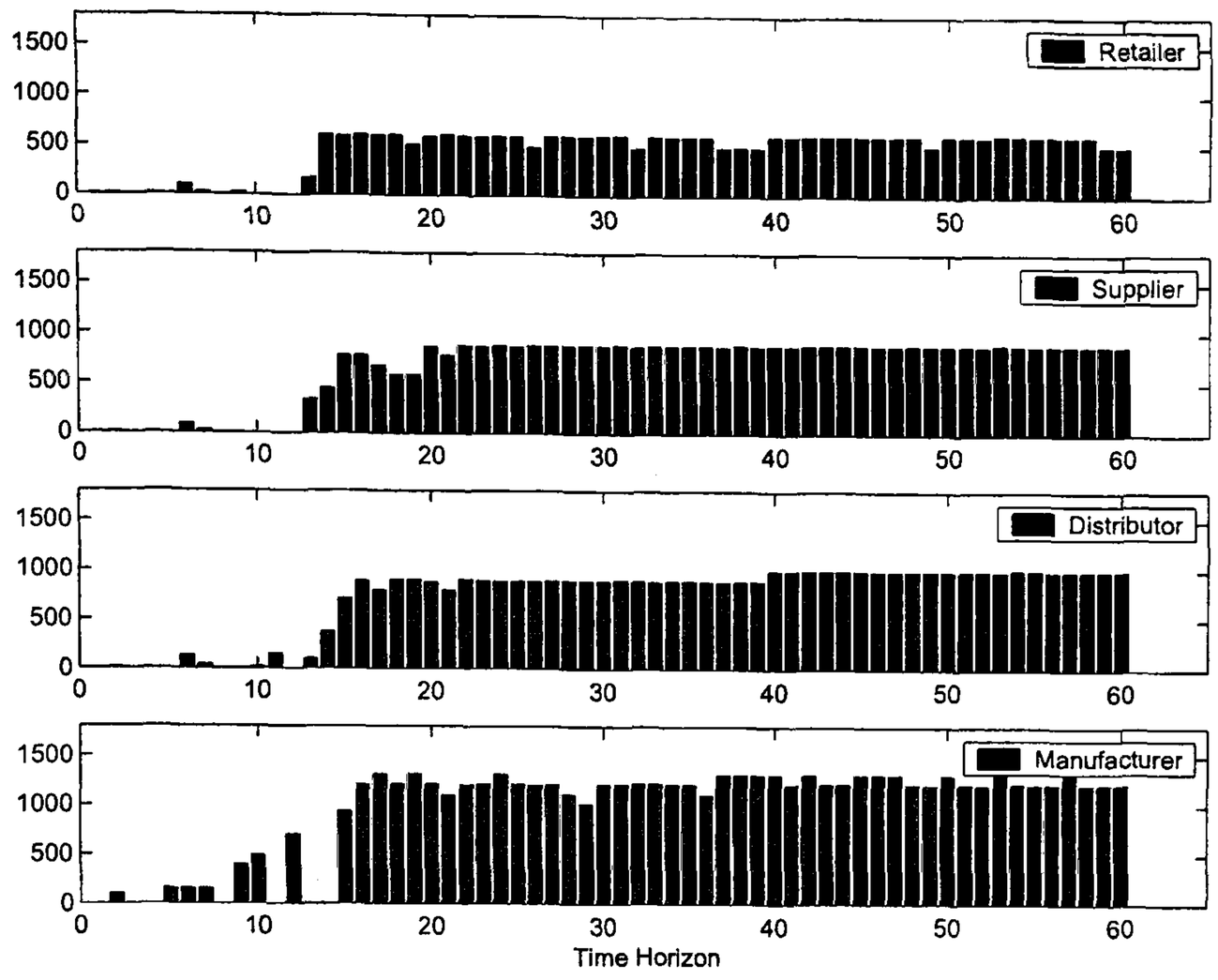

Figure 5.11: Inventory levels in AO policy

Figure 5.11, Figure 5.12 and Figure 5.13 depict the inventory position of all supply chain participants for all three different cases. More specifically, Figure 5.11 shows clearly the increase in inventory as we move upstream the supply chain. This observation has also been made in [MNPV04]. After a simulation time of sixty periods, the inventory has been heightened by 400%, 780%, 900% and 1100% for Retailer, Supplier, Distributor and Manufacturer, respectively. This inventory augmentation clearly illustrates demand amplification (bullwhip effect).

It can be inferred from Figure 5.11 that shortages on each participant occurred only on the first 12 time steps. This is mainly caused by batch ordering or when there has been no sufficient inventory to fulfill the downstream demand before aggressive ordering. This can be more clearly seen by considering the Retailer site. The first 12 markings representing customer demand pattern are: $1'98@1 + +1'91@2 + +1'100@3 + +1‘98@4 + +1‘107@5 + +1‘90@7 + +1‘102@8 + +1‘98@8 + +1‘100@10 + +1‘97@11 + +1‘105@12 + +1‘94@12 + +$ At time $t = 5$

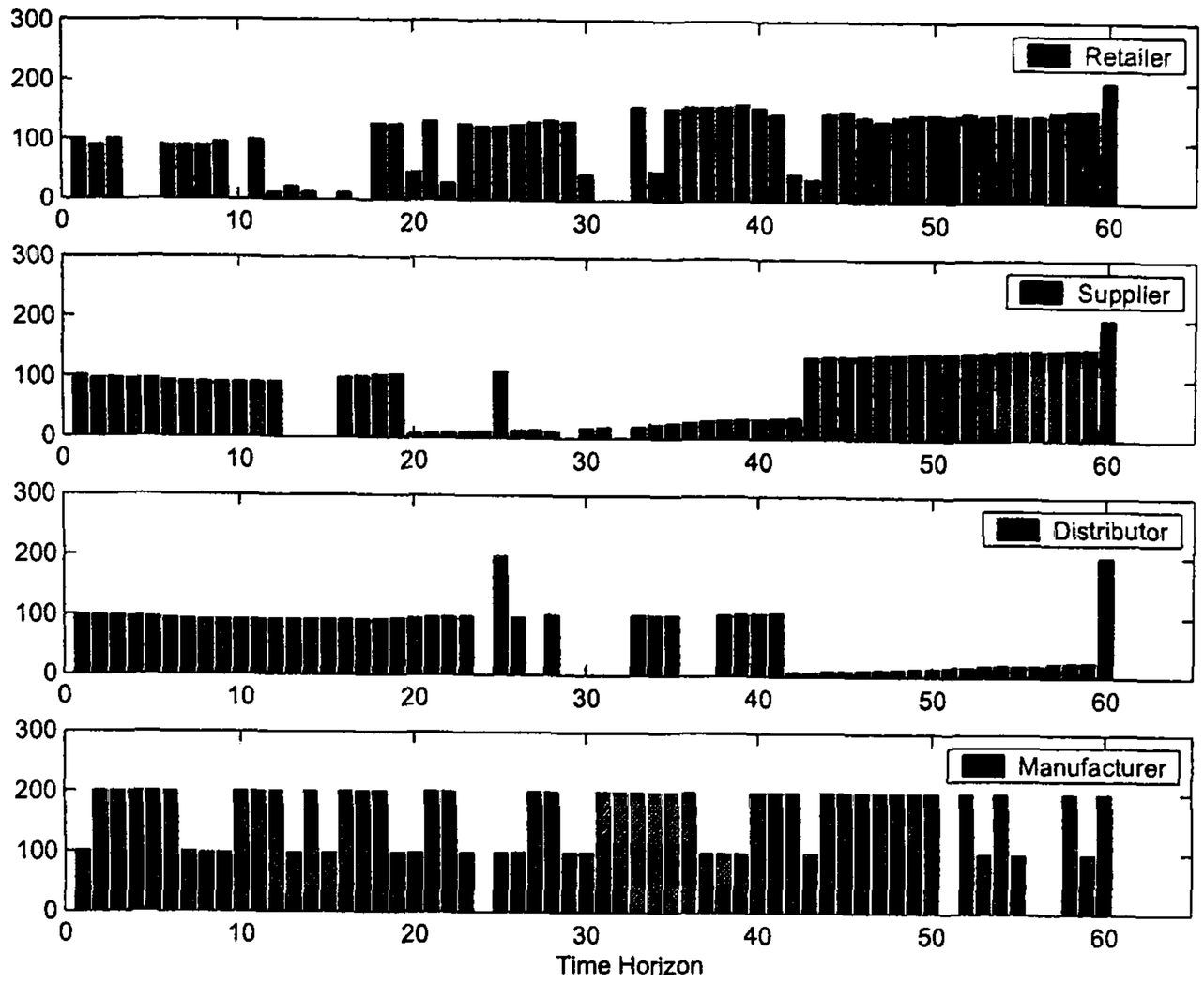

Figure 5.12: Inventory levels in MA policy

Customer orders 107 products while Retailer's inventory is below 100. Furthermore, at times $t = 8$ and $t = 12$ customer places two different orders. Due to aggressive ordering (AO) adopted by Retailer at these times, inventory at the 13th time step increases significantly.

The shortages on inventories depicted in Figures 5.12 and 5.13 demonstrate the impact of batch ordering in forecasting techniques. In the MA case, batch ordering followed by customer takes place at periods represented by markings: 1‘104@3 + +1‘107@3, 1‘92@12 + +1‘99@12. Delays on dispatching the goods to Retailer by Supplier on consecutive simulation periods 31 and 32, and to Supplier by Distributor at periods 13,14,15,29 and 32 also cause shortages in inventories. Similarly, zero inventories appear in the Distributor's and Manufacturer's sites, caused by delays on dispatches and batch ordering. As shown in Figure 5.13 for the ES technique with α=0.9, there are more periods where participants encounter inventory shortages in comparison to MA technique. This means that batch ordering and delays on goods dispatches using the ES method tend to have more impact on inadequacy of supplies. Note also that ES produces slightly more fluctuations on

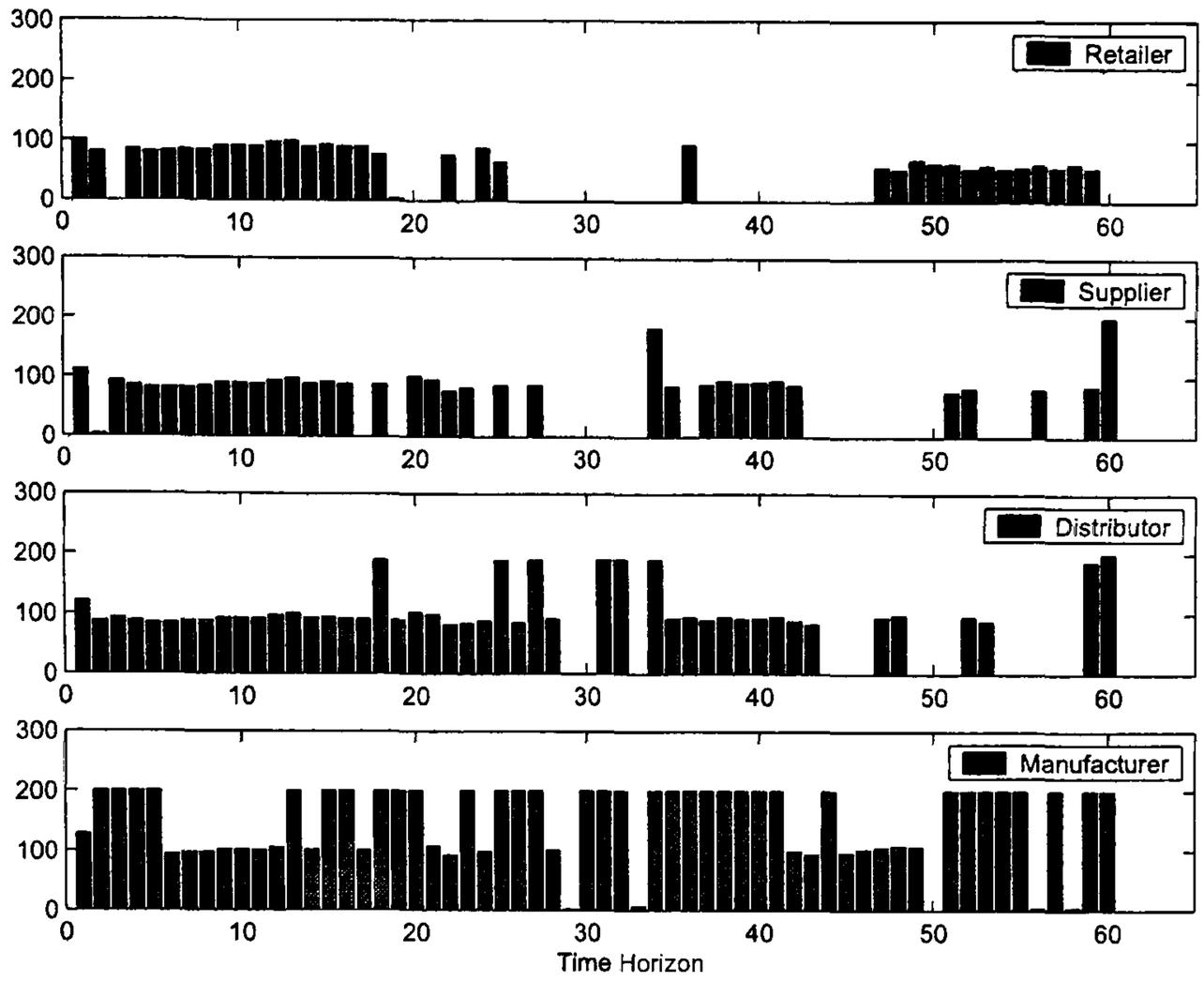

Figure 5.13: Inventory levels in ES policy

inventories than the MA technique, although inventory levels on MA are very low for long periods (e.g., Distributor's inventory for the last 20 periods is below 15 and Retailer's inventory between the 26th and 40th period is below 20).

Figure 5.14, Figure 5.15 and Figure 5.16 depict the backorders in supply chain participants for AO, MA and ES techniques, respectively. It can be inferred that due to the more inventory shortages when participants follow ES forecasting techniques the number of backorders is also larger with this method. Note also that backorders for MA and AO appear mainly on the first half of the simulation period in contrast to ES where they appear on the second half. Figure 5.15 and Figure 5.16 illustrate that backorders in smoothing forecasts take place at the downstream part of the supply chain in contrast with AO (Figure 5.14), where inventory shortages arise upstream. As expected from the observed backorders levels, customer satisfaction with ES is lower compared to the other two techniques.

We can add more suppliers between the Customer and Manufacturer nodes. Each of them can be modelled by any subpage *S1*, *S2* or *S3* based on retailer description given above. This could be possible if we assume that in modelled supply chain each

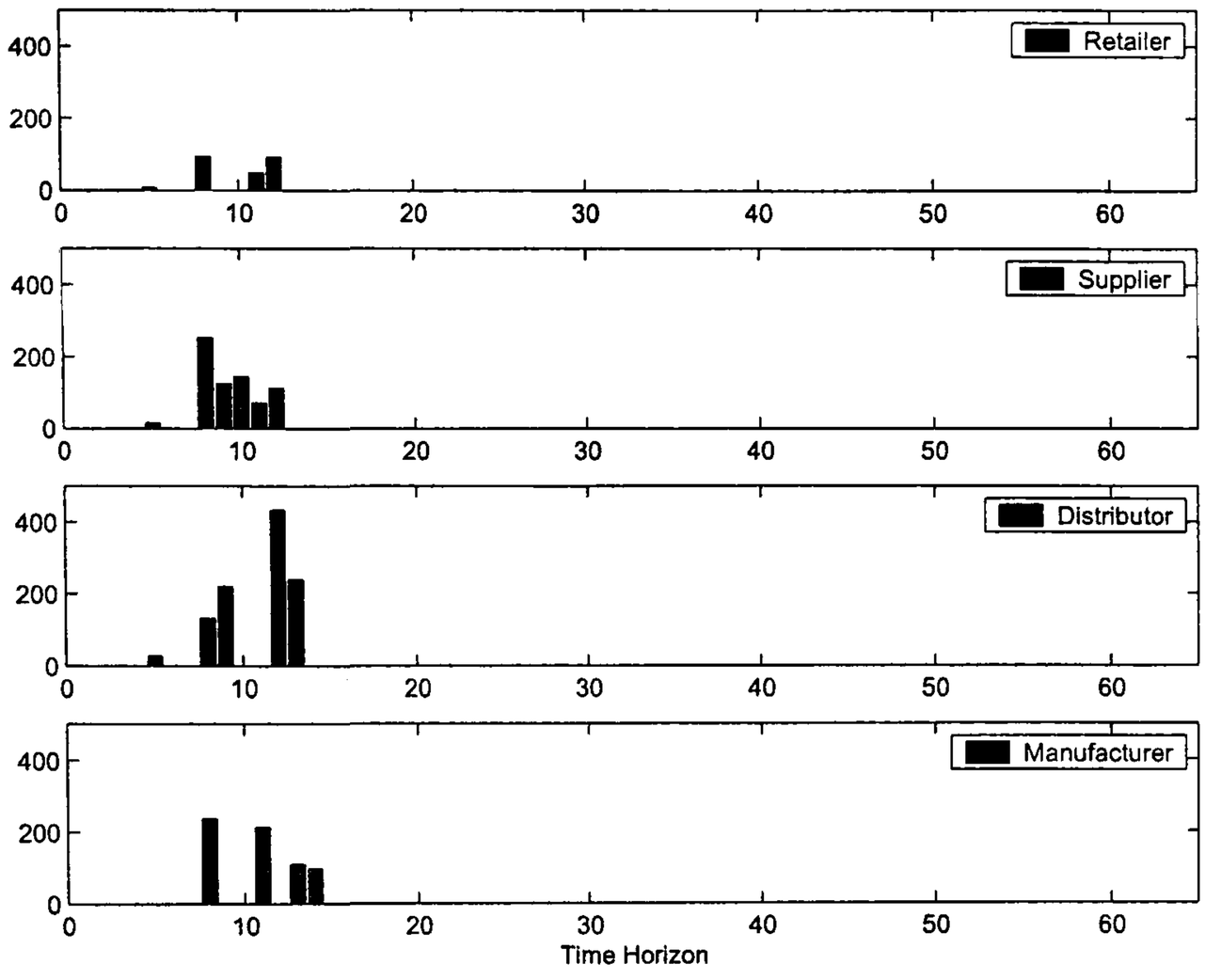

Figure 5.14: Backorders in AO policy

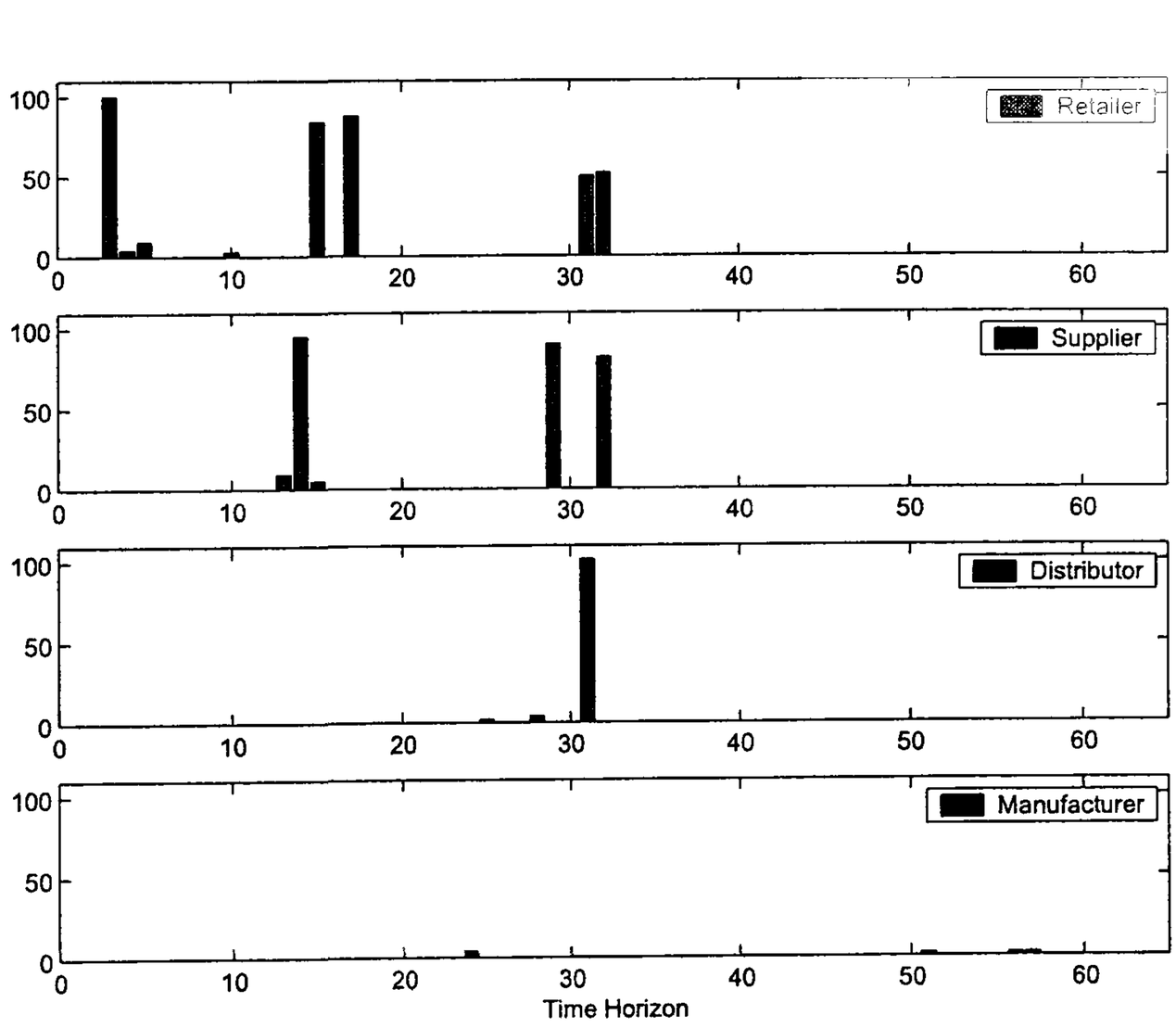

Figure 5.15: Backorders in MA policy

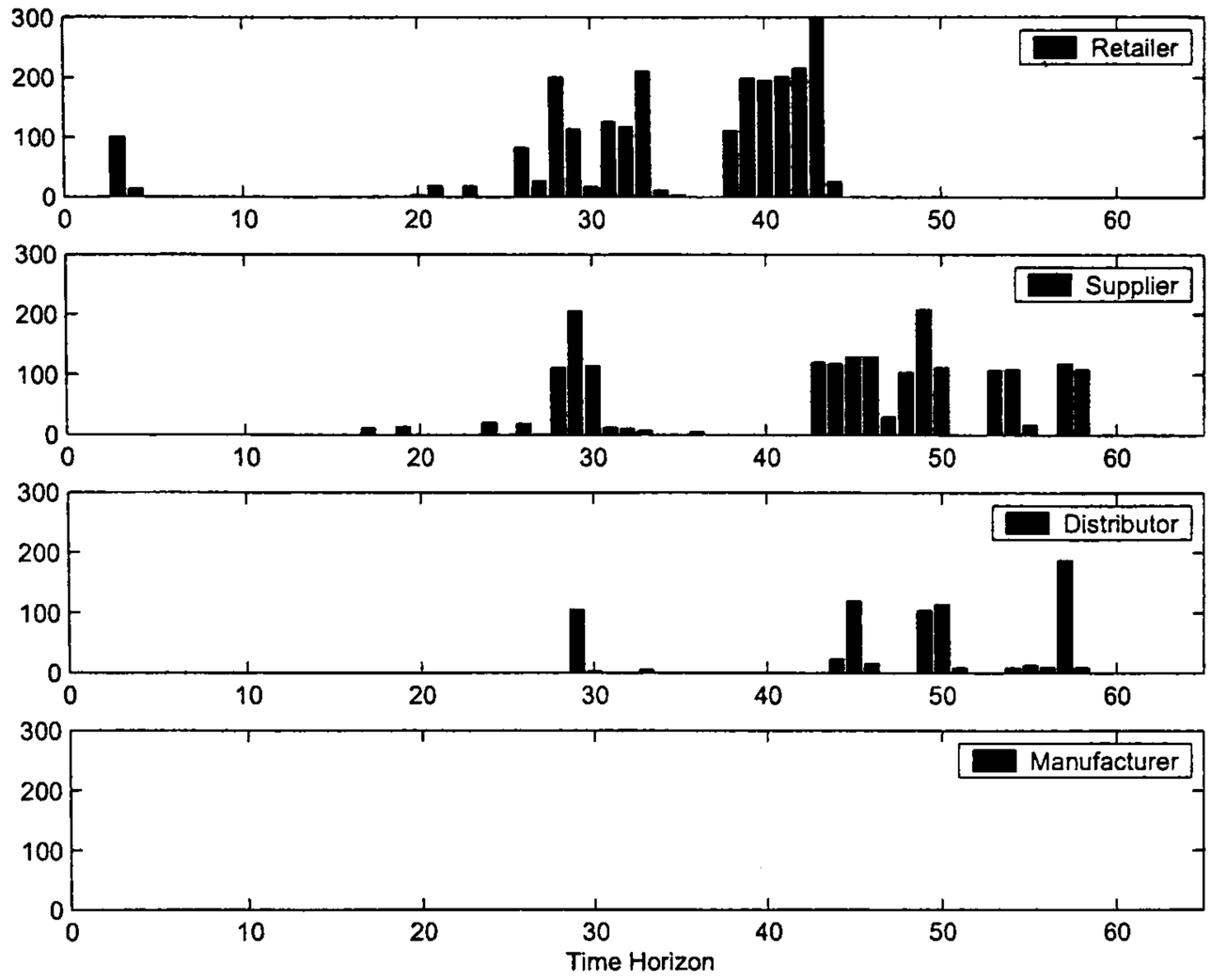

Figure 5.16: Backorders in ES policy

of the node makes decisions following the same inventory control policies. In cases where each node behaviour differs from other nodes, we have to implement a new subpage for each of them (or, at least, for those whose behaviour is different). The use of cost function can also help us to compare the results of different inventory control policies by using the analysis of inventory control models described in chapter 2.

Chapter 6

Modelling methods in aluminium rolling industry: A case study

6.1 Introduction

In previous chapters we have seen that the bullwhip effect happens in manufacturing when information about consumer demand becomes increasingly distorted as it moves upstream in the manufacturing process. This distortion leads to excessive inventory throughout the system, poor product forecasts, insufficient or excessive capacities, product unavailability, and higher costs generally.

Production management problems in industry play an essential role in the supply chain management area, by which managers can determine the production loading plan in order to satisfy the end customer demand. Thus, production planning in manufacturing involves in most cases the synchronisation with the downstream demand and therefore has a strong impact in warehouses of both manufacturers and other participants of supply chains. A more detailed task in manufacturing is production scheduling where managers in the context of the optimal production planning must couple individual products with individual productive resources in the shortest times. This chapter presents the modelling and simulation of an aluminium coils production plant, by providing an efficient representation for such production processes. Four different scenarios are performed to capture the dynamics of production system and to show how the production managers can use simulation tools in order to cope with demand amplification or downstream demand fluctuations

which are studied and analysed in previous chapters.

The main disadvantage of manufacturing processes is that they are often characterised by complexity and the presence of huge amount of data and parameters. To describe, analyse and evaluate these processes, the need for models and simulation tools have long been recognised to be necessary and have been studied extensively. By the manipulation of the model, it is hoped that knew knowledge about the production process can be obtained without the inconvenience or cost of manipulating the real process itself. Therefore, it becomes indispensable to understand production systems' behaviour and the parameters that affect the performance of production lines.

The case study in this chapter is motivated by the problems faced by Bridgnorth Aluminium Ltd., a manufacturing site located in Bridgnorth, England. Bridgnorth Aluminium is the one of the world's leading producers of high quality rolled aluminium lithographic strips (coils) products. In recent years, Bridgnorth has become one of the major suppliers of litho products in today's global market. Current capacity is approximately 46,000 tonnes of finished coils which are used principally for printing purposes by known imaging and film companies like Fuji Photo Film and Agfa.

Decision making in Bridgnorth involves the development of a weekly production loading plan according to a list of products assigned by customers by considering workforce level, manufacturing capacity, priority orders and other factors. This kind of finished products are implemented in make-to-order (MTO) industrial environments where a due date is agreed with the customer. End items are placed in the master weekly schedule, and production planning then specifies the necessary production [Mar97], [Ste89], [VBWJ05]. While the main production schedule is based on orders in hand, it is possible under certain conditions (e.g., a known maintenance plan in machinery that might slow down the production in the near future or reduced orders) to develop a schedule of products with high demand.

The main objectives of the Bridgnorth Aluminium case study is to create a simulation tool as accurate and flexible as possible for modelling the production

process. The work is focused mainly in the most complex part of production line where annealing, cold rolling, levelling-degreasing-stretching operations and temporary storage of limited capacity facilities are integrated with the aid of a moving crane. Sequencing all these operations gives rise to numerous problems and complexity, especially when customer requirements dictate a large variety of product characteristics. Simulation tool outputs include statistical data of each individual work machine station, measurement of throughput in a given simulation time and counting of crane movements and temporary storage capacity area. The full production process is also captured through graphical representation of the workload of each machine centre and temporary storage area. In effect, the user can assess the performance of various scheduling policies and identify bottlenecks and likely future capacity overloads.

The model was implemented in MATLAB but the code can be "translated", after minor modifications, to any other high-level programming language, if required. MATLAB is a powerful tool for mathematical programming, modelling and simulation. Preliminary trials indicated that MATLAB is perfectly adequate for running the code fast and efficiently for the anticipated complexity of the final simulations. The software tool also provides a friendly user interface through which the user interacts with the simulation tool without programming or recompiling. The software programme is written with the aid of functions which model each single process/subprocess (e.g., cold rolling process) and other issues that need to be modelled such as annealing times depending on the coil type entering the annealing machine. All functions are integrated in routines and subroutines and the result of this integration is about 7000 lines of code (which translates to over 20000 lines in typical high level programming language e.g., C, C++) and 41 different functions.

Due to high complexity in some parts of the production plant, we were forced to make certain assumptions which are clearly described later. The main body of the tool was written having in mind a generalised model such that any changes in the parameters of the plant (e.g., number of annealing machines, cooling-times, etc.) can

be made very easily. We have also put an effort to make the modelling assumptions as realistic as possible.

In next section an effort is made to describe the production line for lithographic strip products. An extensive analysis based on this description can increase efficiency by allowing inventory only when it is needed, identifying bottlenecks, balancing capacity and generally coordinating the smooth flow of coils. Moreover, analysing the production line as a whole to find out where most of the time delays occur allows the production manager to focus attention on those "bottlenecks" in order to shorten throughput times.

The structure of this chapter is as follows: First a description of each stage of the production line is given, with all the assumptions clearly stated. Some important issues of Bridgnorth Aluminium production line are examined in terms of modelling and simulation. Next, the developed simulation system is introduced together with a description of the software tool and its main characteristics. Finally, various scenarios are presented addressing key factors. A quantitative analysis of the software tool is finally performed, involving some important indicators (e.g., throughput, effect of number of annealing machines, capacity of High-Bay, effect of cooling time, etc.) in production process modelling.

6.2 Description of production line for lithographic strip products

The production line shown in Figure 6.1 can be divided into three main individual work-stages according to the nature of the manufacturing process, time and layout of the plant's machinery. The first stage includes slabs' delivery, scalping, hot-rolling and tandem mill process. The second integrates temporary storage (High-bay), annealing and cold rolling, while the third stage includes levelling, stretching and degreasing treatment (BWG), quality control and final storage. All parameters used (capacities, delay-times, material properties, etc.) are set nominal values which can

be easily modified between simulations.

6.2.1 Phase 1: Shop floor

In the first production phase, slabs are delivered to the manufacturing plant by following a scheduled delivery date (generally delivery time takes approximately 2-3 weeks) which is based on an annual order plan. Batch of slab units according to customer needs and production scheduling enter Bridgnorth plant once or twice a week. Although there are 31 different slab types, for simulation purposes we consider 18 different coil products. Since certain types of slabs result in certain type of coils, a classification is made before entering in the shop floor.

A weekly process plan according to the amount and type of customers' orders is proposed and forwarded to the scalping process. The upper and lower surfaces are milled (15mm for top and 4mm for bottom surface) on a state-of-the-art scalping machine which takes 15-25 minutes depending of the size and quality of each slab. Sequence of scalping follows the weekly schedule. After that, the slabs leaving the scalping machine are covered by a protective cling film and are placed in a temporary storage area of a maximum capacity of 110 slabs (preferably up to 80). Breakdown and maintenance are taking place 2-3 times annually during the holiday period.

For hot rolling, the rolling slabs are heated to the necessary rolling temperature in a furnace. Up to twenty four slabs each with a maximum weight of ten tons can be heated at the same time. The time taken for heating is 5-6 hours for the narrower slabs and 9 hours for the wider. The furnace sequencing follows First-In-First-Out order and preheat recipes are used according to slab width. In the case of breakdown, (about once every 2 weeks, occurring mainly because of fan damage), those slabs that had been entered the furnace must be re-scalped and re-heated. The furnace is never switched-off but on weekends the temperature is decreased slightly and slabs to be treated as non-litho finished products are inserted according to production schedule.

After the furnace process the rolling slabs are taken directly from the furnace to the hot rolling mill using special cranes. Slabs are positioned first on a roller

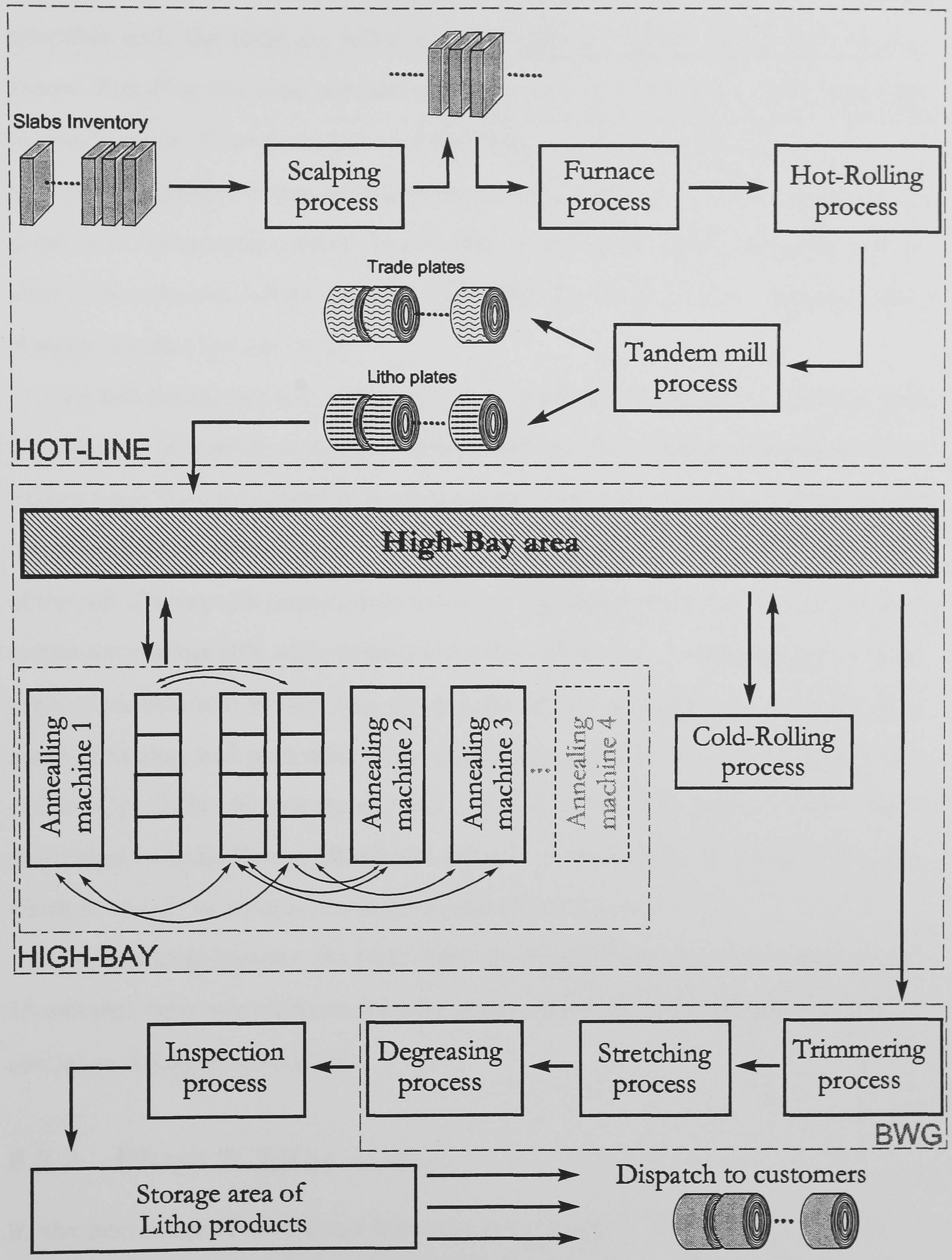

Figure 6.1: Production line of Bridgnorth Aluminium

table driven by electric motors. In the reversible mill a single slab is treated every time, while the time taken for the process is approximately 15 minutes. On the reversible mill, the slabs are rolled down to their hot rolling thickness in several phases. Finishing thickness of rolled plates is set to 15mm for Litho plates although the thickness for those plates intended for Litho products can vary between 14.5 and 15.5mm. Breakdown occurs several times and the productivity of hot-rolling mill is about 65%. Maintenance takes place mainly on weekends. The resulting hot rolled plate is then sheared using a cropping shear with the aid of a camera. Sheared pieces of strips are destined for scrap.

The last continuing hot rolling process involves the tandem mill which is used to achieve a desired thickness and surface quality. The output thickness is set to 2.6mm while the time needed is approximately 12 minutes for narrower slabs and 20 minutes for wider slabs rolled as plates. The tandem mill can also specify the width of the coil. During this process there are about 50-100mm width losses. Tandem mill realisation reaches 60% while breakdowns come about every 8-9 hours; consequently users of tandem mill usually stop the process for half an hour every 8 hours. As a result of tandem mill processing, plates are wound into roll-form (coils) after a series of flat rolls. Coils are then stored temporarily in the hot-line building before being dispatched to Litho Centre. Maximum capacity of storage area is currently 50 coils, which is one of the parameters under consideration for modelling.

For modelling purposes the coils that exit the Hot-Line process are divided into 18 different types according to the final gauge, width and whether are intended for special annealing furnace times.

6.2.2 Phase 2: Litho centre

In the next stage of production line coils are dispatched from the Hot-Line shop-floor to a column location (High-Bay) which is located at the Litho centre. It is assumed that only one coil at specified period time can enter the Litho centre. This period time is called *loading rate*. In the High-Bay distribution warehouse of 440

racks capacity, coils are stored using special automated handling equipment. Apart from those coils delivered from the Hot-Line Shop-floor, High-Bay stores coils to be annealed and coils to be passed through the cold rolling mill. One out of four coils (after the cold rolling) passes an inspection control and returns back to the High-Bay. Due to spacing restriction and several other conditions special computer software is used to manage all these progressions. This software keeps a list of all stored coils and makes a decision of the next process using a sophisticated communication protocol. Thus, the nature of High-Bay design requires the manipulation of large amount of data and it is often an iterative process that forces the production manager to go through the different decision making phases several times before reaching the final decision. For modelling convenience, coils of the same characteristics are grouped together according to order types. However, if differentiation must take place, an external decision maker can adjust any priority list.

The next step for a coil that has been transferred into the High-Bay is annealing. Coils are batch annealed in groups of four and in some cases in groups of three following the grouping decision made by the computer. Grouping is made according to width. There are three different ranges for coils' width. These are: (i) Slabs that have range less than 1150mm, (ii) those with width between 1150 and 1380mm, and (iii) Slabs with a range greater than 1380mm. Annealing time also varies according to coils' width and is 8,5 hours for narrower coils and 10,5 hours for wider. In some cases of special customer demand annealing cycle can be up to 12 hours long. After strip coils achieve the required mechanical properties during annealing they return to High-Bay and remain there for about 72 hours. There are three annealing furnaces installed on the plant operating in parallel. Two of these operate in the same way while the third is a newer annealing furnace and is used for higher cycle annealing processes. There is also an optional annealing area located in Strip Mill which is used for approximately 10 loads a week. Breakdown cases are not remarkable and maintenance of annealing furnaces takes place one week in August and one week in September.

The next process after the coil has been cooled inside the High-Bay following annealing is cold rolling. The cold rolling mill allows the production of plates with low gauges and special tempers. Cold rolling process reduces each time the thickness of the plate by roughly 50%. That means that after the first pass the thickness of the plate becomes 1.3mm, after the second 0.65mm, after the third 0.27-0.28mm or 0.38mm according to coil sizes and characteristics and finally for special customer needs a fourth pass halves the thickness to 0.140mm (10% of entire production). The time needed for each rolling depends on the speed of rolls and can vary from 10 minutes to 30 minutes. After each cold rolling pass the plate is led back to High-Bay (unless an off-line inspection takes place) where it remains for 24 hours to cool down to room temperature. After cold rolling differentiations end, the plate is finally assigned to a specific customer order.

6.2.3 Phase 3: Levelling process and quality control

The third stage is known as the BWG and quality control stage. On this stage coils leave the High-Bay storage area and enter the tension levelling process in order to achieve the highest flatness and strip surface quality. The levelling process is divided in three main stages; side trimming, stretching and degreasing. All these stages occur in a single process line. Each coil is trimmed to the manually set of width ordered by the customer, gap and overlap setup for tight width tolerances and minimum edge burr. This process lasts 15 minutes. Then the coil is flattened by the stretching machine which bends up and down over the interrupting arcs of upper and lower sets of processed long slender strip. Finally, each coil is degreased to remove residual rolling oils and lubricants. After tension levelling, each coil passes from surface inspection and quality control to ensure that its thickness gauge and width is accurate. BWG times vary from 35-70 minutes according to the final gauge given to coil in cold rolling process.

Then coils are placed in the storage area (inventory). There is no set point for inventory level although there are restrictions according to the number of racks.

There are 455 racks and 85 floor locations so on the final storage area approximately 540 coils can be stored. In case where a coil is rejected by the inspection area, it is stored temporarily for four days and rechecked for its dimensional accuracy. Inspection time is 5-7 minutes. If, after the second inspection there is still deviation, rejection is the irrevocable decision. In addition to this, certain specific coil products are taken to relaxation before the inspection control. What stresses the importance of push production is that the whole process never stops even if the storage area is full.

6.3 Modelling issues for the Bridgnorth Aluminium production process

6.3.1 Introduction to production process modelling

Previous description of the manufacturing process suggests a repetitive flow process consisting of a series of machines separated by buffers of finite capacity. A slab flows from outside the plant to the first machine, then to the first buffer, then to the second machine, then to the second buffer and so forth, to the last machine, after which it exits - as a finished lithographic coil product - the production line. Processing a specific type of product on a given machine requires a fixed amount of time (*processing time*). In Bridgnorth Aluminium production environment, processing times on different machines are not equal and furthermore different end products being processed in the same machine need different processing times. The production layout is similar to asynchronous industrial processes where there is an unlimited amount of semi-finished products at the input of the process and an unlimited amount of spaces at the output [AS93], [Mar97].

Generally the design of a production model for a specific group of processes commences from assumptions regarding the operational relationships between the production variables. The inflexibility or constraints inherent in this model

determine the maximum delivery responsiveness which is strongly associated with the throughput and what can be achieved by any production model or control system. However, achieving this maximum delivery flexibility is generally unrealistic because information processing, machinery or buffer allocation and decision making itself takes time. The above limitations appear often in asynchronous production systems where operational relationships are always at the heart of the design process.

Asynchronous manufacturing systems are characterised often by complexity and limitations in terms of modelling and simulation. Therefore, managers prefer to develop an appropriate synchronous model that can be used as an approximation of the behaviour of an asynchronous model. The synchronous model forces events (beginning and end of processing, transfer of parts from one buffer to the next one, breakdowns, maintenance etc.) to occur only at times that are multiples of the common processing time step. On the other hand, the transformation of the asynchronous model into the discrete flow model is obtained by approximating the continuous flow of products by a discrete flow. As an immediate cause of this, synchronous modelling is then transferred to a task that focuses on capturing "snapshots" of the production process in each time step. These snapshots are assumed to be *states* of the system while the changes from one state to another can be found in literature as *transitions*.

Another common problem in manufacturing plants is the sequencing problem occurring whenever there is a choice as to the order in which a number of tasks that can be preformed. Sequencing problems in Bridgnorth Aluminium plant have great impact to the throughput. Thus, the structure of the simulation tool must encompass hierarchically all the activities that result to maximum throughput. Sequencing problems are also associated with job priorities. The concept of priority is inherent in many manufacturing plants. A priority is a numerical attribute of an activity or operation on which a unique selection is made so that two competing activities should never have precisely the same value of priority.

Numerous theoretical foundations have been developed to model and control

dynamical systems whose evolution is determined by the occurrence of discrete events rather than the values of continuous variables [CL99], [DHP+93]. A common framework that allows formal modelling and verification of manufacturing systems is the theory of finite state machines (FSM), which is described in Chapter 2. FSM is a mathematical model which serves as an approximation to physical or abstract phenomena. The theory of FSM assumes that a system can be a finite set of conditions called states, the system behaviour within a given state is identical and it is described by states for significant periods of time. Another descriptive term of FSM theory is that a system may change states only through a finite number of transitions which are the response of the system to external or internal events and take (approximately) zero time.

In this thesis we concentrate our work on the second phase of production plant system. As previous description suggests, Bridgnorth Aluminium production system is built up of many processes working in parallel and can be viewed as a finite set of resources, i.e., the machining equipment, shared between a set of products to be manufactured. The developed simulation model is based on the well-known MATLAB software [oTC], which is selected as the underlying environment because of its various unique advantages: High programming efficiency, strong numerical algorithm support, friendly interface, elegant Graphical User Interface (GUI), and most of all, its open architecture making it possible to customise, expand and integrate complicated system models and control schemes to meet special requirements as those faced in the present model.

6.3.2 Software tool description

In order to resemble all the activities taking place in the second phase we need to associate each single task with an appropriate function defined in the MATLAB environment. These functions are very similar to states deemed in FSM theory and they have as input variables the current state while output variables represent the updated state. Apart from updating the state variables, functions must also conform

with the global clock. As it has been stated before, the main problem arising in simulation methods is how realistic is the simulation model in relation to the real life production attempted to be modelled. Accordingly, to achieve more realistic applications we must first reformulate the theoretical model to allow for time as an explicit variable. In fact, since we assume that the production system is operating synchronously, the global clock is updated in each simulation step (defined also as event). The default simulation step is chosen to be 5 minutes, which is sufficiently (two times) faster than the the fastest time constant among all processes.

Semi-finished coil products being processed in the Litho centre have different characteristics and are associated with specific customer orders. For better modelling convenience, coils have been categorised in eighteen different types according to their width, final gauge (specified in cold rolling machine) and if they are targeted at the special orders (i.e., Fuji). There are three types of width (*Wide*, *Medium*, *Narrow*) and also three different types of final gauge (*Thick* (0,38mm), *Standard* (0,28mm), *Thin* (0,14mm)). These data are stored in MATLAB structures as an input file for maximum flexibility and to improve the readability of the code. Structures are MATLAB arrays with named "data containers" called fields. Like standard arrays, structures are inherently array oriented. Element types are defined using an object oriented design so that new elements can be added easily and without necessitating changes in the core analysis code. Table 6.1 shows the 18 different coil types. A structured array which corresponds to the first coil type is given below.

```
slab_type_1 =
struct('type','type_01','width','wide','special_flag',0,'gauge','thic','timer',-1,'flags',[0
0 0 0]);
```

There are six different fields containing all the information needed for the simulation purposes. The field *type* specifies the type of the coil (type 1, type 2, etc.), *width* the width of the coil (Wide, Medium or Narrow), *special flag* is an integer (0 or 1) which determines if the coil is intended for special treatment according to customer orders (*special flag*=1) or not (*special flag*=0), *gauge* the final gauge of the coil (Thick, Standard or Thin), *timer* holds the time of last entry of coil in High-Bay

(timer=-1 if coil is outside High-Bay), and finally *flags* is an 1x4 array whose second element indicates how many times the coil has been annealed, and third element how many times the coil has been cold rolled. The first and fourth element can be used for future modelling issues (e.g., inspection, BWG, or processes taking place outside High-Bay area). Note that the strings in fields may contain exactly four characters and this is due to modelling convenience while MATLAB handles the data stored in structures.

Table 6.1: The 18 different coil types

Coil type	Width	Gauge	Special Product
1	Wide	Thick	No
2	Medium	Thick	No
3	Narrow	Thick	No
4	Wide	Standard	No
5	Medium	Standard	No
6	Narrow	Standard	No
7	Wide	Thin	No
8	Medium	Thin	No
9	Narrow	Thin	No
10	Wide	Thick	Yes
11	Medium	Thick	Yes
12	Narrow	Thick	Yes
13	Wide	Standard	Yes
14	Medium	Standard	Yes
15	Narrow	Standard	Yes
16	Wide	Thin	Yes
17	Medium	Thin	Yes
18	Narrow	Thin	Yes

The hierarchical structure of the simulation flow system is illustrated in Figure 6.2. As it can be inferred by Figure 6.2, the only interaction between user and tool is the choice of simulation parameters in GUI.

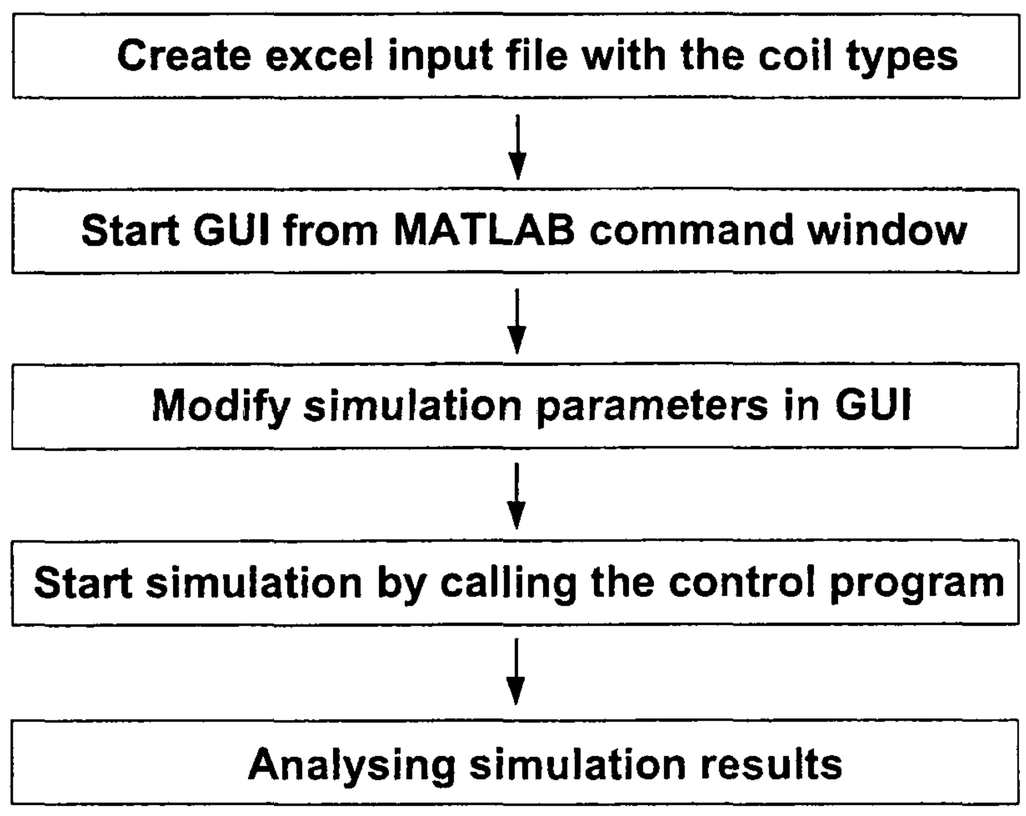

Figure 6.2: Hierarchical structure of the simulation flow

The software tool benefits from MATLAB is the ability of reading data from Microsoft Excel spreadsheets. The user initially creates an input file in Microsoft excel worksheet (.xls). This file has as many rows as the simulation days while columns contain the coils entering the Litho centre. Then, by using the command `xlsread`, the software tool returns numeric data in array form, from the worksheet. A typical worksheet for simulation period of seven days is presented in the Table 6.2. The odd columns (1st, 3rd, ...) represent the type of coils while the even (2nd, 4th, ...) the number of coils of the left adjacent column elements.

Table 6.2: Excel worksheet input file with coil types

	Type	No	Type	No	Type	No	Type	No	Type	No	Type	No
Day1	6	8	9	6	19	4	6	4	6	24	9	10
Day2	4	3	4	9	8	3	8	5	8	4		
Day3	2	4	5	4	5	22						
Day4	6	8	8	3	6	3	6	10				
Day5	5	8	6	14	1	20						
Day6	6	7	5	4	5	4	8	4				
Day7	6	4	9	6								

The need of handling data with the simulation going on causes the volume of stored data to steadily increase, which in turn decelerates the simulation process. Thus, the software tool must be able to handle the data attentively by providing at the same time simulation speed. To overcome this problem we have created several functions. These MATLAB functions take as inputs the variables defined by the user, compute the required results using user's input and then pass those result back to the user. The commands evaluated by these functions, as well as any intermediate variables created by those commands are hidden. These functions are stored as M-files and are very similar to script files with the only difference that these functions communicate with MATLAB workspace only through the variables passed to them and through the output variables they create. All the developed MATLAB code lines used in functions, are presented in the Appendix B.

Software tool implements also buffers in order to manipulate more accurately the pre and post conditions of each process in the High-Bay. Buffer sizing and allocation is a major concern faced by Bridgnorth Aluminium production line since the distribution of the coils through the various tasks in High-Bay is carried out by

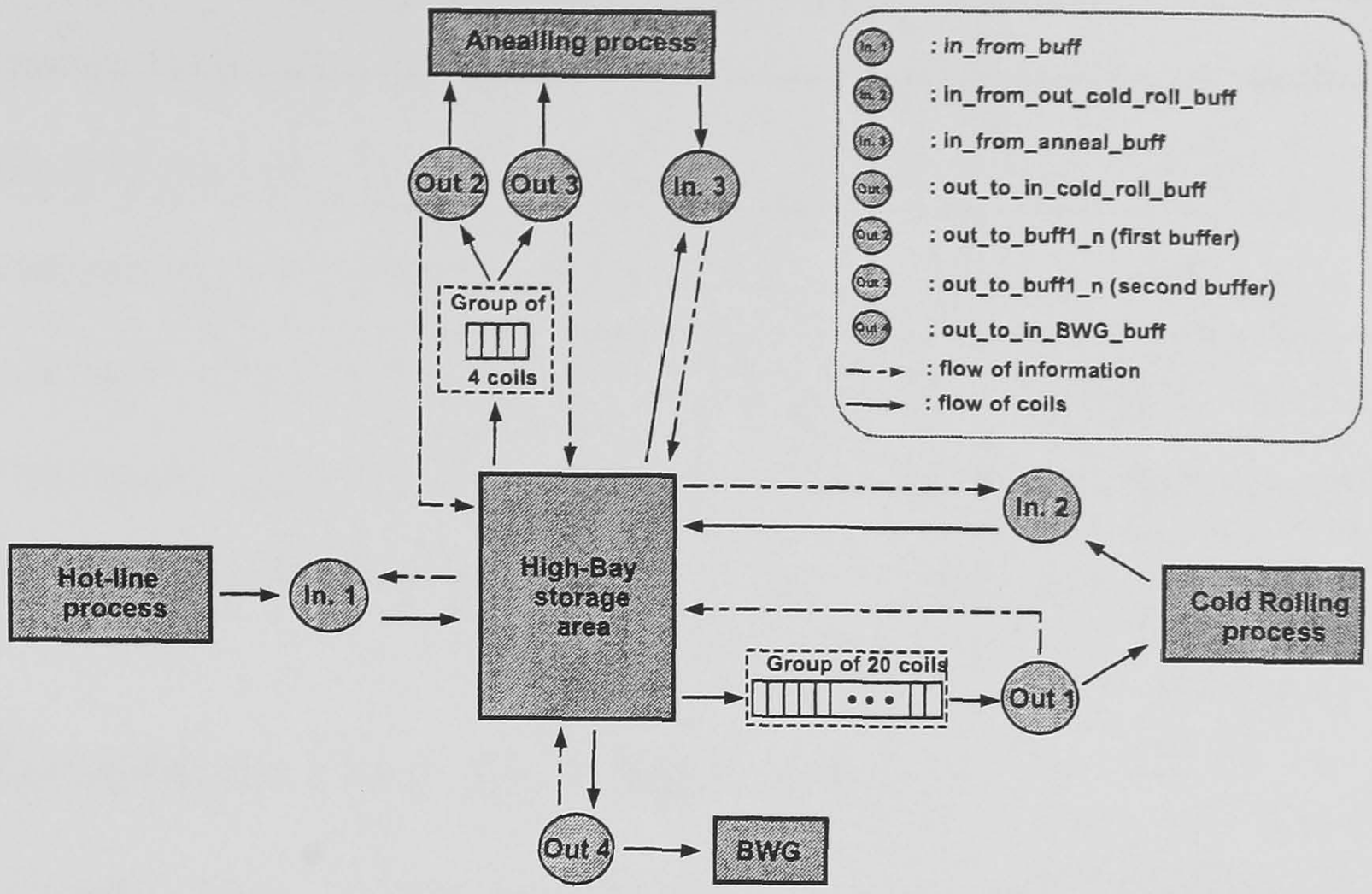

Figure 6.3: Buffer allocation in Litho centre

the crane. While the crane can perform a certain amount of movements in a given time, the software tool must be able to capture all the potential movements and update the input and output buffers in each simulation step according to certain criteria (e.g., size of buffers, priorities and engaged machines).

In this simulation tool it is assumed that there are seven buffers outside the High-Bay storage area. One buffer is between the Hot-Line and the High-Bay (*In.1*), three buffers between High-Bay annealing machines (*In.3*, *Out.2* and *Out.3*), two buffers between High-Bay and cold rolling machine (*In.2* and *Out.1*) and one buffer before BWG (*Out.4*). The buffer allocation in Litho centre is depicted in Figure 6.3. The names of buffers in the top-right legend are referred to the High-Bay storage area and are associated with the function that updates the state of the High-Bay `update_highbay_new`. For modelling convenience, the simulation tool implements also several virtual buffers which are clearly described in section 6.3.4.

During the simulation flow the user can benefit by monitoring the simulation process with the aid of two windows. The first window shows dynamically the state of the High-Bay and the condition of both cold rolling and annealing machines. The second one depicts the changes taking place in input and output buffers of the

High-Bay. The user may decide if any window will appear during the simulation process or not in GUI. The simulation also provides some useful statistics at the end of the simulation period. These statistics include the average crane movements per hour, the percentage of annealing furnace occupancy, the percentage of cold rolling machine occupancy, the percentage of BWG occupancy and finally the percentage of High-Bay occupancy in terms of capacity changes. Simulation windows and statistics are described in case study which is performed in section 6.4.

6.3.3 Graphical User Interface (GUI)

The GUI provides a user-friendly access for using the developed simulation system. The main window is displayed in Figure 6.4. Initiated from MATLAB command window, the GUI handles the whole process of simulation, from creating a new simulation model, starting the simulation and changing the simulation parameters, priorities and simulation options which are depicted in three different fields (rectangles).

More specifically, in the first field (Parameters) the user can (pre)set the number of annealing machines, cooling time after annealing while coils remain in High-Bay, the time which specifies the rate coils entering the High-Bay and finally the maximum movements of crane hourly. In the second field (Priorities) the user can determine the priorities of four different processes (annealing, cold-rolling (TSM), BWG, and the coil entry to Litho centre) which determine the potentially movements of crane. Finally, the GUI provides the convenience to the user of selecting an input file, setting initial conditions, the choice of plot types during simulation and definition of number of simulation days. Thus, it is possible to set the simulation system to any desirable initial state and start a new simulation task representing the real scheduling process or even assume maintenance or inactivity periods. MATLAB graphical display gives also to the user the flexibility of adding different simulation parameters or changing default parameters such as cold rolling or annealing process times.

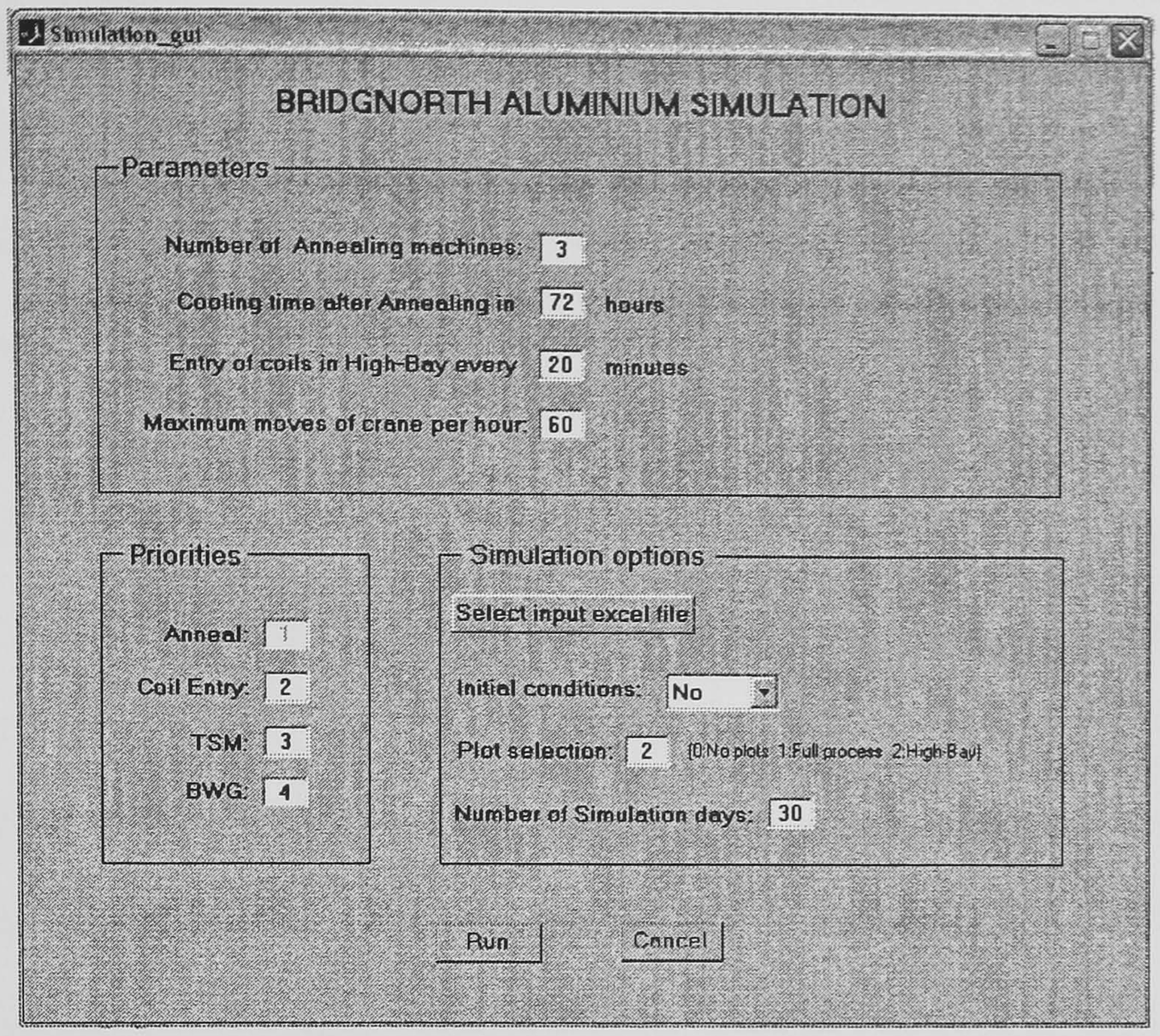

Figure 6.4: Graphical User Interface for the simulation tool

6.3.4 Implementation of software tool

Due to high complexity of the Bridgnorth Aluminium production process, more than 7000 lines of MATLAB code have been written (equivalent to more than 20000 lines in any high or low level programming language e.g., C, C++). Additionally, 41 different functions have been developed to capture all individual tasks in detail. In order to avoid any wearisomeness in presenting all the functions used in the software tool a selective choice has been made to those functions which are considered as more useful and illustrate the main idea of the simulation tool. All these functions are called in the main program routine `run_process.m` to be initiated from GUI after the user click on *Run* button, which executes the whole process of simulation. In this file some default process parameters are defined together with input and initial conditions, states updating, plotting attributes and statistics table updating.

In order to avoid having the input excel spreadsheet files with all the coils enter the plant in the same directory with the main programme, a separate directory *data*

is created. This directory saves all the data history of the input coils created by the user and can be modified easily at the beginning of the file `run_process.m`. Apart from those parameters shown in the GUI the user can also change other parameters which are defined in function `default_parameters`. These parameters include the simulation step (in minutes), the simulation time (in hours), High-Bay capacity, the capacity of all the buffers shown in Figure 6.3, the cooling time after cold rolling and the choice of presenting or not the statistics at the end of the simulation process.

In case where the user runs the software tool without initial conditions all states and buffers are set initially to zero. Otherwise, the state and buffer conditions contain the coils of the previous simulation process which were saved at the end of the simulation. The main idea of the simulation tool as stated before is to manipulate a list created by those coils entered by the user. This list of coils after having entered the High-Bay storage area, represents the High-Bay condition and it is updated in each simulation step. This list is similar to a stack array (new coils insert the stack to the bottom while the oldest are on the top). This means that every time the software must choose which coils should leave the High-Bay either for annealing or cold rolling treatment or to BWG, the list is read commencing from the top.

The m-file `run_process.m` comprises the flow of coils and all the procedures in Litho centre as they have been described in section 6.2.2. The coils enter the High-Bay from Hot-Line with a pre-specified loading rate via the buffer `in_from_buff`. The coils which are intended to enter the High-Bay area are subject to crane availability and to the High-Bay current capacity. Next the software updates the High-Bay state and outputs, by checking the number of coils inside the High-Bay storage area at the beginning of the current time interval and on whether a list of coils for cold-rolling has been identified in an earlier step. The main idea of updating the High-bay is to "load out" first the chosen coils to output buffers and then to "load in" those coils located in input buffers and can be entered into the High-Bay area. Hence, the following rule sequence occurs. It is determined which of the cold-rolled coils have cooled so they can be moved to `out_to_in_BWG_buff`, which coils need

Table 6.3: BWG times in minutes for the 18 different coil types

Gauge		
Thick	Standard	Thin
35	40	70

to be loaded to annealing machine through the buffer `out_to_buff1_n`, and also which of the coils that they have been annealed have cooled so they can move to `out_to_in_cold_roll_buff`. Then, the software defines which coils can be entered from annealing furnaces via buffer `in_from_anneal_buff`, which coils have finished cold rolling and are situated in buffer `in_from_out_cold_roll_buff`, and finally which coils can be entered from Hot-Line via the buffer `in_from_buff`. Next the computer programme simulates the activity of BWG. First it updates the state and output of the buffer `update_in_BWG_buff` which is located between High-Bay and BWG, and then the state and output of BWG process. All the finished coils are collected in an output buffer in a list form which constitutes the output of the manufacturing system. BWG times are shown in Table 6.3.

After updating the BWG process the software updates the state of the buffer `update_buff1_n` which is located between High-Bay and annealing machines. In case where one or more annealing furnaces are empty, the software tool reads the list of coils in High-Bay which are intended for annealing treatment and checks if there are four coils of the same characteristics and if there are it creates a group (quartet of coils). This quartet must have four coils that have not been annealed, have the same width and they are intended (or not) for special products. This information can be easily obtained from the MATLAB structures and more specifically from the field *width*, the field *special_flag* and the second number of the array in the field *flag*. Then the state of all the annealing machines is being updated and those coils that have completed the annealing process are first placed into the buffer `in_from_anneal_buff` and then are led back with the aid of the crane to the High-Bay storage area for cooling. Table 6.4 shows the annealing times for all the eighteen different coil types.

Table 6.4: Annealing times in hours for the 18 different coil types

Width	**Standard Products**	**Special Products**
Wide	10,75	12,75
Medium	9,75	12,25
Narrow	8,75	11,25

Table 6.5: Cold rolling times in minutes for the 18 coil types (4 groups)

Width	Final Gauge											
	Thick			Stan			Thin			
	Pass 1	Pass 2	Pass 3	Pass 1	Pass 2	Pass 3	Pass 1	Pass 2	Pass 3	Pass 4
Wide	10	15	20	10	15	20	10	15	20	25
Medium	10	15	20	NA	15	20	NA	15	20	25
Narrow	10	15	20	NA	15	20	NA	15	20	25

Finally, the computer programme simulates the cold rolling process, by updating first the buffer `out_to_in_cold_roll_buff`. This buffer loads a coil that needs to pass cold rolling process. The software tool creates a list of twenty or forty grouped coils according to their width and final gauge. This information is provided by the fields *width* and *gauge* in MATLAB structures. Every time the cold rolling process has been completed the coil is loaded to buffer `in_from_out_cold_roll_buff` and then is led back to the High-Bay storage area for cooling. There are overall four different groups with different rolling times in minutes as Table 6.5 indicates. The first group consists of twenty coils whose final gauge is *Thick* and those coils whose final gauge is *Standard* and their width is *Wide*. The second group can have forty coils of *Standard* final gauge and *Medium* or *Narrow* width. The third group may have twenty coils with final gauge *Thin* and width *Wide*. Finally, the fourth group is made of twenty coils of *Thin* final gauge and *Medium* or *Narrow* width. Note that times in cold rolling machine are irrelevant to special products.

All above activities can be fulfilled provided that buffers' capacity are not exceeded and that there are sufficient crane movements in each simulation step. Note that all the assignment decisions are made in the same order that they would be implemented in the actual process. This order is specified in GUI by selecting the Priorities sequences. All the code written in MATLAB is given in Appendix C.

6.4 Modelling cases and simulation performance

In this section four main scenarios are conducted to test the effectiveness of the developed simulation tool and at the same time to bring out the most important results. Simulation provides the means of testing the behaviour of the system, using specific metrics and by implementing a number of different scenarios. Since this is, essentially, a discrete event simulation it tracks the sequence of events in the Bridgnorth production system in discrete time steps. Several MATLAB functions were used in order to gather and manipulate specific model data while final reports and figures generated after each simulation present quantitative information about important aspects of model (machine occupancy, number of cranes moves etc.) together with statistical data based on certain metrics which can be used to evaluate the model's performance.

The proposed simulation model allows also the investigation of the dynamic behaviour of the production system under study and it provides a significant insight of the system's characteristics. Since the overall model has been implemented in MATLAB code, the model can be easily modified as required in order to meet alternative design specifications. Apart from being flexible, the graphical interface of MATLAB with the aid of plotting schemes during simulation provides excellent visual representation of the state evolution of the production system.

Figure 6.5 depicts a current state - 300th iteration (1 day and 1 hour) - of all the activities taking place in the production system. The first horizontal bar represents those coils located in the Hot-Line and are due to enter the High-Bay storage area. The second bar shows coils entering the High-Bay at a specified loading rate, initially set to 20 minutes, while the third bar represents coils currently stored inside the High-Bay storage area. These coils are grouped according to the several stages of treatment they have undergone (annealing, cold-rolling etc.). The fourth, fifth, sixth, seventh

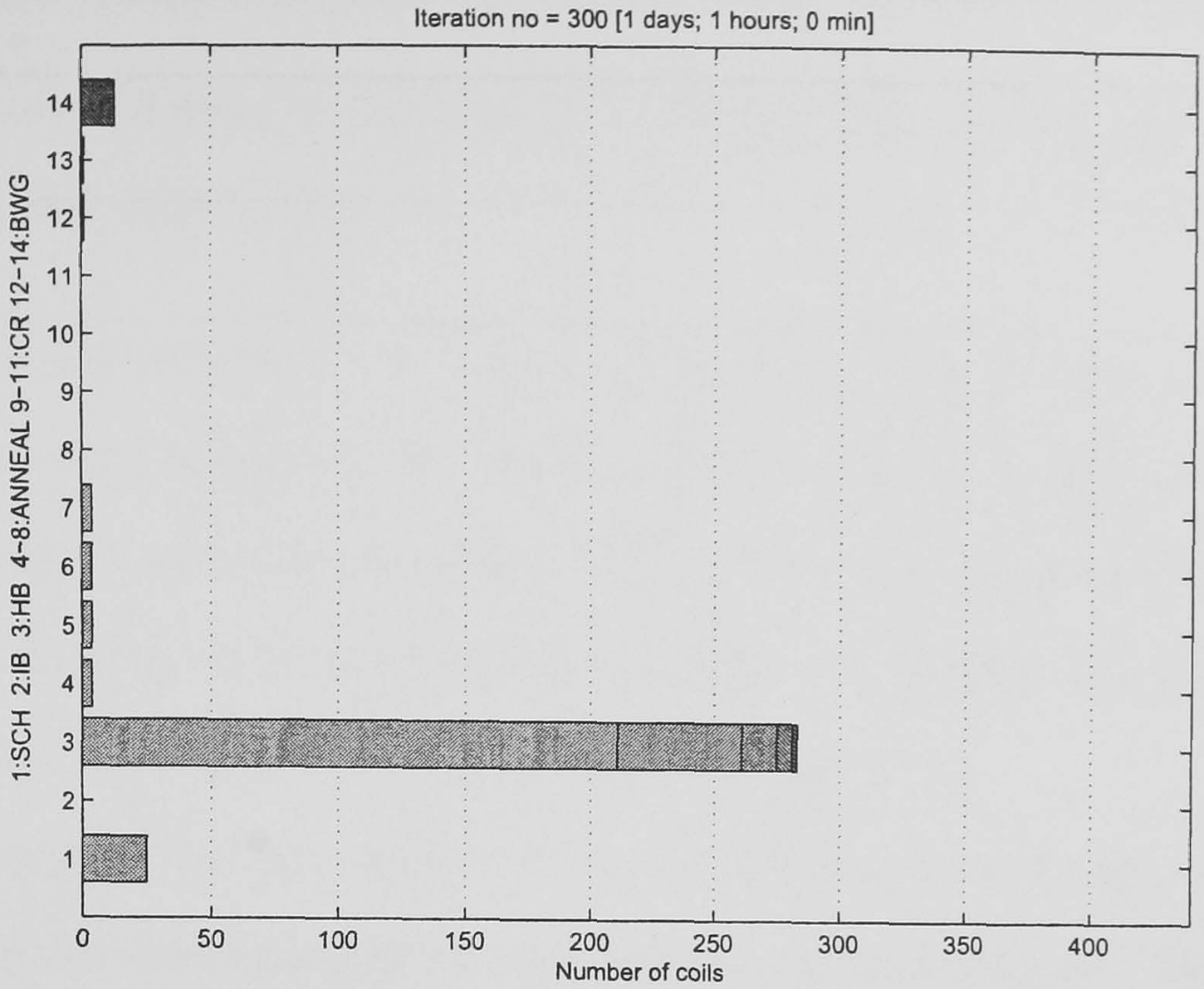

Figure 6.5: A snapshot of all High-Bay activities during a simulation run

and eighth bar charts represent the annealing stage. More specifically the fourth and eighth bar depict the input and output annealing buffer, respectively, while the fifth, sixth and seventh depict the current state of the three annealing machines. In a similar way the ninth, tenth and eleventh bars illustrate the cold-rolling activity, while the twelfth and thirteenth bars the corresponding BWG activity. Finally, the fourteenth bar represents the finished products that have undergone BWG treatment and have exited the Litho centre.

Simulation runs allow the visualisation of the coils' flow, grouping, and variations in intermediate buffer and queue sizes. Moreover, the interrelations between machine centres and related tasks are depicted efficiently as well as machine utilisation at each simulation step. The performance of the system was assessed using a number of defined metrics such as:

- *Throughput rate*, defined as the number of coils that have been completely treated in a specified time period.

- *Machine Usage*, defined as the percentage of machine centres and High-Bay

storage area busy over time.

- *Crane moves*, defines as the number of crane movements in each machine centre location and coils' entry.

The model was validated on actual system data, provided by the Bridgnorth aluminium company, with respect to buffer sizes, throughput rates, crane moves and inventories. The proposed model was deadlock-free, as it was found after extensive simulations. This can be ensured in the programme by limiting the total number of coils circulating in the High-Bay area and its feedback loop.

In order to gain further insight of the production system, the following four different scenarios were investigated via extensive simulation runs. These scenarios include:

1. Scenario 1: Simulation based on current production plant settings and layout.

2. Scenario 2: Installing of an additional similar annealing machine operating in parallel with other three.

3. Scenario 3: Reducing pre-set times the coils remain in the High-Bay during cooling and after annealing.

4. Scenario 4: The effect of order fluctuations based on current setup of the production plant.

For better evaluation and comparison of all above scenarios, it is assumed throughout the simulations that initially the High-Bay storage area is empty and that no machine process is initiated before the start of the simulation. It is also assumed that the loading rate of coils entering the High-Bay storage area is one every 20 minutes, while the maximum crane movements are limited to 60 per hour. The priorities of crane service to different locations are set as follows. First priority is given to the annealing process, second to cold-rolling, third to BWG and finally coil entry to the High-Bay activity is served last. Crane movements are executed in

this order until either no additional move is possible during each step or maximum number of crane moves per step is reached. Simulations of the first three scenarios were performed for 180 coils entering the High-Bay area over a period of one week. These coils (read from a Microsoft excel worksheet) are depicted in Table 6.6.

Table 6.6: The Excel input file of 168 coils used in simulation runs

	Type	No	Type	No	Type	No	Type	No
Day1	6	8	9	6	6	9	1	4
Day2	6	10	2	4	5	5	3	4
Day3	9	3	6	4	11	4	4	8
Day4	9	12	1	4	13	8	2	4
Day5	6	12	3	4	7	4	10	4
Day6	5	8	6	12	8	4	17	4
Day7	9	3	12	4	15	8	14	4

6.4.1 Scenario 1: Simulation based on current production plant settings and layout

This scenario incorporates the current production plant machinery, settings and layout based on the description given in section 6.2.2. The report for the simulation period of 35 days is given in Appendix C. Simulation results showed that the overall throughput rate was 98 coils in 35 days. Figure 6.6 depicts the changes in throughput rate during the full simulation. Note that despite the fact that 168 coils entered the High-Bay in seven days, only 98 of them converted to finished products while 70 coils (168-98) remained inside the High-Bay storage area due to annealing and cold-rolling grouping restrictions as Table 6.4 and Table 6.5 indicates.

It is clear that the first coil exited the Litho centre in the 19th day, while the throughput rate was increased dramatically between the 19th and 25th day. That happened mostly due to several other activities (coils' grouping, annealing, cold-rolling, cooling procedures etc) that needed to be performed prior to BWG treatment.

Figure 6.7 demonstrates the percentage of annealing furnaces' occupancy over time. These percentage values were more than 99% for the first 29 days highlighting

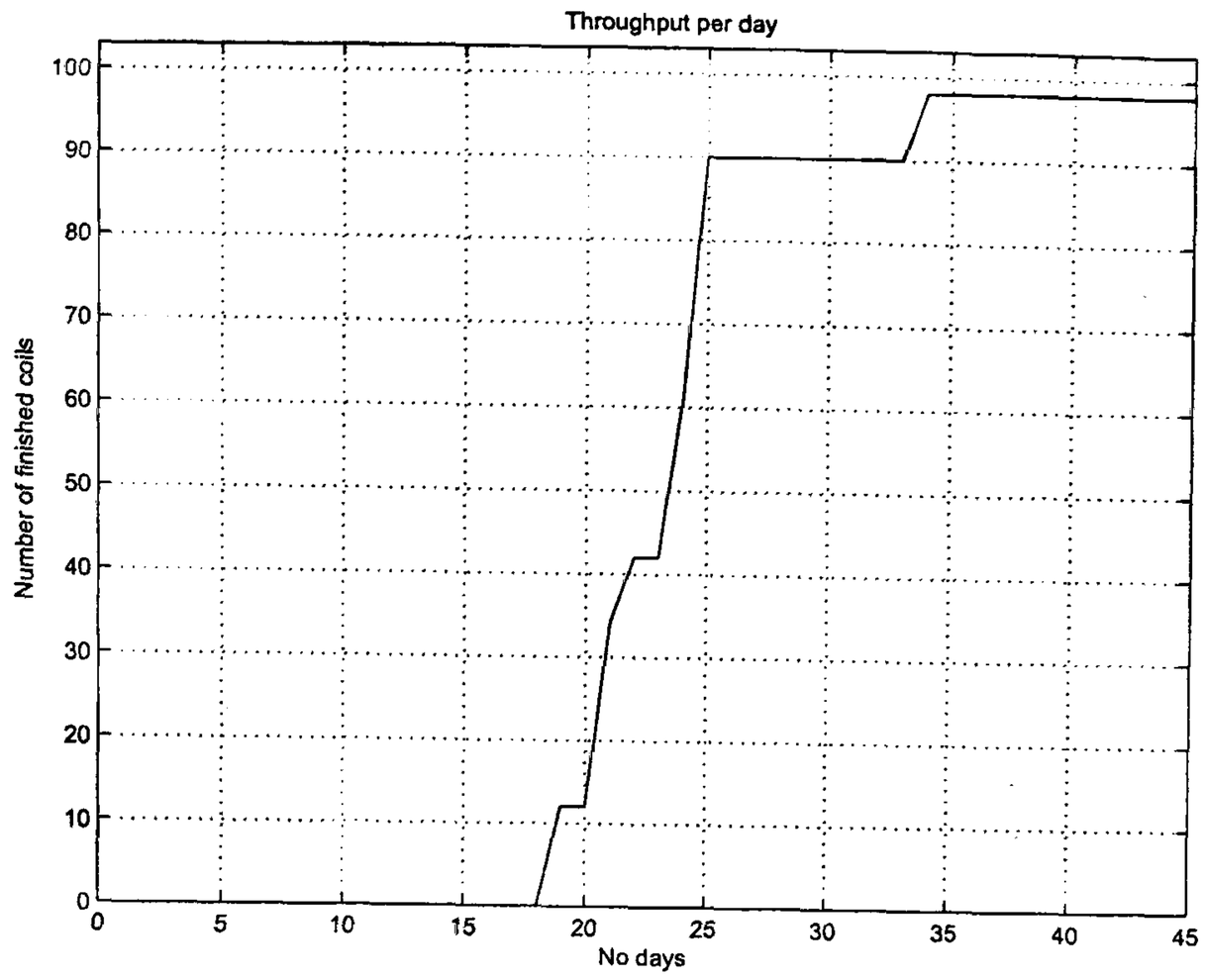

Figure 6.6: Throughput rate for Scenario 1

the fact that at least one annealing furnace each day is fully occupied. According to cold-rolling centre data, Figure 6.8 shows a high usage percentage during the 15th and 24th day. The starting day of cold-rolling (15th) brings out the grouping procedure followed by annealing. Figure 6.8 also attests that the last cold-rolling activity took place on 33rd day which is reasonable since the last annealing cycle was completed on 29th day.

Figure 6.9 illustrates the fact that coils enter the BWG machine centre in batches. More specifically the BWG area is busy only five days during the entire simulation in which all coils passed through the BWG stage, while the last BWG activity took place on the 34th day, as one full day of cooling time is necessary after the last cold-rolling process.

Another useful observation concerning the activities taking place in the Litho centre is reflected by crane movements and the areas served during the coils' transfer. Figure 6.10 illustrates the number of crane movements in total (all services). Simulation runs indicated that the maximum number of crane movements was 14, although the maximum number of potential crane movements per hour was set before the simulation start to 60. Table 6.7 provides information on the total number of

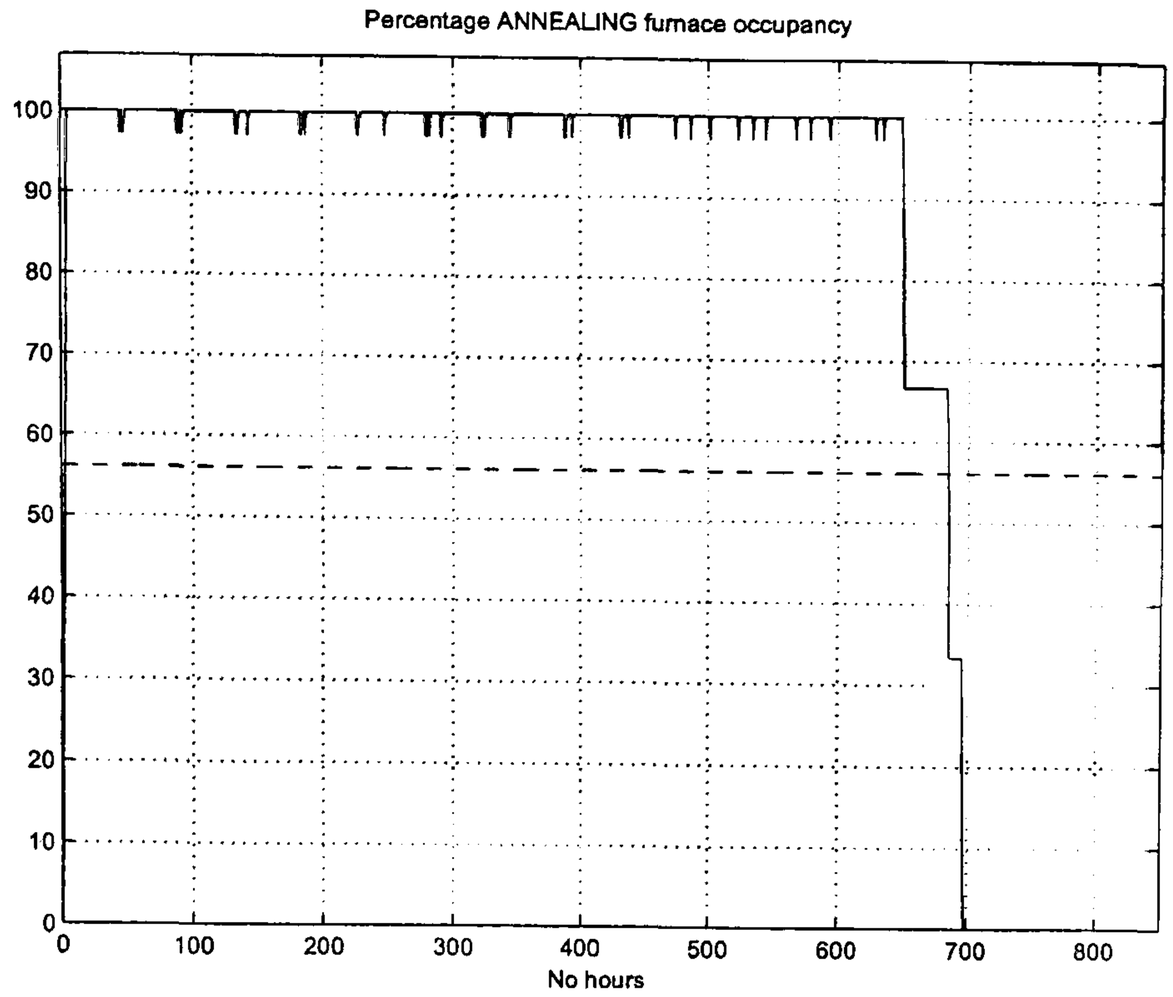

Figure 6.7: Annealing furnaces occupancy for Scenario 1

crane movements in terms of their location throughout the entire simulation. Note that the crane spends its most time to serve the annealing activity while the BWG is the centre where the crane has the minimum number of routes.

Table 6.7: Total crane movements of each location for Scenario 1

	Coil Entry	Annealing machines	Cold-rolling input	Cold-rolling output	BWG machine
Crane movements	168	328	300	300	98

A good indicator of a smooth flow of semi-finished coils through the production system, is the capacity changes in the High-Bay storage area. As stated before, the High-Bay storage area's behaviour is similar to an intermediate buffer where all coils are stored before they are transferred for annealing, cold-rolling or BWG treatment.

Figure 6.11 presents the High-Bay storage area occupancy per hour. It is clear that in the first eight days, the capacity increases as coils enter the storage area from the Hot-Line. Then between the 8th and 16th day the High-Bay varies in capacity (34% of maximum capacity) while after the 19th day the High-Bay storage area capacity starts to decrease until the 26th day as a result of the first coils leaving the High-Bay area for BWG treatment. Between the 27th and 33rd day the number of

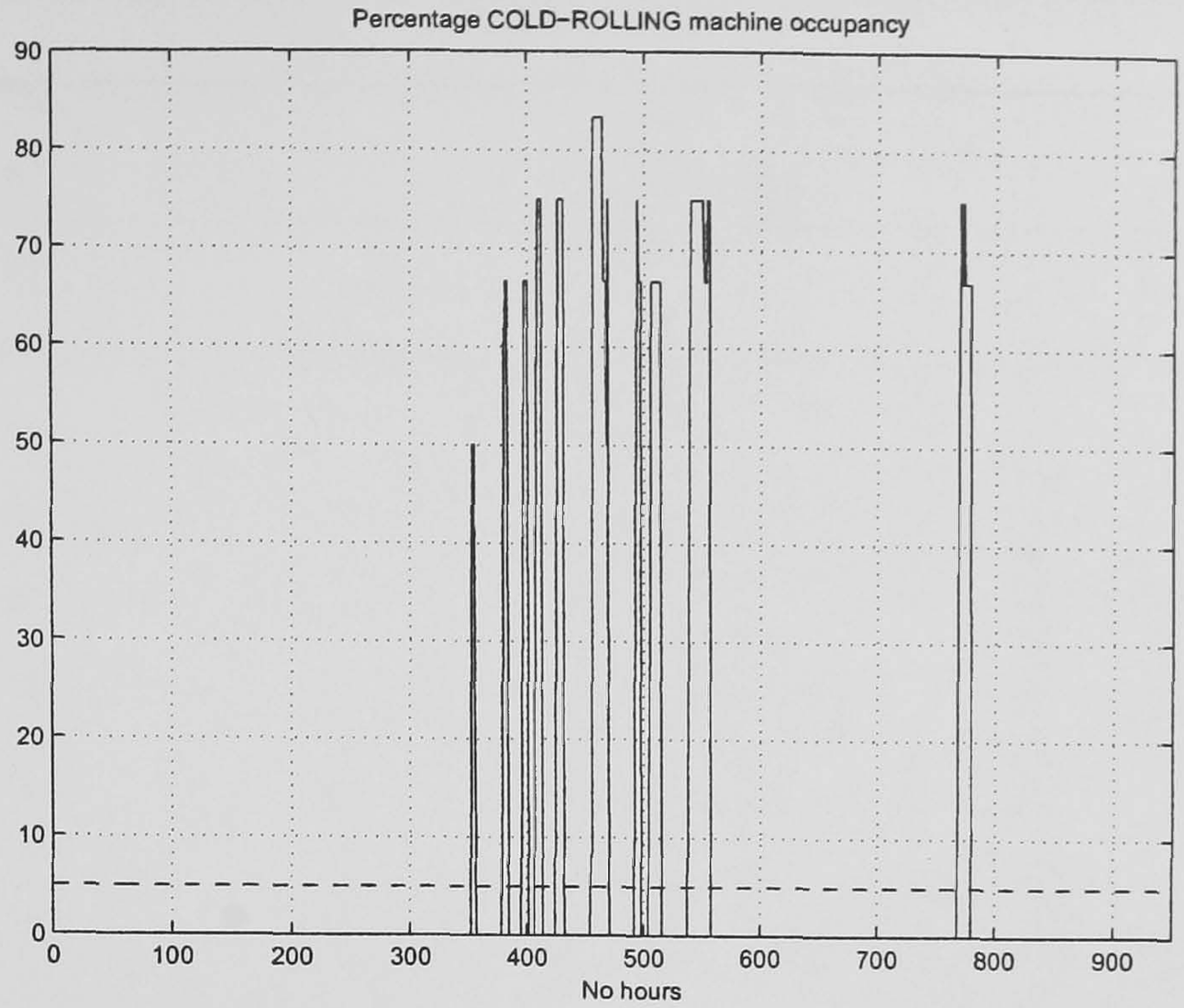

Figure 6.8: Cold-rolling machine usage for Scenario 1

coils stored in the High-Bay storage area increase again since no BWG process is taking place.

Table 6.8: Percentage of each location's average use for Scenario 1

	Annealing	Cold-rolling	BWG	High-Bay storage area
Usage	80.19	7.06	7.68	23.45

Finally, Table 6.8 gives the average use of the following locations: Annealing machine centres, Cold-rolling activity, High-Bay storage area capacity and BWG work centre.

6.4.2 Scenario 2: Installing of an additional annealing machine operating in parallel with the other three

The second scenario examines the case of installing an additional annealing machine in the plant. The report for the simulation period of 29 days is given in Appendix C. Simulation results showed that the total throughput rate is now 109 coils over 29 days. As expected, the throughput rate has increased in comparison with Scenario 1, where the total throughput was found to be 90 coils in 29 days, as depicted

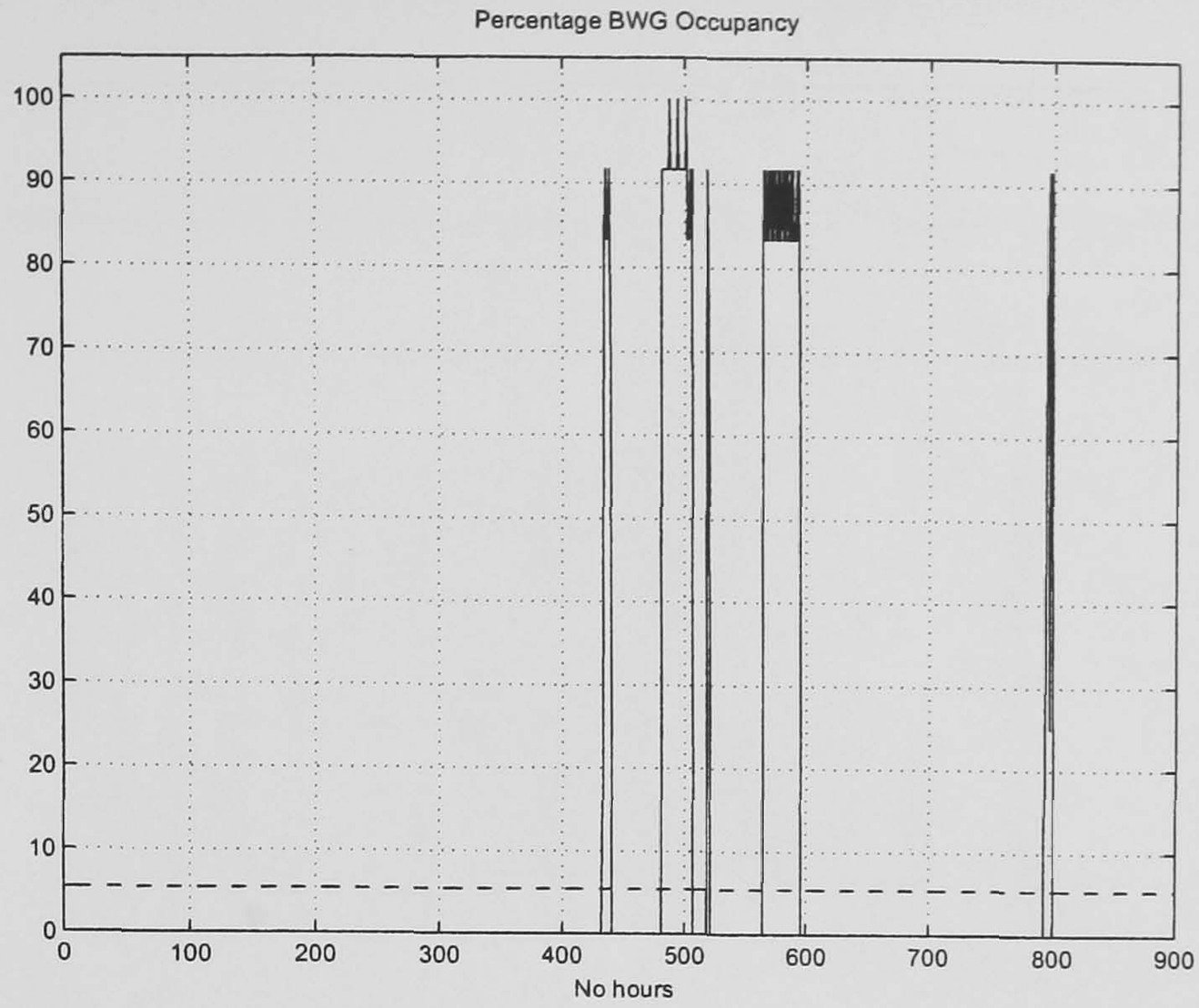

Figure 6.9: BWG occupancy for Scenario 1

in Figure 6.6. Thus installing a fourth annealing furnace increases throughput by 21.11%. Figure 6.12 illustrates the changes in throughput rate for the second scenario. Note again that despite the fact that 168 coils entered the High-Bay over a period of seven days, only 109 of them have left the Litho centre, while 59 coils (168-109) remained as semi-finished products at the end of the simulation due to annealing and cold-rolling grouping restrictions as shown in Table 6.4 and Table 6.5.

The first coil exited the Litho centre on the 15th day, instead of the 19th as in the first scenario while the throughput rate was increased dramatically between the 15th and 28th day. This indicates that the second scenario provided not only faster throughput rates but also a smoother flow of coils through the production line.

Figure 6.13 shows the percentage of annealing furnaces' occupancy over time. These percentage values exceed 99% for the first 21 days highlighting again the fact that at least one annealing furnace each day is fully occupied. Figure 6.14 depicts the cold-rolling machine centre usage, where the main occupancy time occurred between the 12th and 20th day. The last day of cold-rolling activity was performed on the 27th day which is again very reasonable since the last annealing cycle was completed on the 23rd day.

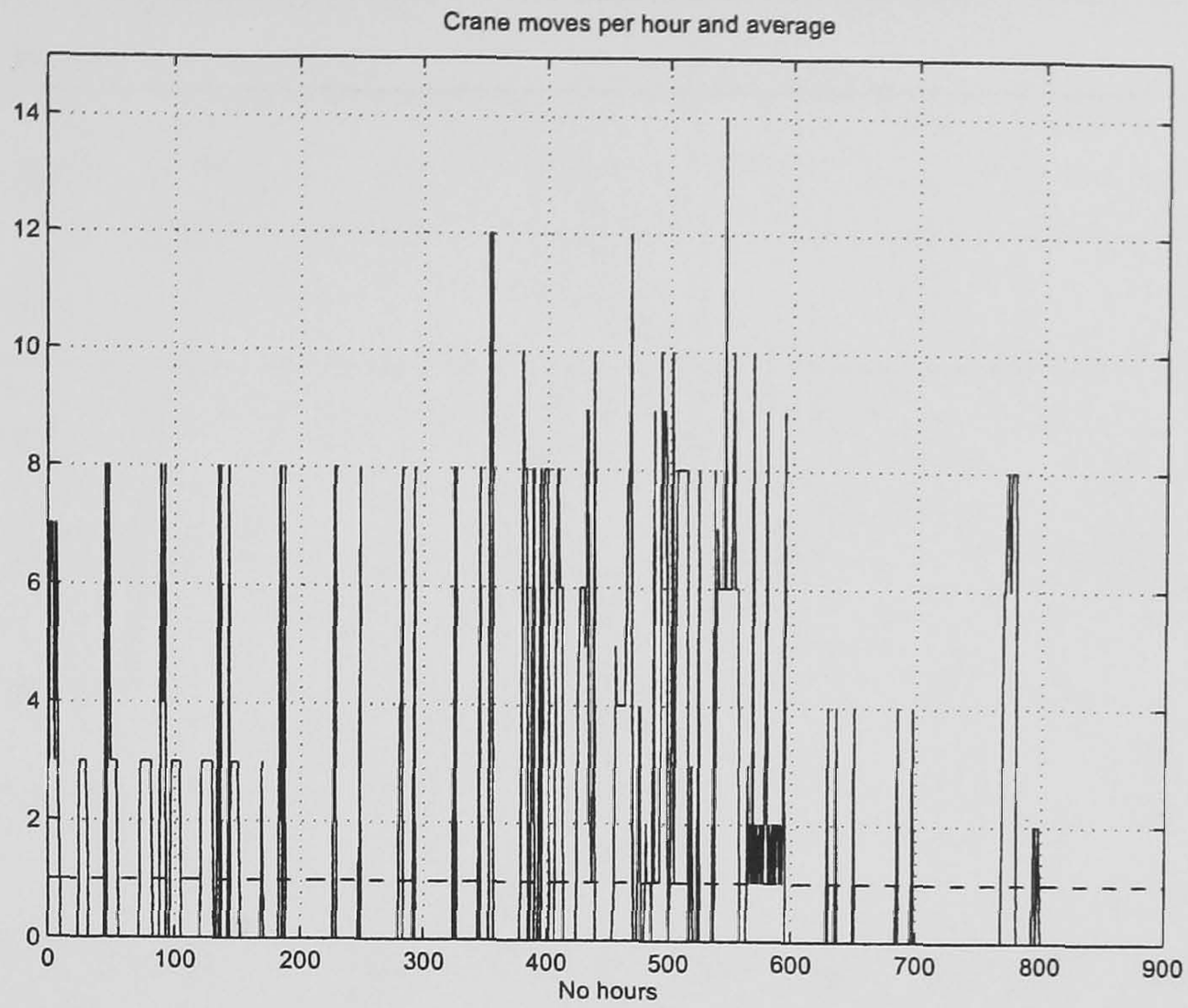

Figure 6.10: Crane movements in total for Scenario 1

Figure 6.15 demonstrates again the fact that coils enter the BWG machine centre in batches. This time BWG is busy for only eight days during the whole simulation run during which all coils pass through the BWG stage, while the last BWG activity took place on the 28th day (one day again after the last cold-rolling process is completed).

Figure 6.16 shows the total number of crane movements at all four locations. Simulation runs indicate that the maximum number of crane movements per hour is 19 in contrast to the first scenario where the maximum number of movements was only 14. This observation reveals the fact that when a fourth annealing machine is installed the crane can serve all the annealing furnaces in the interspace of one hour, provided that the maximum number of potential crane movements per hour is set to 60. Table 6.9 gives information for the total number of crane movements at each location. Note again that the crane spends most time in serving the annealing activity while the BWG is the location which is served for least amount of time than the other three.

Capacity changes per hour in the High-Bay storage area can be viewed in Figure 6.17. Similarly to the first scenario in the first eight days the capacity increases

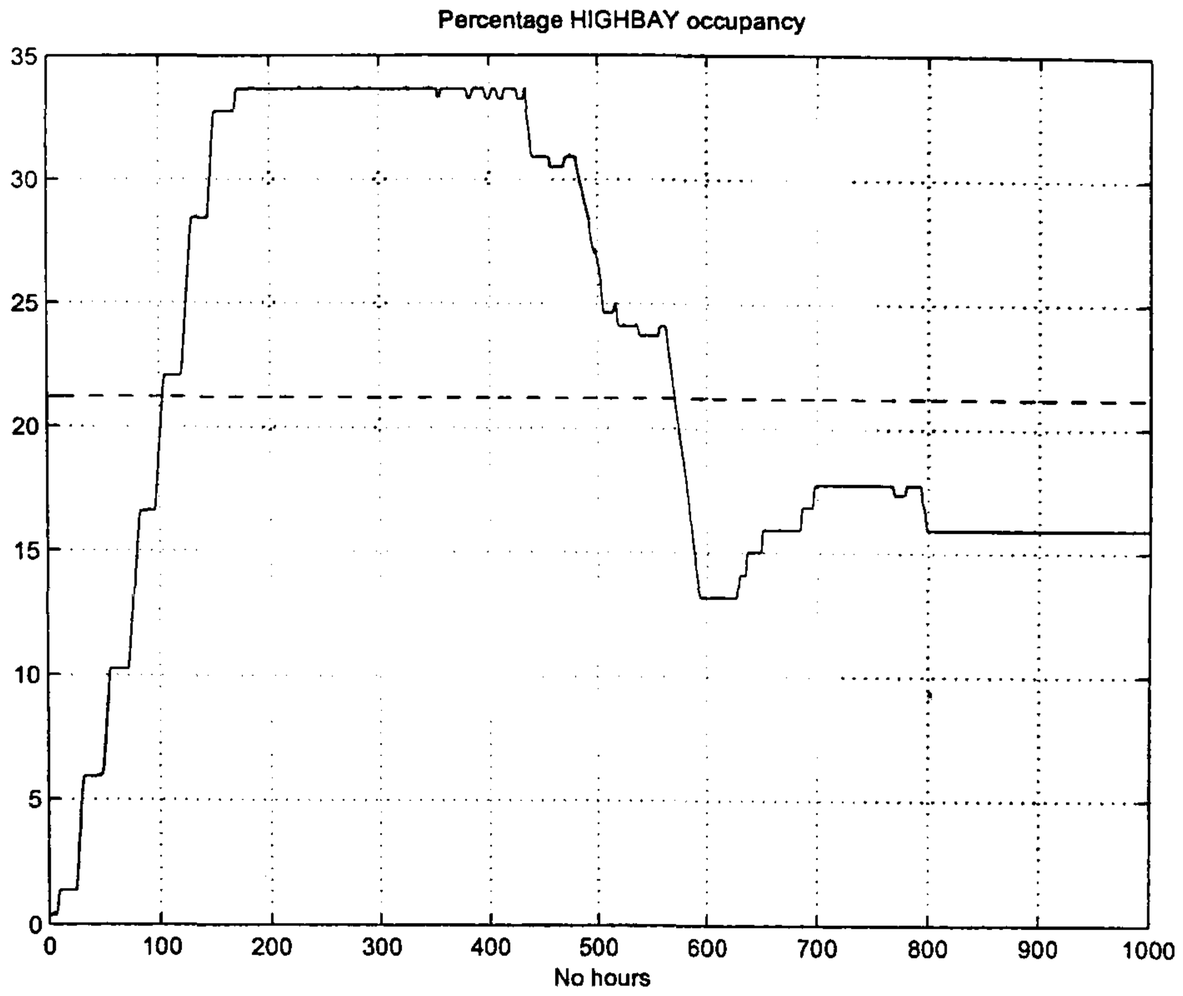

Figure 6.11: High-Bay storage area occupancy for Scenario 1

Table 6.9: Total crane movements in each location for Scenario 2

	Coil Entry	Annealing machines	Cold-rolling input	Cold-rolling output	BWG machine
Crane movements	168	328	320	320	109

while coils enter the storage area from the Hot-Line. Then between the 8th and 14th day the High-Bay occupancy remains constant (34% of maximum capacity) while between the 14th and 21st day the High-Bay storage area occupancy starts to decrease as a result of the number of coils returning to the High-Bay storage area after annealing or cold-rolling exceeds the number of coils entering the BWG machine centre.

Table 6.10 summarises the results for the second scenario associated with the average usage of the following locations: Annealing machine centres, Cold-rolling activity, High-Bay storage area capacity and BWG work centre.

Table 6.10: Percentage of each location's usage average for Scenario 2

	Annealing	Cold-rolling	BWG	High-Bay storage area
Usage	72.56	9.025	10.09	22.28

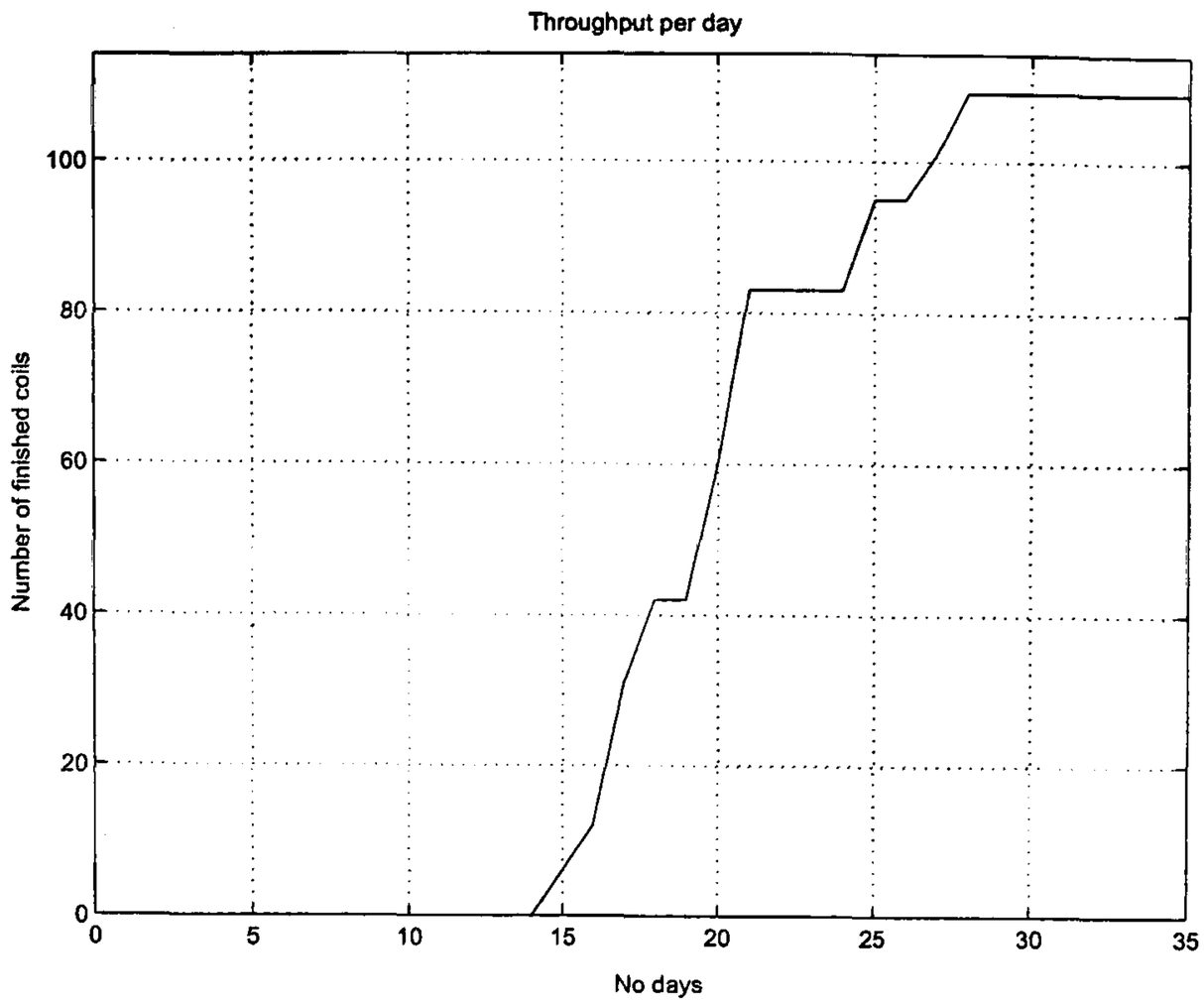

Figure 6.12: Throughput rate for Scenario 2

By comparing Tables 6.8 and 6.10 it can be inferred that by installing a fourth annealing machine the usage of cold-rolling machine increases by 21.77% and the corresponding BWG usage increases by 23.88%. In contrast, the High-Bay storage area occupancy decreases by 4.99% since now less number of coils remain in the High-Bay storage area on average during the simulation run of the production system. The annealing furnaces' usage also decrease by 9.51% also as expected, since the total number of annealing cycles for the second scenario increase due to the addition of fourth annealing furnace.

6.4.3 Scenario 3: Reducing pre-set times the coils remain in High-Bay during cooling and after annealing

In this scenario the number of annealing machines is reset to three, but now the cooling time after all annealing procedures are reduced by 50%. Thus the cooling time in the High-Bay storage area is set to 36 hours instead of 72 in Scenario 1. The report for the simulation period of 33 days is also included in Appendix C. Simulation results show that the throughput rate is now 102 coils over 33 days. It is clear that

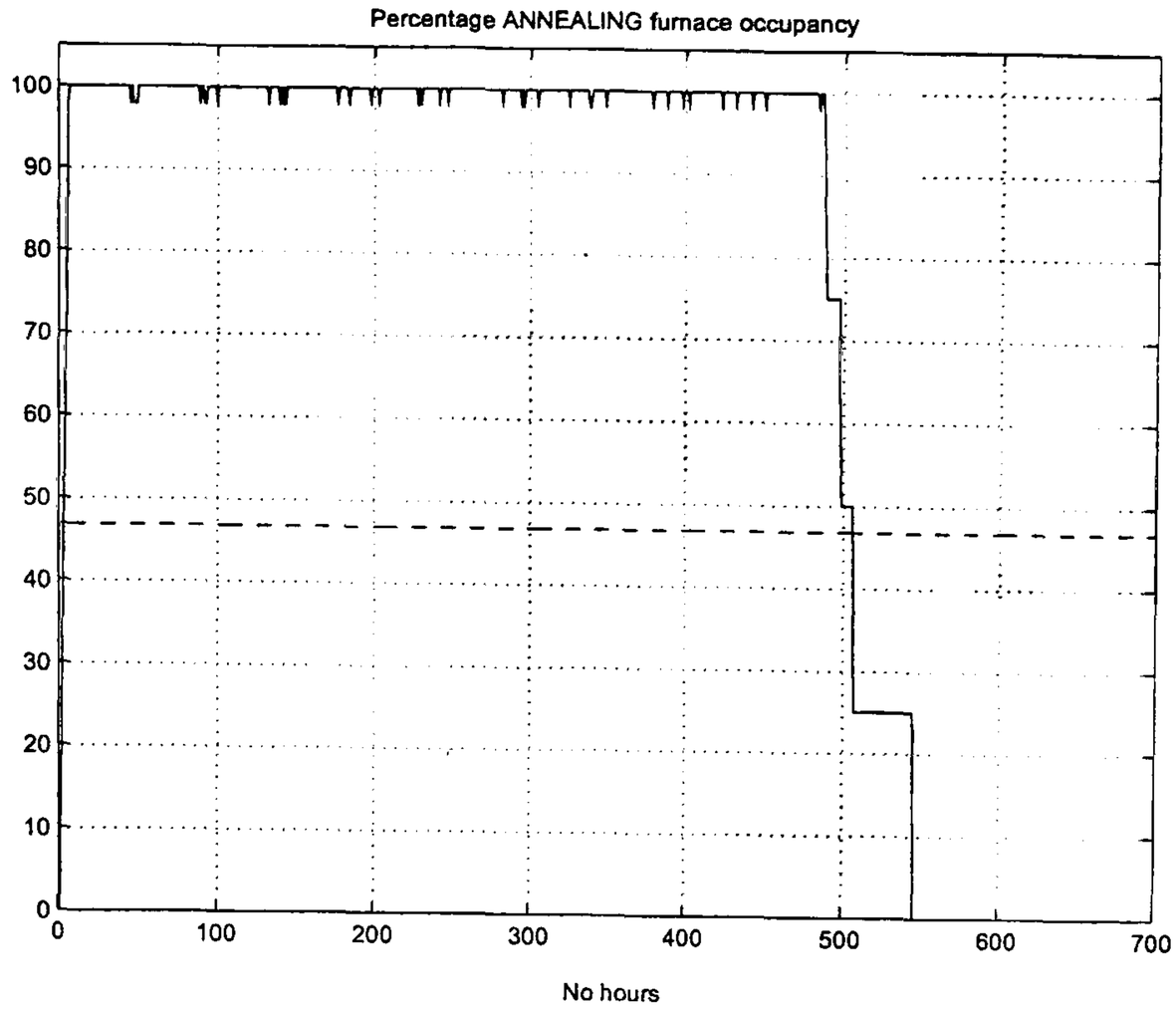

Figure 6.13: Annealing furnaces occupancy for Scenario 2

the throughput rate has increased compared to the corresponding throughput of 90 coils in the first scenario. This significant increment of 11.76% can be also viewed in Figure 6.18, which shows the changes in throughput rate for the third scenario. Note again that although 168 coils have entered the Litho centre in seven days, only 102 of them leave the Litho centre while 66 coils (168-109 this time) remained in the High-Bay storage area at the end of the simulation.

The first coil leave the Litho centre in the 17th day, instead of the 19th as in the first scenario, while the throughput rate increases constantly between the 18th and 24th day. This indicates that the third scenario provides faster coils' exits.

Figure 6.19 depicts the percentage of the three annealing furnaces occupancy over time. These values were in excess of 99% for the first 27 days, highlighting again the fact that at least one annealing furnace each day is fully occupied. Figure 6.19 shows also that the last annealing cycle occurs at the 30th day. More specifically, the usage of the annealing machine this day is 2% and only four coils have been annealed. The cold-rolling machine centre usage for the third scenario can be viewed in Figure 6.20, where the main occupancy time takes place between the 15th and 23rd day. The last

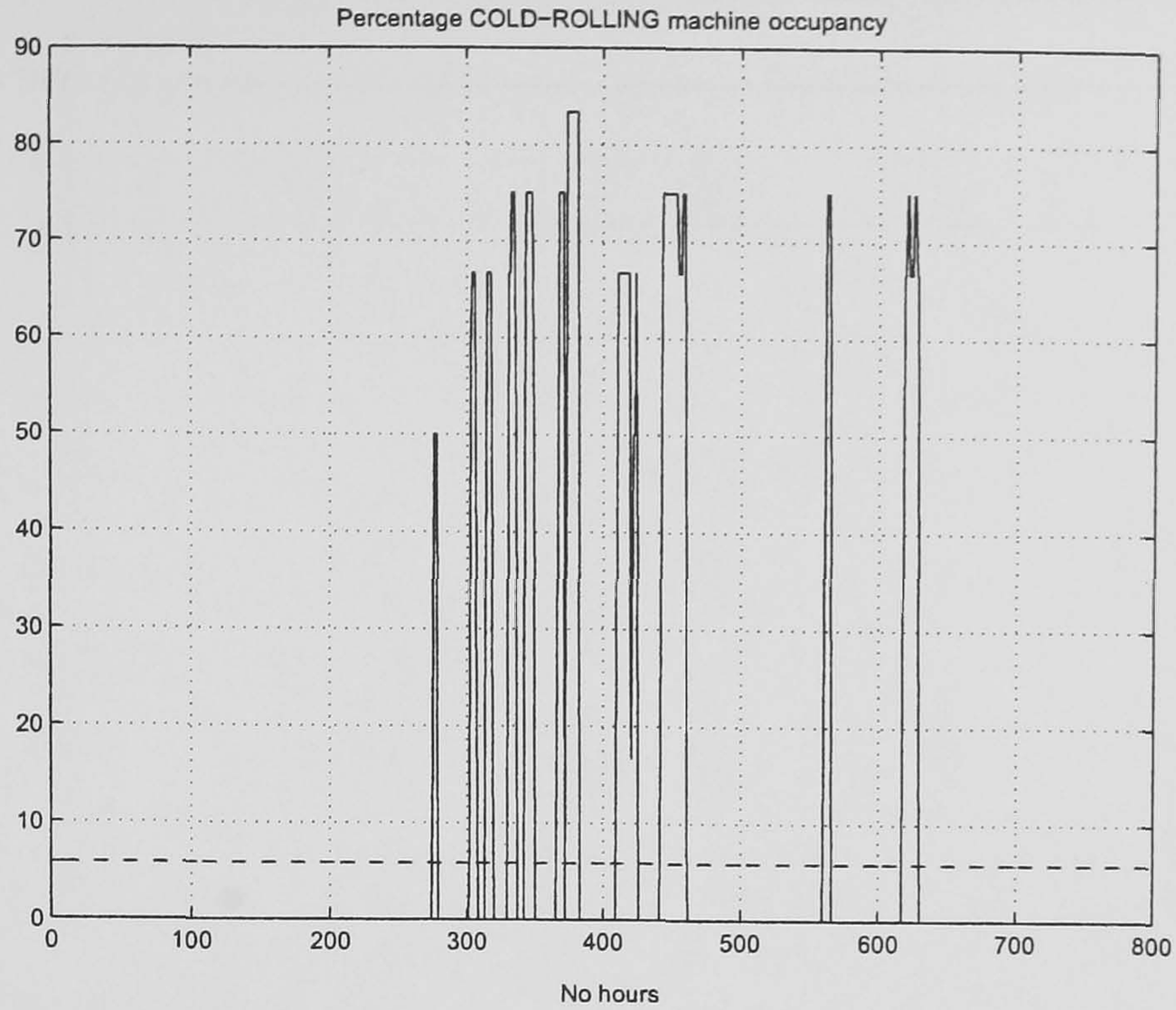

Figure 6.14: Cold-rolling machine usage for Scenario 2

day of cold-rolling activity is the 31st day which is again reasonable since the last annealing cycle occurred during the 30th day. Note that cooling time after annealing for this scenario is set to 36 hours (1.5 day).

Figure 6.21 illustrates that in a similar way with the other two scenarios, coils enter the BWG machine centre in batches. This time the BWG is busy for only eight days during the whole simulation period for which all coils passed the BWG stage, while the last BWG activity takes place on the 32th day (again one day after the last cold-rolling process has been completed).

Figure 6.22 shows the total number of crane movements at all four locations. Simulation runs indicate that the maximum number of crane movements per hour are 16, in contrast with the first scenario where the maximum number of movements was 14. Table 6.11 provides useful information for crane movements at each location throughout the simulation run. Note again that the crane spends the most time to serve the annealing process while the BWG is the location where the crane spends the least time.

Occupancy changes per hour in the High-Bay storage area are depicted in

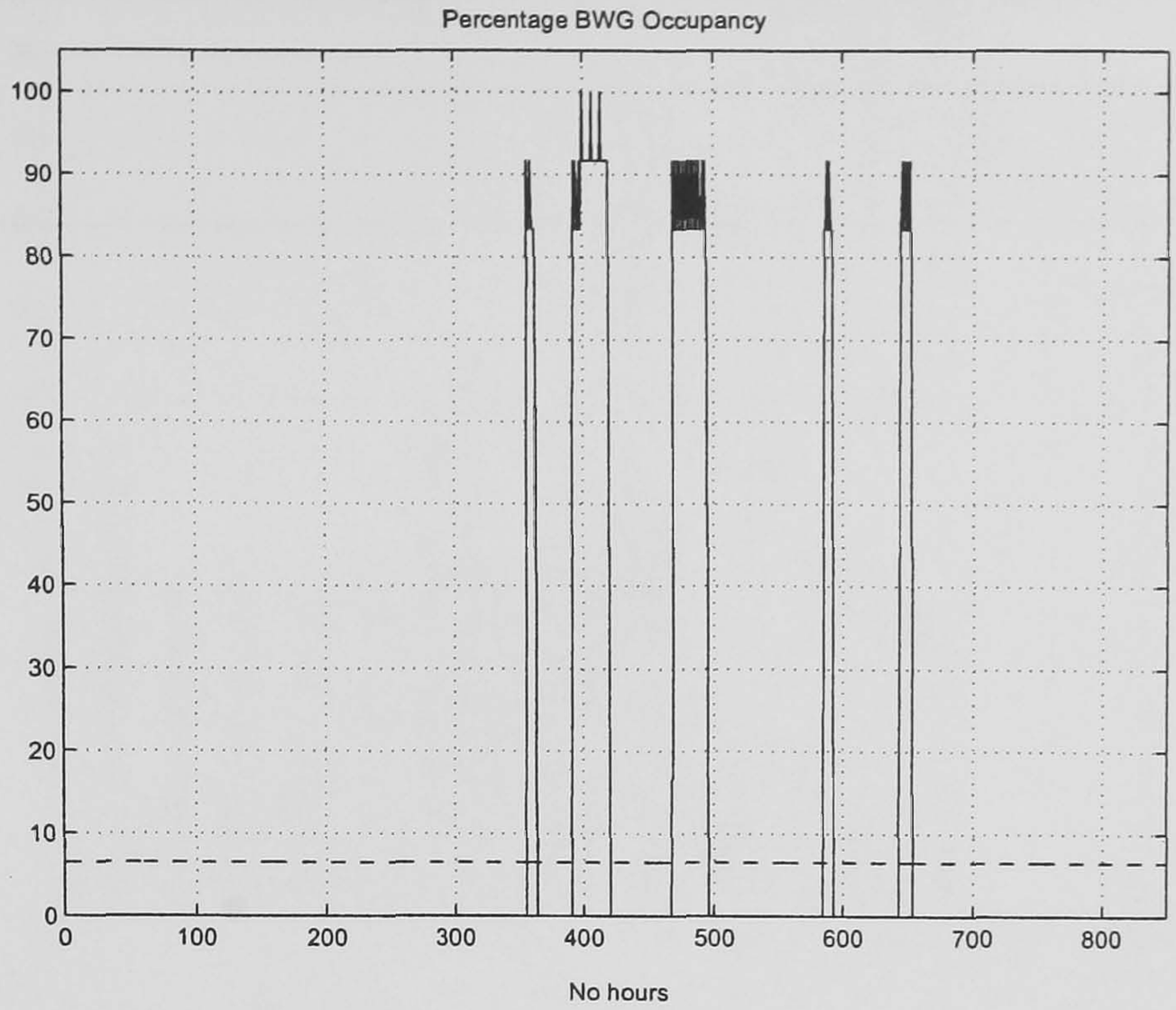

Figure 6.15: BWG occupancy for Scenario 2

Table 6.11: Total crane movements in each location for Scenario 3

	Coil Entry	Annealing machines	Cold-rolling input	Cold-rolling output	BWG machine
Crane movements	168	328	320	320	102

Figure 6.23. Similarly to the first scenario during the first eight days occupancy increases while coils enter the storage area from Hot-Line. This represents the "transient" response of the system, as initially the High-Bay was assumed to be empty. Then, between the 8th and 15th day the High-Bay capacity remains constant (34% of maximum capacity) while between the 16th day and 26th day the High-Bay storage area capacity starts to decrease as some coils which have been cooled after being cold-rolled leave the High-Bay area for BWG machine centre. Note that in the interim period between the 26th and 29th day High-Bay occupancy exhibits again some small fluctuations depending on the differences between the rates at which coils enter or re-enter the High-Bay (after annealing or cold-rolling) and the rate at which coils enter the BWG machine centre.

Table 6.12 summarises the corresponding data for the third scenario, related to the average usage of the annealing machine centres, cold-rolling activity, High-Bay

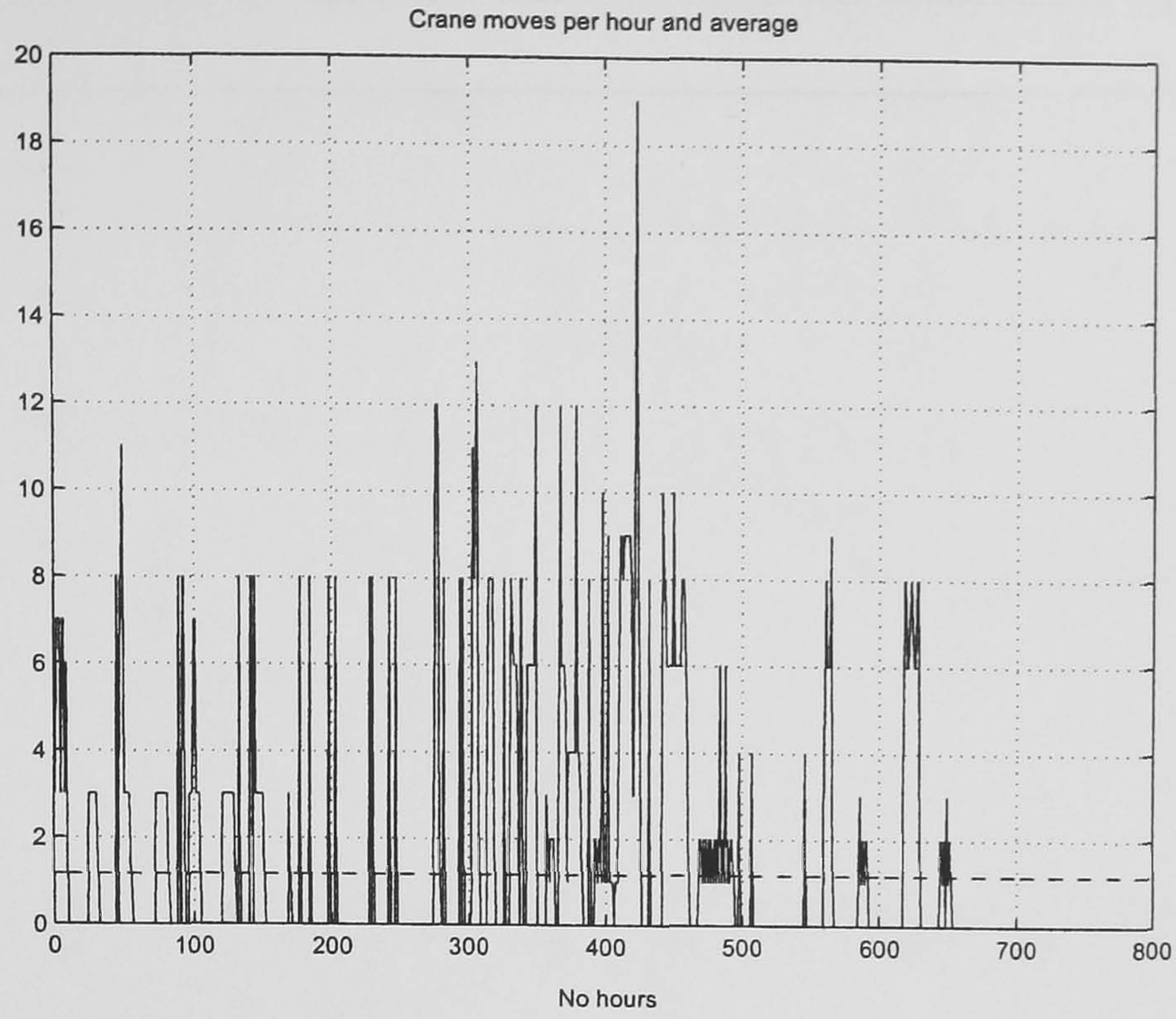

Figure 6.16: Crane movements in total for Scenario 2

storage area capacity and BWG work centre.

Table 6.12: Percentage of each location's usage average for Scenario 3.

	Annealing	Cold-rolling	BWG	High-Bay storage area
Usage	85.04	7.86	8.40	22.98

Comparing the results of Table 6.8 and Table 6.12, it can be concluded that by reducing by 50% the cooling time after the annealing process, the usage of the annealing furnaces has been increased by 5.7%, the corresponding cold-rolling machine usage by 10.23% and the BWG usage by 8.57%. In contrast to the other three locations, the High-Bay storage area occupancy fell on average by 2%, which is expected since coils spend less time inside the High-Bay for cooling purposes.

6.4.4 Scenario 4: The effect of order fluctuations based on current production plant

Previous scenarios studied the impact of changing two of the main plant's parameters that characterise the production system. This scenario has a different purpose since

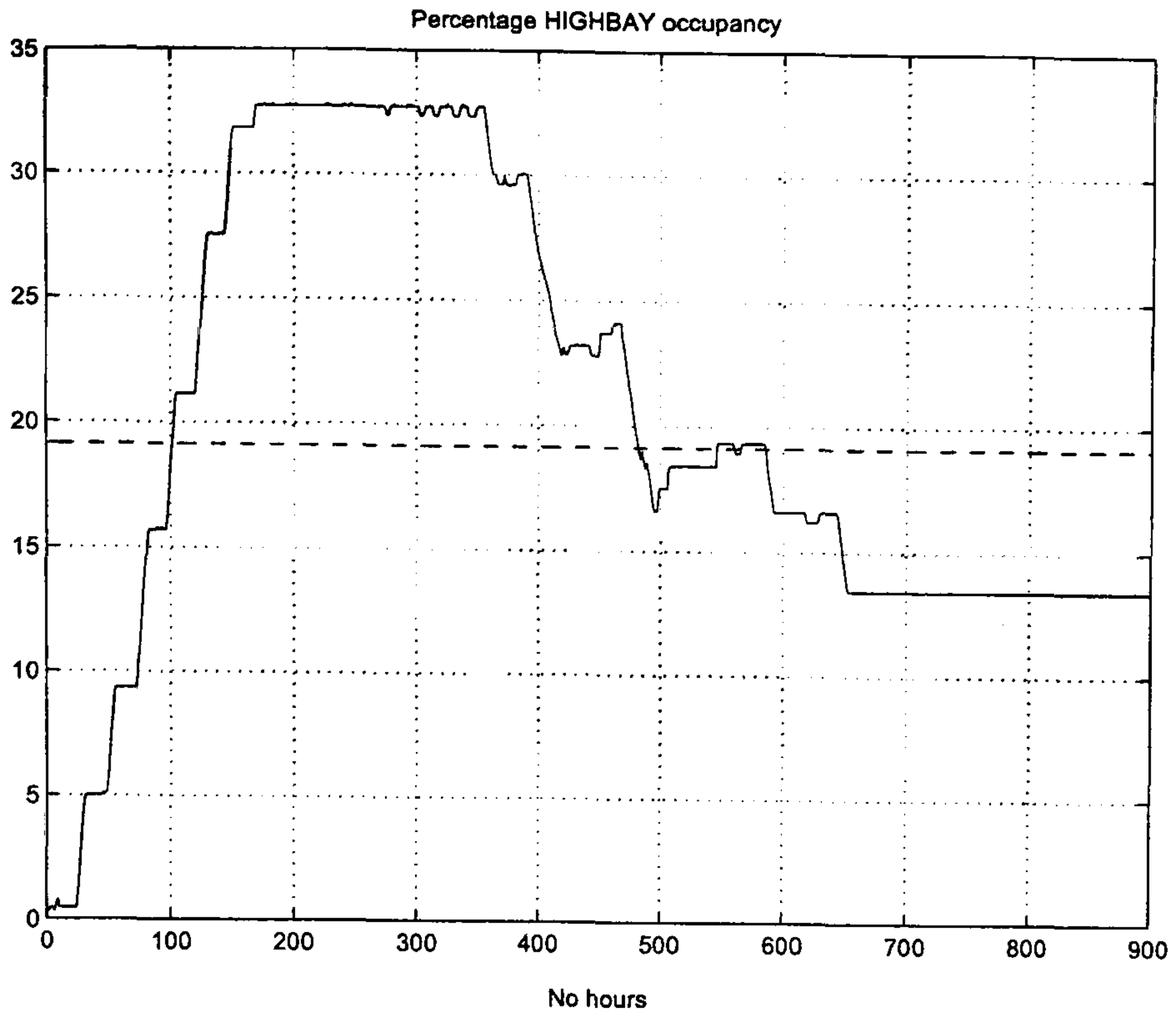

Figure 6.17: High-Bay storage area occupancy for Scenario 2

it examines the effect of demand fluctuations on the efficiency of the production process. The type of demand fluctuations considered are presented in Table 6.13, where it is assumed that coils enter the Litho centre from Hot-Line every two days (i.e., 1st, 3rd, 5th and 7th day). This scenario uses exactly the same average input rate as in the previous three scenarios.

Table 6.13: The Excel input file of 168 coils used in Scenario 4

	Type	No	Type	No	Type	No	Type	No	Type	No	Type	No	Type	No	Type	No
Day1	6	8	9	6	6	9	1	4								
Day2																
Day3	6	8	2	4	5	4	3	4	9	3	6	4	11	4	4	8
Day4																
Day5	9	12	1	4	13	8	2	4	6	12	3	4	7	4	10	4
Day6																
Day7	5	8	6	12	8	4	17	4	9	3	12	4	15	8	14	4

The report for a simulation period of 35 days is included in Appendix C. Simulation results show that the throughput rate is now 98 coils in 35 days, which is equal to the corresponding throughput rate derived in the first scenario. Figure 6.24 depicts the changes in throughput for the complete simulation, which appears to have similar pattern with the Figure 6.7.

There are also many similarities with the annealing furnaces and cold-rolling machines centres' usages plots - depicted in Figure 6.25 and Figure 6.26, respectively

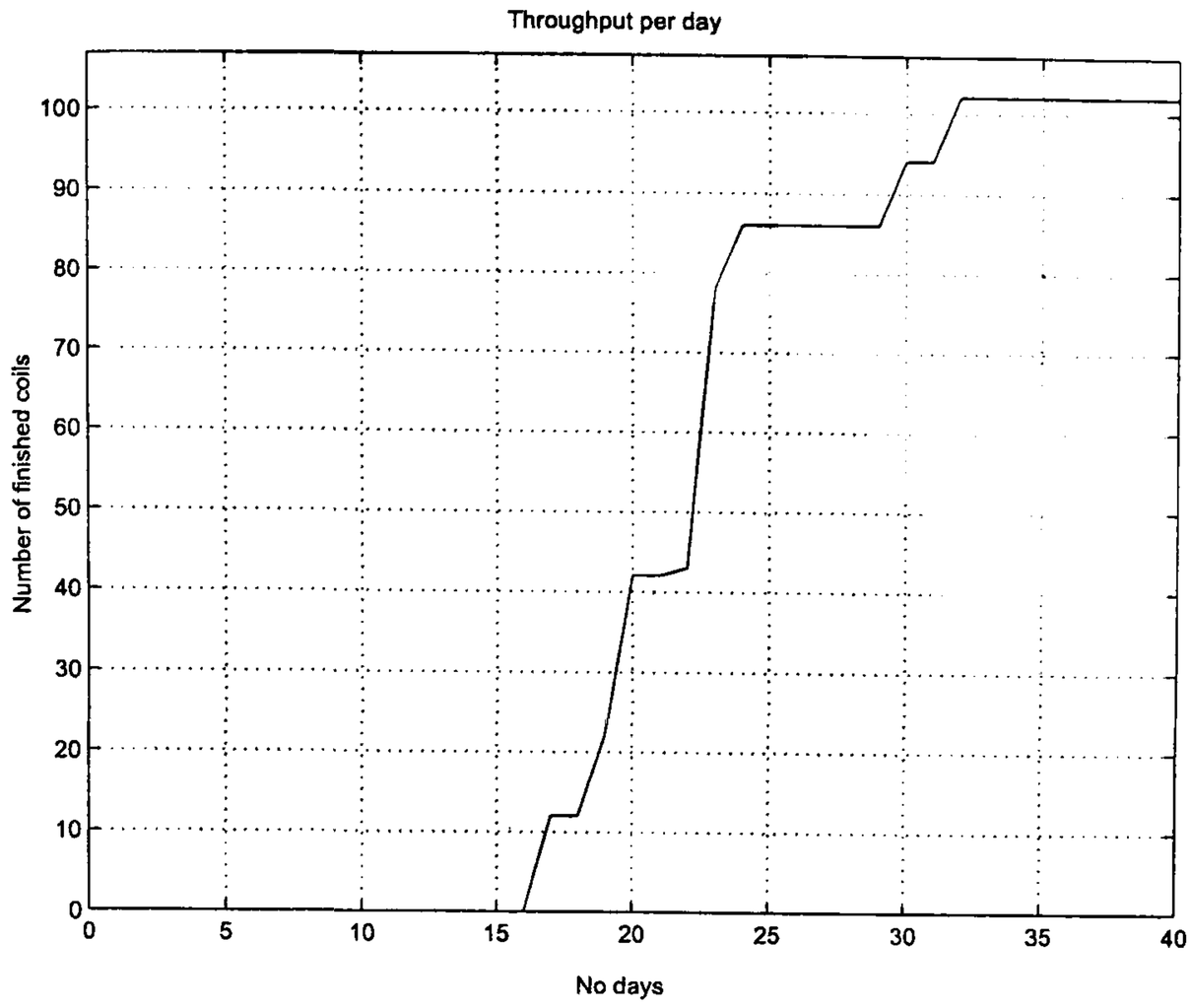

Figure 6.18: Throughput rate for Scenario 3

- with the corresponding Figures presented in Scenario 1.

Demand fluctuations have only significant impact to the first cold-rolling processes (and hence to the first BWG machine centre activities depicted in Figure 6.27).

Actual crane movements per hour for the fourth scenario shown in Figure 6.28 exhibit more fluctuations in comparison to the crane movements corresponding in Scenario 1, especially for the first ten days. This crane performance can be explained by the fact that in two consecutive days (e.g., 3rd and 4th day), it performs different tasks. Thus, during the first day, in addition from movements related to annealing and cold-rolling the crane also loads into the High-Bay storage area coils arriving from the Hot-Line, while on the second day the crane only serves the annealing and cold-rolling machine centres.

Finally, the High-Bay area in the fourth scenario has been found to contain all the 168 coils with a delay of one day (9th day). Figure 6.29 depicts the changes in the High-Bay storage area, where demand fluctuations result in more rapid occupancy increases from day to day. A similar observation has also been made in chapter 3,

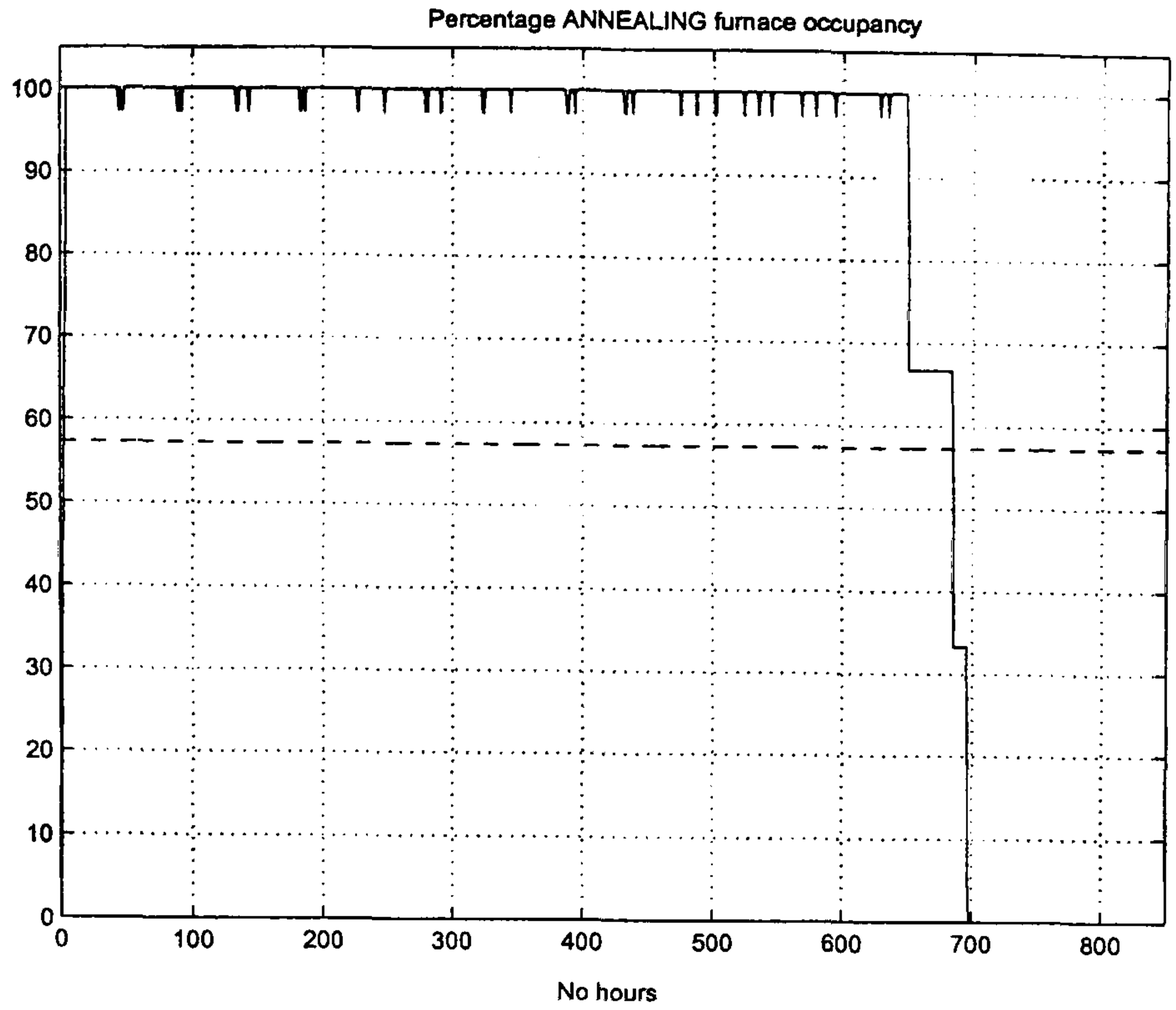

Figure 6.19: Annealing furnaces occupancy for Scenario 3

where it has been shown that demand variability in series supply chains leads to demand amplification in downstream nodes (bullwhip effect) and thus downstream participants experience rapid fluctuations in their inventories.

Simulation results also suggest that the average usages at all four locations have similar values to those derived in Scenario 1 (see Table 6.8.

6.4.5 Performance and simulation results analysis

The overall conclusion that can be drawn from the simulation results for both scenario 2 and scenario 3 together with comparisons with the main findings of Scenario 1, is that significant improvements in the production system performance can be obtained. More specifically, it was found that installing an additional annealing machine or introduction of new annealing technology results in a considerable increase in the overall throughput rate and also in increased machine utilisation in the cold-rolling and BWG processes. Moreover, crane movements per unit time are greater than those derived in Scenario 1, which suggests an increase in Work-In-Process inventory. Thus, simulations of this type may be used (together with economic considerations

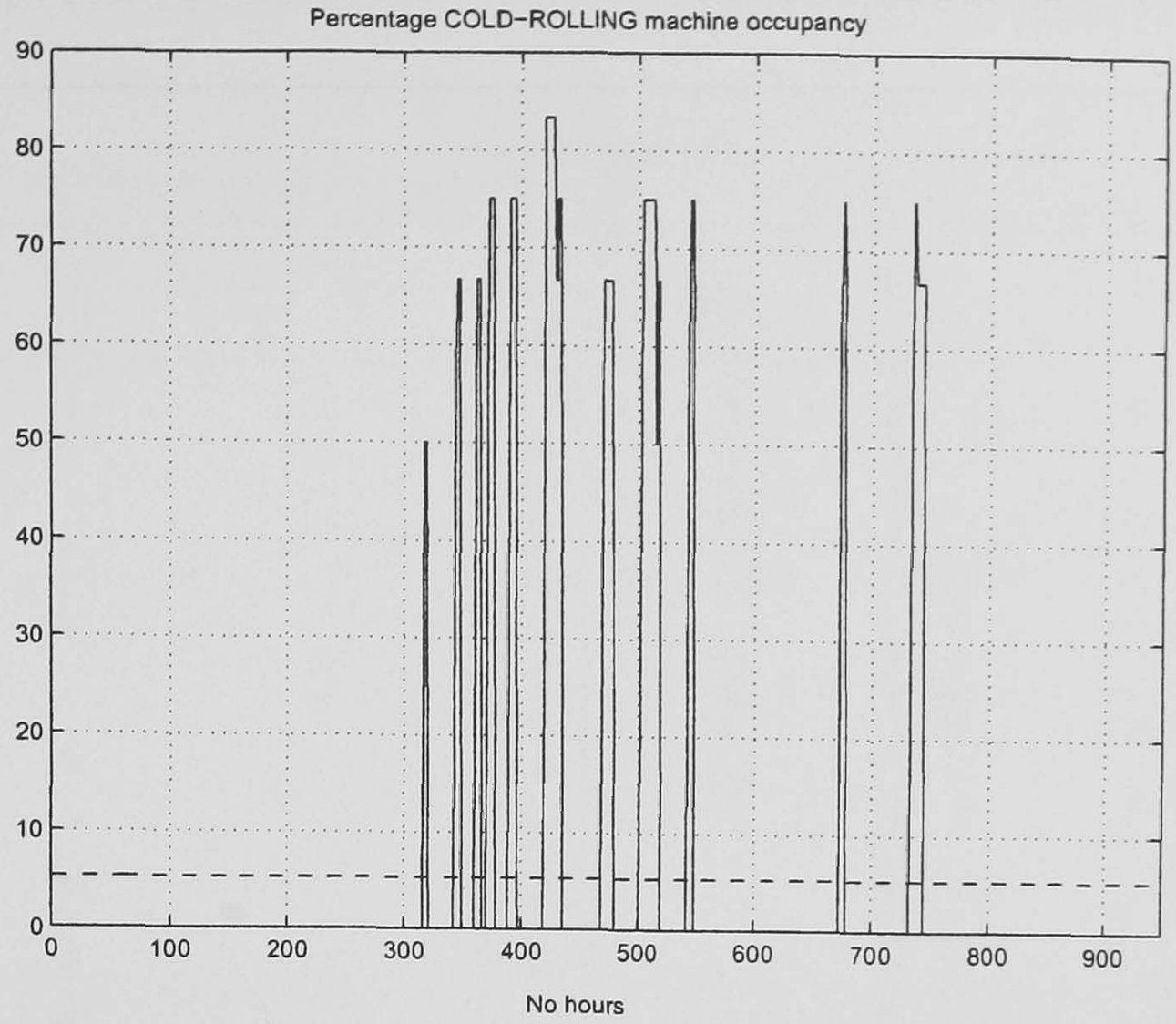

Figure 6.20: Cold-rolling machine usage for Scenario 3

such as expected future demand levels, costs, depreciation factors, etc.) to make rational decisions about future investments.

Increased throughput rates were also achieved in the case when the pre-set times the coils remain in High-Bay storage area during cooling after annealing, are reduced. In this scenario the average usage of all locations has also increased, while the crane moves on average are also increased compared with Scenario 1.

Finally, demand fluctuations in scenario 4 exhibit increased variability levels in High-Bay storage area occupancy. An additional effect of demand fluctuations is that the number of crane routes made in two consecutive hours may differ considerably. However, in order to examine better the impact of demand fluctuations in production systems, the metrics must be analysed and assessed after the system has reached its steady state. However, this study is beyond the scope of this research work and requires extensive simulation studies involving a huge amount of real data.

It is clear that the assessment of all alternative scenarios provided by extensive simulations based on the proposed model can aid significantly decision-making management procedures, especially related to cost over benefit investment decisions.

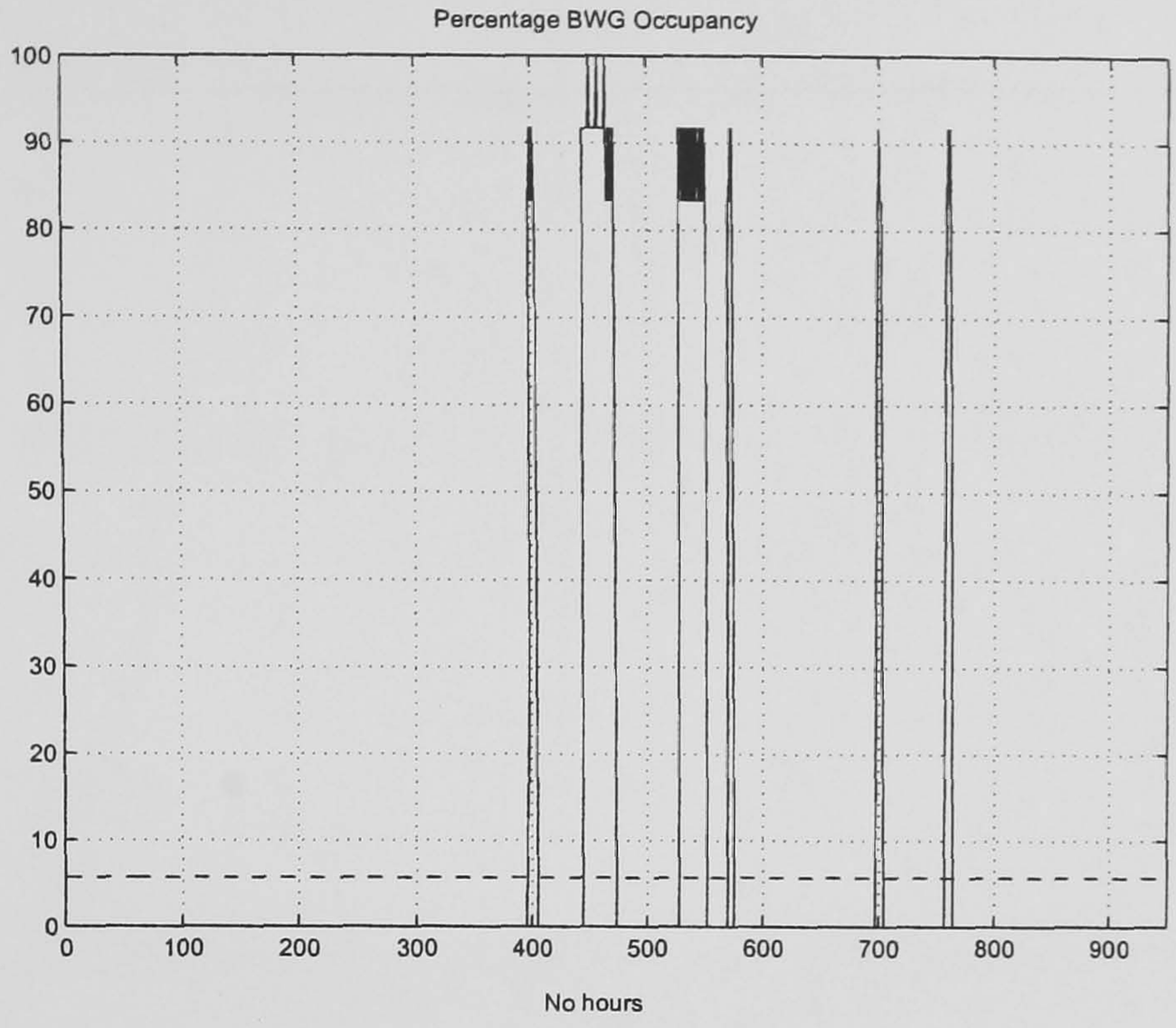

Figure 6.21: BWG occupancy for Scenario 3

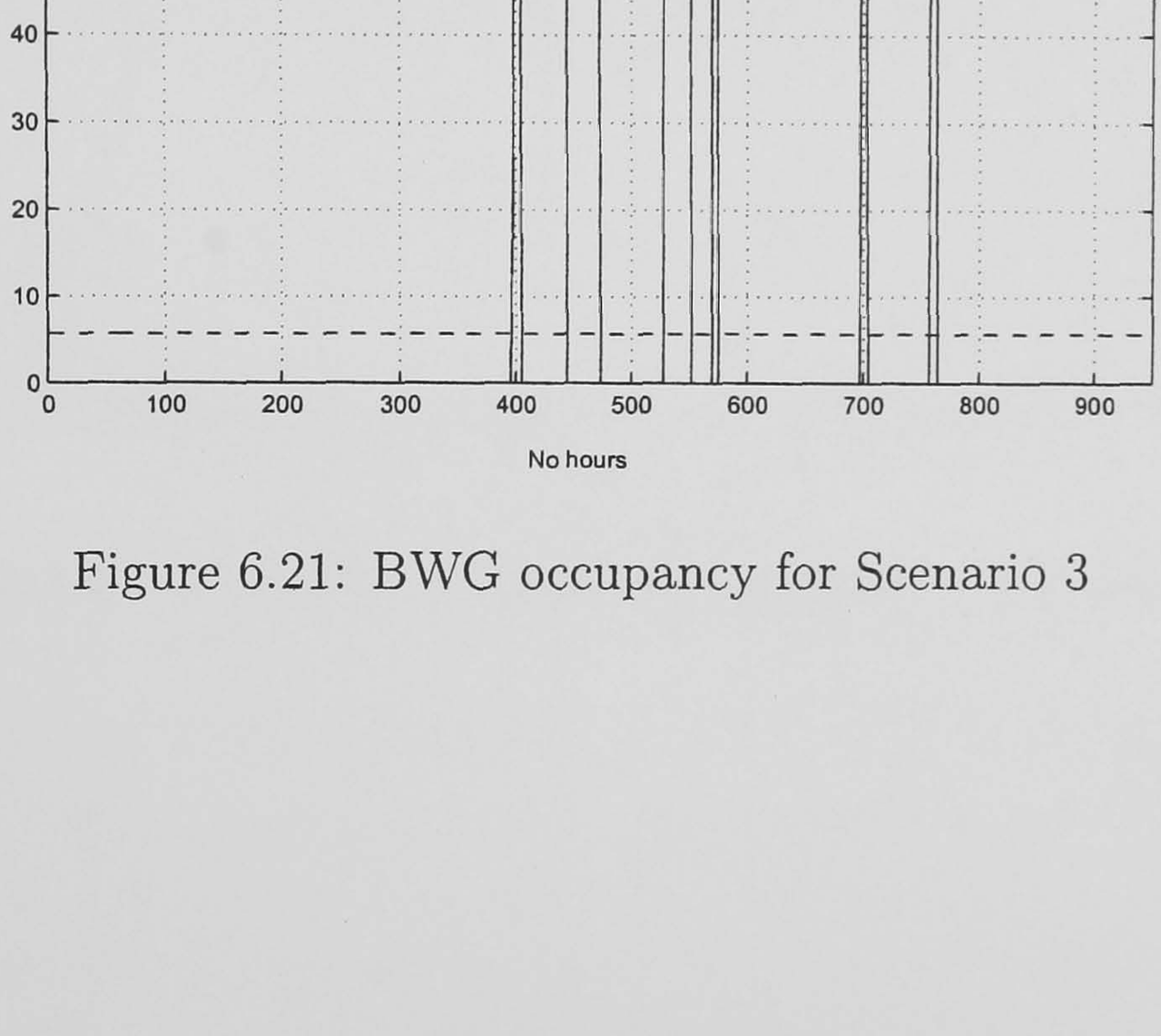

Figure 6.22: Crane movements in total for Scenario 3

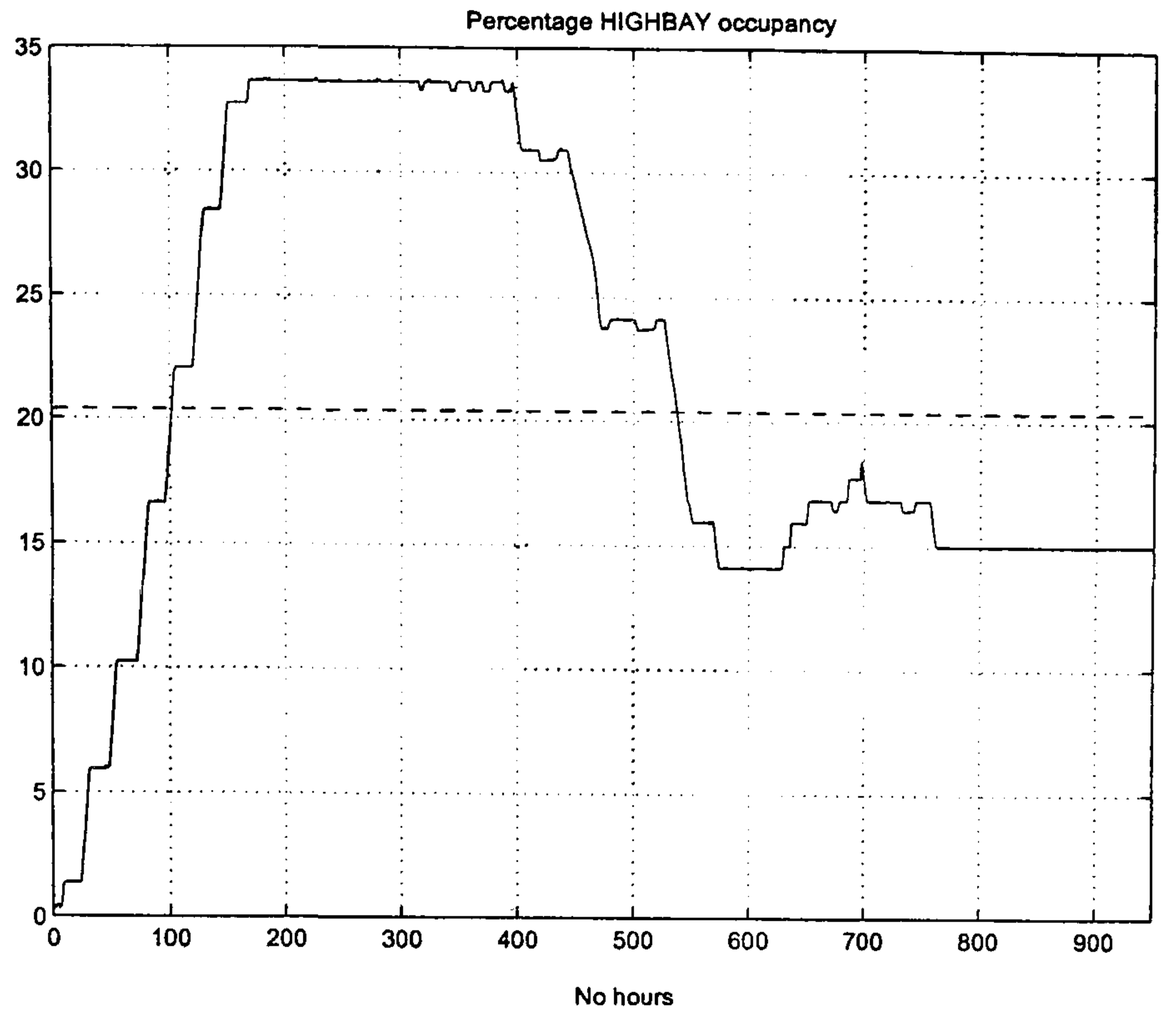

Figure 6.23: High-Bay storage area occupancy for Scenario 3

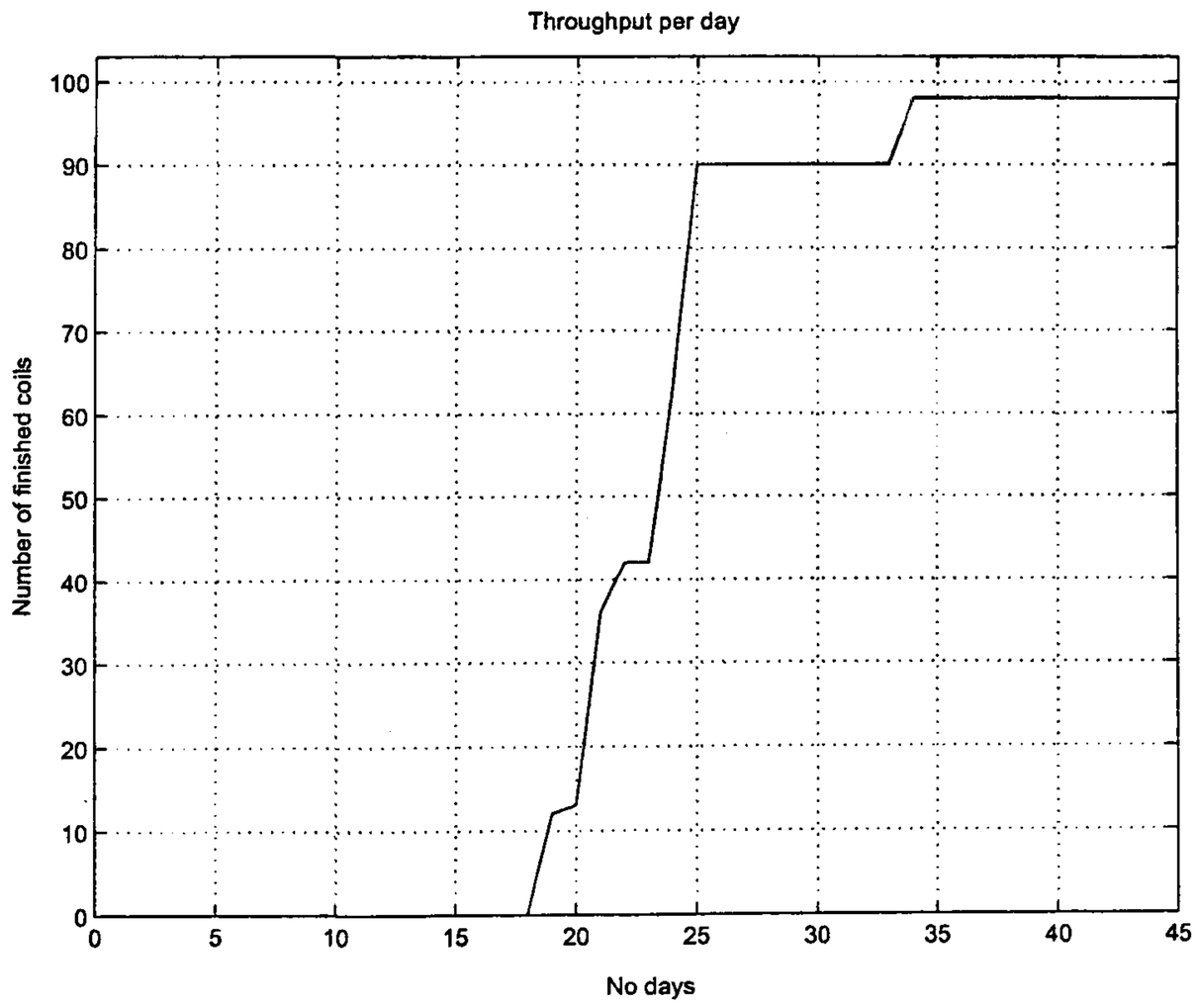

Figure 6.24: Throughput rate for Scenario 4

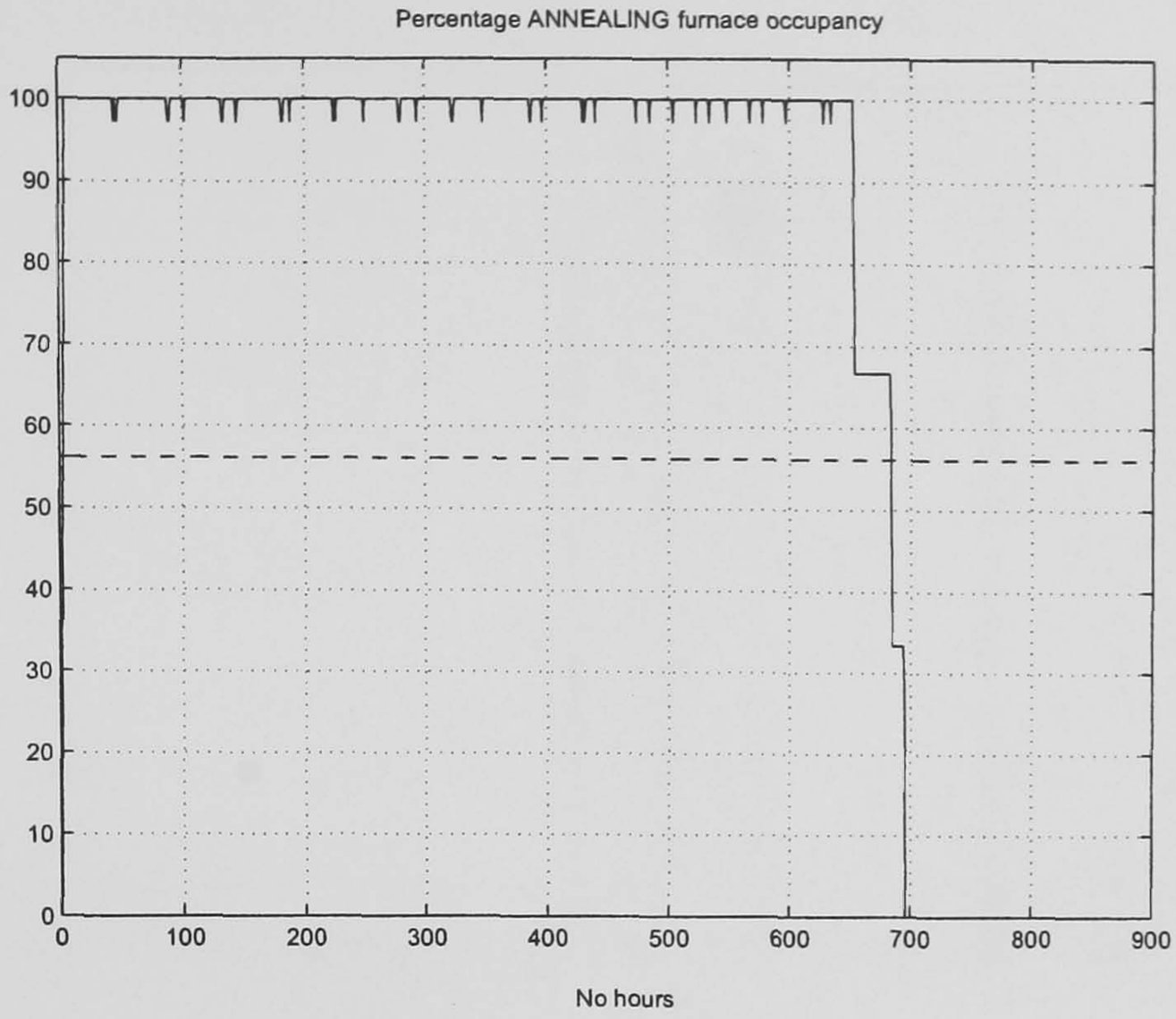

Figure 6.25: Annealing furnaces occupancy for Scenario 4

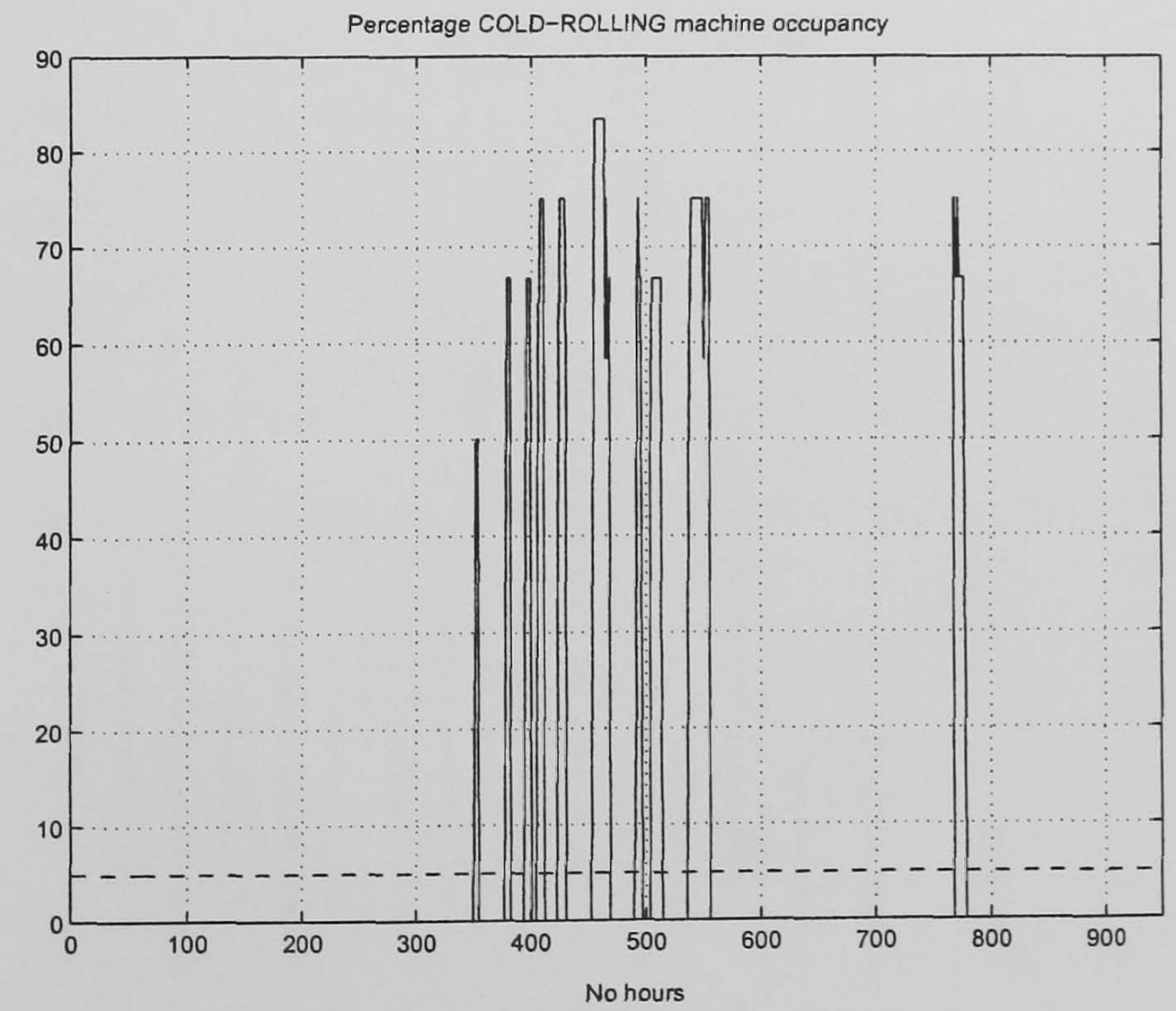

Figure 6.26: Cold-rolling machine usage for Scenario 4

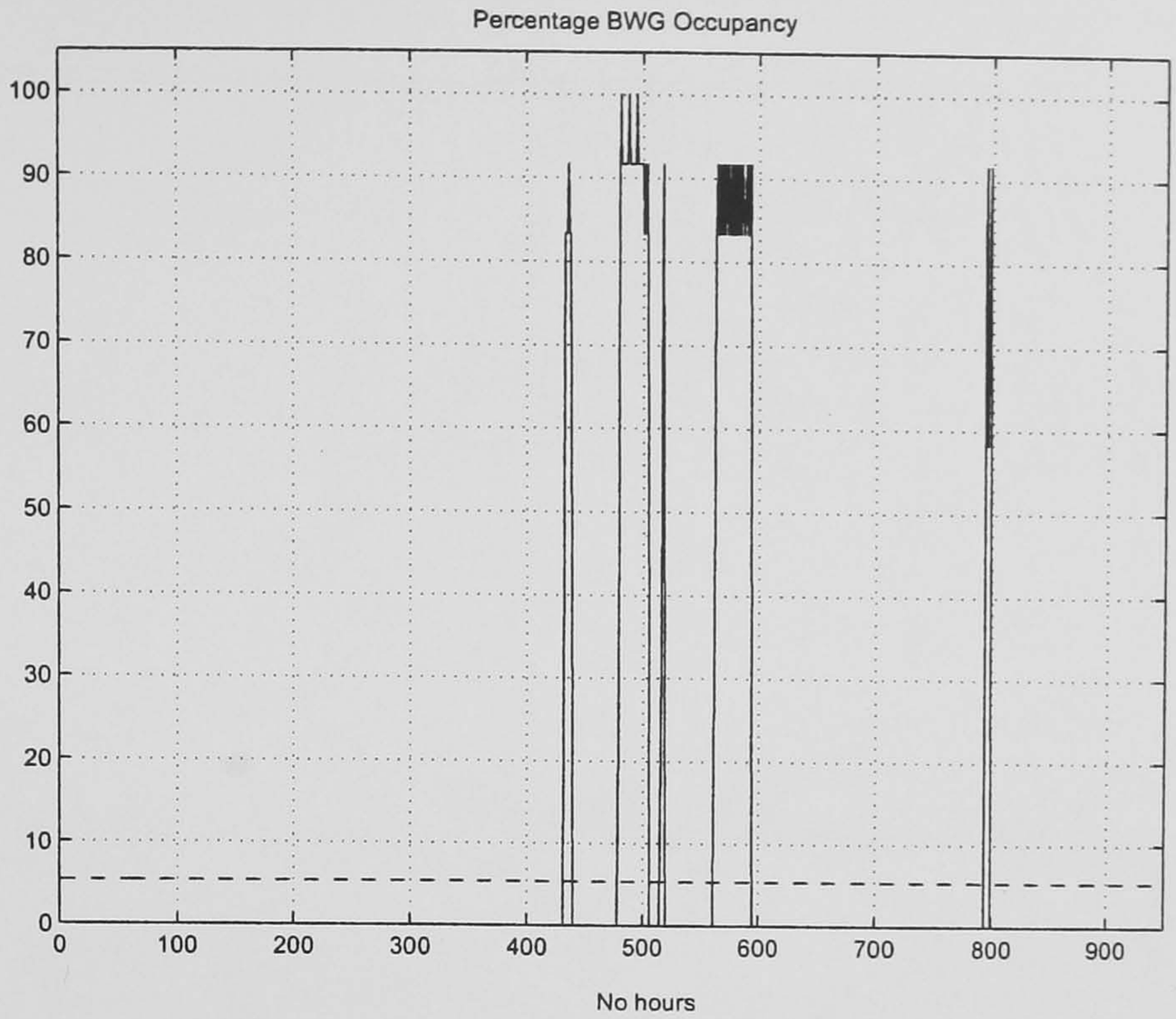

Figure 6.27: BWG occupancy for Scenario 4

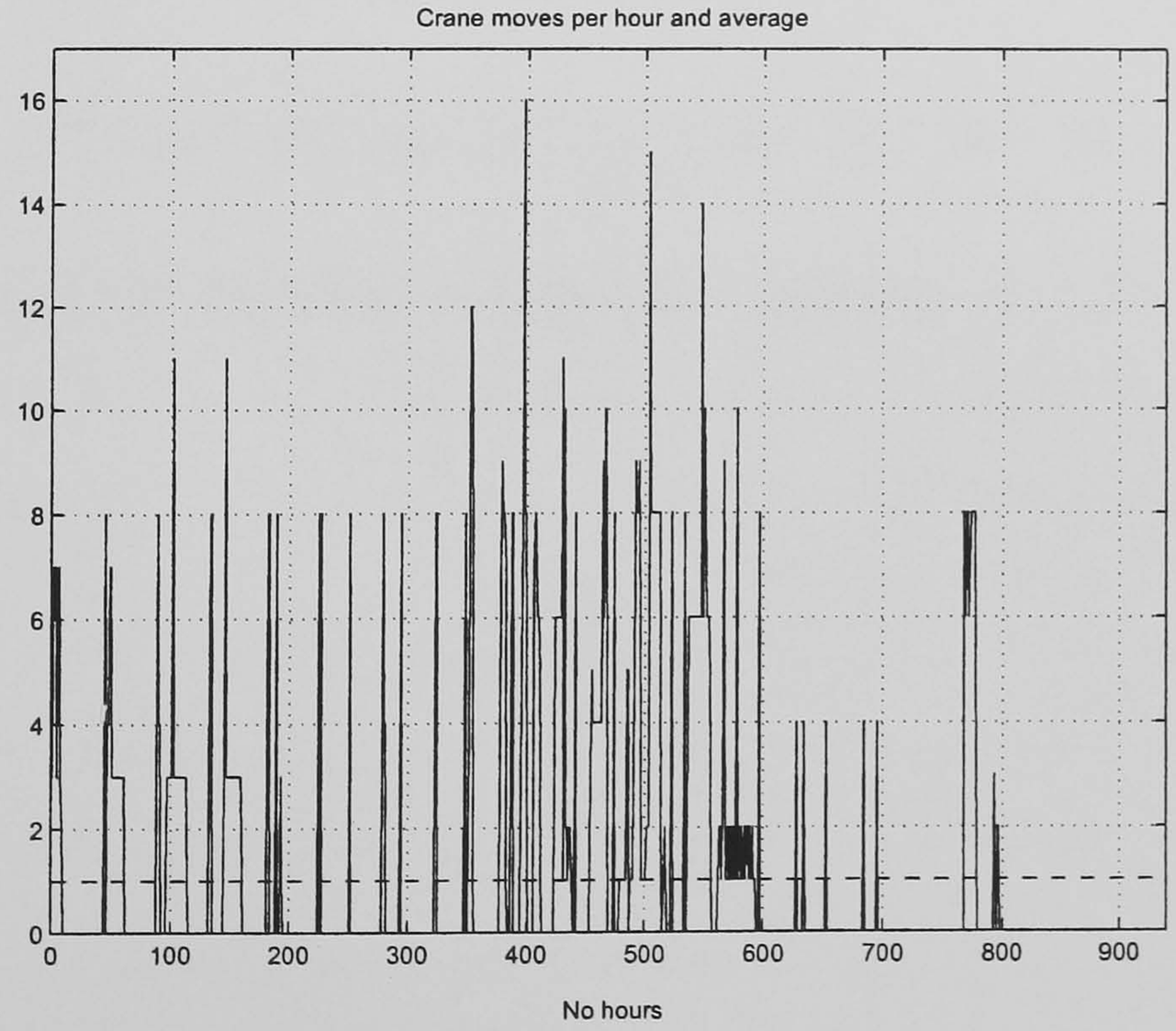

Figure 6.28: Crane movements in total for Scenario 4

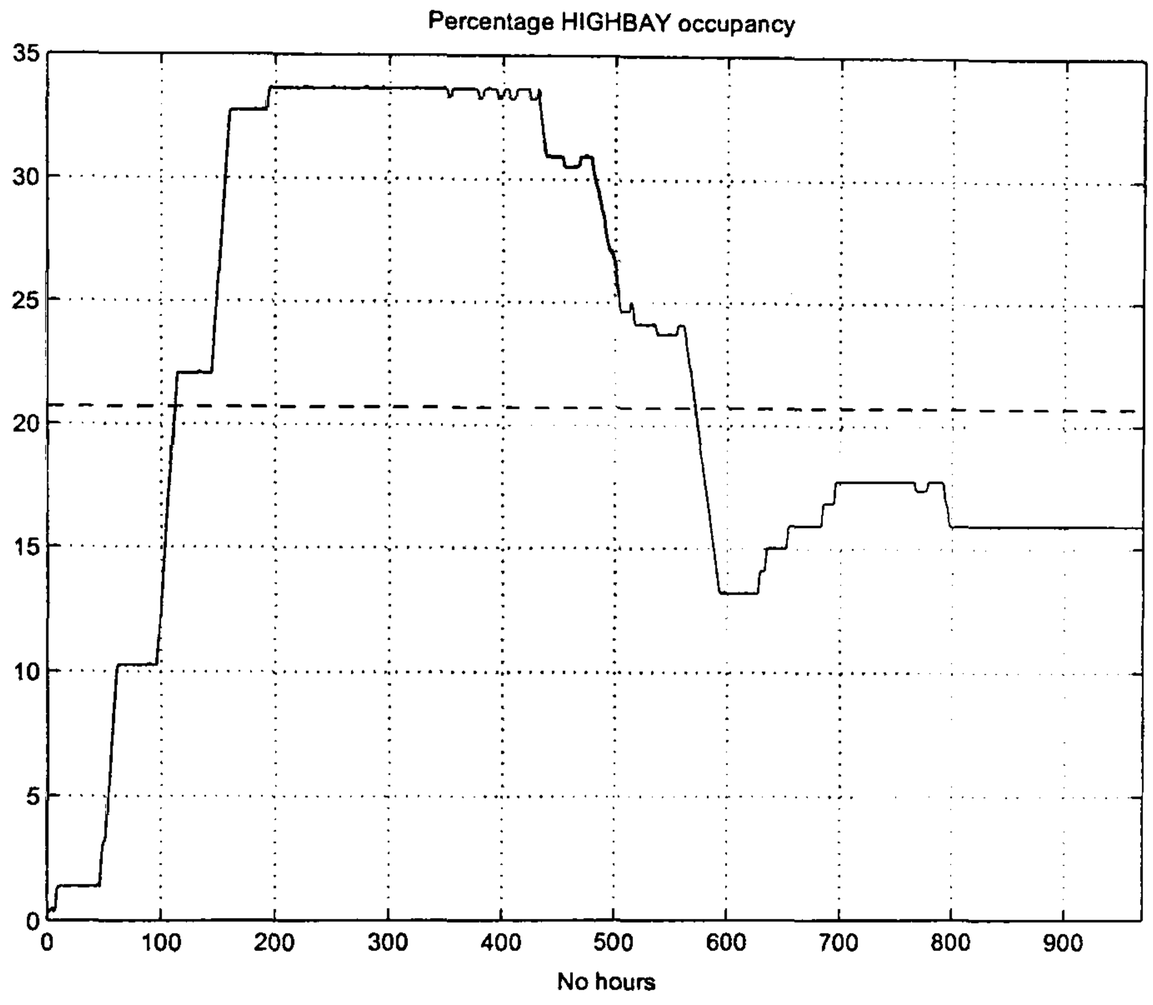

Figure 6.29: High-Bay storage area occupancy for Scenario 4

Chapter 7

Conclusions and further work

In this section we summarise the main conclusions of the work.

A novel state-space model has been presented for analysing the effect of proportional policies in series multi-node supply chains. An augmented state space model of the complete series supply chain has been obtained using as state variables the inventory position and the amount of goods delivered by each node. The state space model depends parametrically on the inventory set-points and replenishment gain factors. It has been shown that the structure of the model also allows for a simple recursive updating scheme for calculating the covariance matrix in parametric form, for models with an arbitrary number of nodes.

A detailed model stability analysis has defined explicit ranges of the parameters for which the model is stationary. In addition, the modelling approach has resulted in a full characterisation and prediction of the bullwhip effect via the covariance analysis, leading to an explicit algebraic equation depending on the proportional inventory replenishment gain factors. This has allowed for the partitioning of the stability gain space in two areas corresponding to the attenuation and amplification regions. The analysis was extended to the case when the customer demand profile is correlated via filtering a white noise sequence through a first order autoregressive (AR) model. In this case it was shown that the attenuation region expands at the expense of the amplification region to a degree depending on the value of the model's correlation parameter.

Under the assumption of full co-operation between adjacent nodes an optimisa-

tion problem was formulated involving the hierarchical minimisation of fluctuations in inventory and of the mean inventory level, subject to a probabilistic constraint on the ability of the node to meet downstream demand which is associated with the chain's service levels. It has been shown that such a "selfish" policy does not lead to demand amplification.

Finally, three separate local estimation schemes have been proposed and analysed in the absence of information sharing. This potentially allows the application of the optimal policies obtained under full information to be extended to the more realistic case where nodes do not exchange information.

The analysis based on the dynamic model of the series supply chain developed in chapter 3 and chapter 4, is limited by a linearity assumption which states that each node has sufficient inventory at all times to meet downstream demand. Hierarchical Coloured Petri Nets are implemented in chapter 5 to examine different inventory policies which cannot be applied within the dynamic model framework developed in the two previous chapters. The proposed simulation model compares various scenarios including deterministic end-customer demand profiles in batches and stochastic profiles which are normally distributed. Three different ordering policies widely used in supply chain management were analysed. Simulation results suggest that MA continuous policies offer advantages for most of the scenarios analysed. Using this method the bullwhip effect can be reduced, thus avoiding high back-order levels, while customer satisfaction is increased.

A detailed case study was undertaken in chapter 6, for the production line of the "Bridngorth Aluminium Ltd" company which produces high quality rolled aluminium lithographic strips. Discrete Event Systems (DES) background theory was used to model the production process by using the general methodology developed in the earlier part of this work, resulting in a versatile simulation tool coded in MATLAB programming language. The effect of highly volatile demand profiles were investigated in terms of decreased throughput rates, reduced levels of average machine centres utilisation and other factors which imply increased costs for

the manufacturing process.

The model was successfully validated using actual production data and it was found that the simulation tool is suitable for modelling, analysis and performance-evaluation of the complex production process. With the aid of the model, various scenarios were investigated via extensive simulation runs, such as the effects of installing additional machine centres and reducing the pre-set times the products spend in intermediate storage areas. Results show that production managers can strongly benefit from the proposed model in understanding the dynamics of the process and in improved decision-making.

We conclude the chapter by outlining a number of interesting topics related to the results of the thesis which can be addressed in future work.

Supply chains are complex dynamic systems with intricate interrelations and cumbersome functions. Therefore, additional modelling work is required to capture the main characteristics of real supply chains.

The use of additional information provided by the structure of the covariance matrix may be used to estimate unknown parameters of the system (e.g., future demand profiles) hopefully leading to an effective decentralised control scheme. A second step towards the alleviation of the bullwhip effect phenomenon is the implementation of more sophisticated forecasting and control schemes, e.g., based on Model Predictive Control and optimisation techniques. Alternative estimation schemes (e.g., based on likelihood theory) can also be studied and compared with those developed in this work.

Naturally any model is an idealisation of the real system and as such its utility can be extended only up to a certain point. It may be useful to test the limits of validity of the developed model by scrutinising its main assumptions. One limitation of the model is the assumption that all participants follow the same type of (proportional) inventory replenishment policy, which of course cannot be guaranteed in practice. Assuming multiple vendors placing orders to their upstream node by pursuing a mixture of individual policies, it may be interesting to investigate to what extend

their aggregate demand resembles at least approximately an equivalent proportional policy.

Hierarchical Coloured Petri nets and simulation tools can be used to identify improved inventory policies and to analyse the impact of the main bullwhip effect causes discussed in Chapter 3. An important modelling issue is the formulation of realistic inventory cost functions for each stage of the supply chain, which can be subsequently used to estimate the optimal inventory levels for minimum cost, under the constraint of meeting order demand from the downstream stages.

Due to the bullwhip effect, a poor plan can easily propagate to the whole supply chain areas. The impact of a poor plan on the overall business may be huge. It can cause cycles of excessive inventory and severe backlogs, poor product forecasts, unbalanced capacities, poor customer service, uncertain production plans, and high backlog costs, or sometimes even lost sales. Ultimately the suppression of the bullwhip effect can only be achieved if the various nodes of the supply chain co-operate to some extend with each other. Therefore, it may important to investigate whether such co-operative attitudes can be engineered into the system, either via a form of contractual obligations between supply chain participants, or incentives following from the collective benefits which arise from smooth chain flows and increased levels of customer satisfaction.

Bibliography

[AA03] K. Albertson and J. Aylen, *Forecasting the behaviour of manufacturing inventory*, International Journal of Forecasting **19** (2003), 299–311.

[AM00] J. Andersson and J. Marklund, *Decentralized inventory control in a two-level distribution system*, European Journal of Operational Research **127** (2000), 483–506.

[AM02] E. G. Anderson and D. J. Morrice, *Capacity and management in queuing-based supply chains*, Proceedings of the 2002 Winter Simulation Conference (San Diego, USA), 2002.

[AS93] R. G. Askin and C. R. Standridge, *Modeling and analysis of manufacturing systems*, John Wiley & Sons, Inc., New York, NY, 1993.

[Axs93] S. Axsäter, *Exact snd approximate evaluation of batch-ordering policies for two-level inventory systems*, Operations Research **41** (1993), no. 4, 777–785.

[Axs98] ———, *Evaluation of installation stock based (R,Q)-policies for two-level inventory systems with poisson demand*, Operations Research **46** (1998), no. 3, S135–S145.

[BÖ2] C. Böhnlein, *Modellierung und simulation des bullwhip-effekts mit petrinetzen*, Würzburg, 2002.

[Bal04] R. H. Ballou, *Business logistics/supply chain management*, 5th ed., Pearson Prentice Hall, Upper Saddle River, NJ, 2004.

[BBP06] D. Bauso, F. Blanchini, and R. Pesenti, *Robust control strategies for multiinventory systems with average flow constraints*, automatica **42** (2006), 1255–1266.

[BC98] M. P. Baganha and M. A. Cohen, *The stabilizing effect of inventory in supply chains*, Operations Research **46** (1998), no. 3, S72–S83.

[BDS03] D. N. Burt, D. Dobler, and S. L. Starling, *World class supply management : The key to supply chain management*, 7th ed., McGraw-Hill/Irwin, New York, NY, 2003.

[BHT+96] M. C. Bonney, M. A. Head, C. C. Tien, N. Huang, and R. J. Barson, *Inventory and enterprise integration*, International Journal of Production Economics **45** (1996), 91–99.

[BK04] S. Benjaafar and J. S. Kim, *On the effect of product variety in productioninventory systems*, Annals of Operations Research **126** (2004), 71–101.

[Bli86] Blinder, *Can the production smoothing model of inventory behavior be saved?*, Quarterly journal of economics **101** (1986), 431–454.

[Bos01] S. K. Bose, *An introduction to queueing systems*, Kluwer Academic Publishers, Massachusetts, 2001.

[BRF+03] M. W. Braun, D. E. Rivera, M. E. Flores, W. M. Carlyle, and K. G. Kempf, *A model predictive control framework for robust management of multi-product, multi-echelon demand networks*, Annual Reviews in Control **27** (2003), 229–245.

[CA98] M. C. Caramanis and O. M. Anli, *Manufacturing supply chain coordination through synergistic decentralized decision making*, Proceed-

ings of Rensselaer's International Conference on Agile, Intelligent, and Computer-Integrated Manufacturing (Troy, NY), 1998.

[Cap85] A. S. Caplin, *The variability of aggregate demand with (S,s) inventory policies*, Econometrica **53** (1985), no. 6, 1395–1409.

[CDRSL00] F. Chen, Z. Drezner, J. K. Ryan, and D. Simchi-Levi, *Quantifying the bullwhip effect in a simple supply chain the impact of forecasting lead tiimes and information*, Management Science **46** (2000), no. 3, 436–443.

[CG04] C. Chandra and J. Grabis, *Application of multi-steps forecasting for restraining the bullwhip effect and improving inventory performance under autoregressive demand*, European Journal of Operational Research **166** (2004), 337–350.

[Che98] F. Chen, *Echelon reorder points, installation reorder points, and the value of centralized demand information*, Management Science **44** (1998), no. 12, S221–S234.

[CL99] C. G. Cassandras and S. Lafortune, *Introduction to discrete event systems*, Kluwer Academic Publishers, Massachusetts, 1999.

[Cou03] Supply Chain Council, *Supply-chain operations reference model. SCOR version 6.0*, SCC, 2003.

[CPA99] M. C. Caramanis, I. C. Paschalidis, and O. M. Anli, *A framework for the decentralized control of manufacturing enterprises*, Proceedings of DARPA-JFACC Symposium on Advances in Enterprise Control (San Diego), 1999, pp. 99–109.

[CRSL00] F. Chen, J. K. Ryan, and D. Simchi-Levi, *The impact of exponential smoothing forecasts on the bullwhip effect*, Naval Research Logistics 47 (2000), 269–286.

[CS60] A. J. Clark and H. Scarf, *Optimal policies for a multi-echelon inventory problem*, Management Science **6** (1960), 475–490.

[Dag03] C. F. Daganzo, *A theory of supply chains*, Springer Verlag, Berlin, 2003.

[Dag04] ———, *On the stability of supply chains*, Operations Research **52** (2004), 909–921.

[DDLT02] J. Dejonckheere, S. M. Disney, M. R. Lambrecht, and D. R. Towill, *Transfer function analysus of forecasting induced bullwhip in supply chains*, International Journal of production economics **78** (2002), 133–144.

[DDLT03] ———, *Production, manufacturing and logistics measuring and avoiding the bullwhip effect: A control theoretic approach*, European Journal of Operational Research **147** (2003), 567–590.

[DDLT04] ———, *The impact of information enrichment on the bullwhip effect in supply chains: A control engineering perspective*, European Journal of Operational Research **153** (2004), 797–811.

[DDVK00] G. Doumeingts, Y. Ducq, B. Vallespir, and S. Kleinhans, *Production management and enterprise modelling*, Computers in Industry **42** (2000), 245–263.

[DHP+93] F. Dicesare, G. Harhalakis, J. M. Proth, M. Silva, and F. B. Vernadat, *Practice of petri nets in manufacturing*, Chapman and Hall, London, 1993.

[dKG03] A. G. d. Kok and S. C. Graves, *Supply chain management : Design, coordination and operation*, Elsevier, Amsterdam, 2003.

[dKJvD+05] T. d. Kok, F. Janssen, J. v. Doremalen, E. v. Wachem, M. Clerkx, and W. Peeters, *Philips electronics synchronizes its supply chain to end the bullwhip effect*, Interfaces **35** (2005), no. 1, 37–48.

[DT03] S. M. Disney and D. R. Towill, *On the bullwhip and inventory variance produced by an ordering policy*, Omega **31** (2003), 157–167.

[DV85] M. H. A Davies and R.B. Vinter, *Stochastic modeling and control*, Chapman and Hall, London, England, 1985.

[EGE02] I. Elmahi, O. Grunder, and A. Elmoundni, *A petri net approach for the evaluation and command of a supply chain using the max plus algebra*, Proceedings of the 2nd IEEE International Conference on Systems, Man and Cybernetics (Hammamet, Tunisia), vol. 4, 2002.

[Eng78] D. W. Engels, *Alexander the Great and the logistics of the Macedonian army*, University of California Press, Los Angeles, 1978.

[EP97] E. Esposito and R. Passaro, *Material requirement planning and the supply chain at Alenia aircraft*, European Journal of Purchasing and Supply Management **3** (1997), no. 1, 43–51.

[fCPN] Computer Tool for Coloured Petri Nets, http://wiki.daimi.au.dk/cpntools/cpntools.wiki.

[Fia05] P. Fiala, *Information sharing in supply chains*, Omega **33** (2005), 419–423.

[FM84] P. Falster and R. B. Mazumder, *Modelling production management systems*, North-Holland, Amsterdam, 1984.

[For61] J. Forrester, *Industrial dynamics*, MIT Press, Cambridge,MA, 1961.

[Fre01] F. Frederix, *An extended ERP methodology for the discrete manufacturing industry*, European Journal of Operational Research **129** (2001), 317–325.

[FZ84] A. Federgruen and P. Zipkin, *Computational issues in an infinite horizon, multi-echelon inventory model*, Management Science **32** (1984), no. 4, 818–836.

[Gaa06] G. Gaalamn, *Bullwhip reduction for arma demand: The proportional order-up-to policy versus the full-state-feedback policy*, automatica **42** (2006), 1283–1290.

[GBS01] R. Ganeshan, T. Boone, and A. J. Stenger, *The impact of inventory and flow planning parameters on supply chain performance an exploratory study*, International Journal of Production Economics **71** (2001), 111–118.

[GH98] D. Gross and C. M. Harris, *Fundamentals of queueing theory*, John Wiley & Sons, Inc., New York, NY, 1998.

[Gil05] K. Gilbert, *An arima supply chain model*, Management Science **51** (2005), no. 2, 305–310.

[GKZ] S. C. Graves, A. H. G. Rinnooy Kan, and P. H. Zipkin (eds.).

[GP04] I. Giannoccaro and P. Pontrandolfo, *Supply chain coordination by revenue sharing contracts*, International Journal of Production Economics **89** (2004), 131–139.

[HD04] T. Hosoda and S. M. Disney, *An analysis of a three echelon supply chain model with minimum mean squared error forecasting*, Proceedings of the Second World Conference on POM and 15th Annual POM Conference (Cancun,Mexico), 2004.

[HEC00] M. M. Helms, L. P. Etkin, and S. Chapman, *Supply chain forecasting: Collaborative forecasting supports supply chain management*, Business Process Management **6** (2000), no. 5, 392–407.

[Hei02] J. Heikkilä, *From supply to demand chain management: efficiency and customer satisfaction*, Journal of Operations Management **20** (2002), 747–767.

[Hen06] J. Hennet, *A bimodal scheme for multi-stage production and inventory control*, automatica **42** (2006), 793–805.

[HJ90] R. A. Horn and C. R. Johnson, *Matrix analysis*, Cambridge University Press, England, 1990.

[HJ95] ———, *Topics in matrix analysis*, Cambridge University Press, England, 1995.

[HW63] G. Hadley and T. Whitin, *Analysis of inventory systems*, Prentice Hall, New Jersey, 1963.

[Jar91] J. Jarrett, *Business forecasting methods*, Basil Blackwell Ltd, Oxford, 1991.

[Jen97] K. Jensen, *Coloured Petri Nets. Basic concepts, analysis methods and practical use. Volume 1, 2, 3*, Springer-Verlag, Berlin, 1997.

[JK04] Z. Jemaï and F. Karaesmen, *Decentralized inventory control in a two-stage capacitated supply chain*, Laboratoire MGSI (Modélisation et Génie des Systèmes Industriels), IUT de Montreuil, France, and Koç University, Istanbul, Turkey, 2004.

[Kah87] J. Kahn, *Inventories and the volatility of production*, The American economic review **77** (1987), no. 4, 667–679.

[KCHH06] J. G. Kim, D. Chatfield, T. P. Harrison, and J. C. Hayya, *Quantifying the bullwhip effect in a supply chain with stochastic lead time*, European Journal of Operational Research **173** (2006), 617–636.

[KCJ98] L. M. Kristensen, S. Christensen, and K. Jensen, *The practitioner's guide to coloured petri nets*, International Journal of Software tools for technology transfer **2** (1998), 98–132.

[KH03] S. Kima and D. Ha, *A JIT lot-splitting model for supply chain management: Enhancing buyersupplier linkage*, International Journal of Production Economics **86** (2003), 1–10.

[Kij02] M. Kijima, *Stochastic processes with applications to finance*, Chapman & Hall/CRC, Florida, 2002.

[Kim00] B. Kim, *Coordinating an innovation in supply chain management*, European Journal of Operational Research **123** (2000), 568–584.

[KK02] K. H. Kim and J. Kim, *Determining load patterns for the delivery of assembly components under JIT systems*, International Journal of Production Economics **77** (2002), 25–38.

[KMPS05] E. Krauth, H. Moonen, V. Popova, and M. Schut, *Performance indicators in logistics service provision and warehouse management A literature review and framework*, Distributed Engine for Advanced Logistics, working paper, 2005.

[LB93] H. L. Lee and C. Billington, *Material management in decentralized supply chains*, Operations Research **41** (1993), no. 5, 835–847.

[Lee96] H. L. Lee, *Effective inventory and service management through product and process redesign*, Operations Research **44** (1996), no. 1, 151–159.

[LKL02] Z. Li, A. Kumar, and Y. G. Lim, *Supply chain modelling - a coordination approach*, Integrated Manufacturing Systems **13** (2002), 551–561.

[LKvdA04] E. R. Liu, A. Kumar, and W. v. d. Aalst, *A formal modeling approach for supply chain event management*, Proceedings of the 14th Workshop on Information Technologies and Systems (Washington, DC) (A. Dutta and P. Goes, eds.), 2004, pp. 110–115.

[LLBC00] H. Li, P. L. Lee, P. Bahri, and I. T. Cameron, *Decentralized control design for nonlinear plants: a ν-metric approach*, Computers and Chemical Engineering **24** (2000), 273–278.

[LPW97a] H. L Lee, V. Padmanabhan, and S. Whang, *The bullwhip effect in supply chains*, Sloan Management Review **38** (1997), no. 3, 93–102.

[LPW97b] ———, *Information distortion in a supply chain. the bullwhip effect*, Management Science **43** (1997), 546–558.

[LSS03] R. A. Lancioni, M. F. Smith, and H. J. Schau, *Strategic internet application trends in supply chain management*, Industrial Marketing Management **32** (2003), 211–217.

[LST00] H. L. Lee, K. C. So, and C. S. Tang, *The value of information sharing in a two-level supply chain*, Management Science **46** (2000), 626–643.

[LWJ+04] P. Lin, D. S. Wong, S. Jang, S. Shieh, and J. Chu, *Controller design and reduction of bullwhip for a model supply chain system using z-transform analysis*, Journal of Process Control **14** (2004), 487–499.

[Mar97] J. S. Martinich, *Production and operations management : an applied modern approach*, John Wiley & Sons, Inc., New York, NY, 1997.

[MB04] J. A. D. Machuca and R. P. Barajas, *The impact of electronic data interchange on reducing bullwhip effect and supply chain inventory costs*, Transportation Research Part E: Logistics and Transportation Review **3** (2004), 209–228.

[Men99] J. T. Mentzer (ed.), *Supply chain management*, Berrett-Koehler Publishers, Inc., San Francisco, CA, 1999.

[Mer01] J. R. Meredith, *Hopes for the future of operations management*, Journal of Operations Management **19** (2001), 397–402.

[Met97] R. Metters, *Quantifying the bullwhip effect in supply chains*, Journal of Operations Management **15** (1997), 89–100.

[MHM03] R. McIvor, P. Humphreys, and L. McCurry, *Electronic commerce: supporting collaboration in the supply chain?*, Journal of Materials Processing Technology **139** (2003), 147–152.

[Mir06] G. Miragliotta, *Layers and mechanisms: A new taxonomy for the bullwhip effect*, International Journal of production economics **104** (2006), 365–381.

[ML88] M. C. Morris and H. L. Lee, *Strategic analysis of integrated production-distribution systems: Models and methods*, Operations research **36** (1988), no. 2, 216–228.

[MNPV04] D. Makajić-Nikolić, B. Panić, and Mirko Vujošević, *Bullwhip effect and supply chain modelling and analysis using cpn-tools*, Proceedings of the 5th Workshop and Tutorial on Practical Use of Coloured Petri Nets and the CPN-Tools (Aarhus, Denmark) (K. Jensen, ed.), 2004, pp. 219–234.

[MP95] H. P. Marvel and J. Peck, *Demand uncertainty and returns policies*, International economic review **36** (1995), no. 3, 691–714.

[MPvLV02] Y. A. Merkuryev, J. J. Petuhova, R. v. Landeghem, and S. Vansteenkiste, *Simulation-based analysis of the bullwhip effect under different information sharing strategies*, Proceedings of the 14th European Simulation Symposium (Dresden,Germany) (A. Verbraeck and W. Krug, eds.), 2002.

[MT80] J. A. Muckstadt and L. J. Thomas, *Are multi-echelon inventory methods worth implementing in systems with low-demand-rate items?*, Management Science **26** (1980), no. 5, 483–494.

[MVJE05] D. J. Morrice, R. A. Valdez, P. Chida-(Jr.) J, and M. Eido, *Discrete event simulation in supply chain planning and inventory control at Freescale semiconductor inc.*, Proceedings of the 2005 Winter Simulation Conference (M. E. Kuhl, N. M. Steiger, F. B. Armstrong, and Joines J. A, eds.), 2005, pp. 1718–1724.

[MWH98] S. Makridakis, S. C. Wheelwright, and R. J. Hyndman, *Forecasting : methods and applications*, John Wiley & Sons, Inc., New York, NY, 1998.

[oTC] MATLAB: The Language of Technical Computing, http://www.mathworks.com/products/matlab.

[Pet62] C.A. Petri, *Kommunikation mit automaten*, Ph.D. thesis, University of Darmstadt, Germany, 1962.

[PGYT00] E. Perea, I. Grossmann, E. Ydstie, and T. Tahmassebi, *Dynamic modeling and classical control theory for supply chain management*, Computers & Chemical Engineering **24** (2000), 1143–1149.

[PLGYT01] E. Perea-López, I. E. Grossmann, B. E. Ydstie, and T. Tahmassebi, *Dynamic modeling and decentralized control of supply chains*, Industrial & Engineering Chemical Research **40** (2001), 3369–3383.

[PLYG03] E. Perea-López, B. E. Ydstie, and I. E. Grossmann, *A model predictive control strategy for supply chain optimization*, Computers & Chemical Engineering **27** (2003), 1201–1218.

[Poi99] C. C. Poirier, *Advanced supply chain management : how to build a sustained competitive advantage*, Berrett-Koehler Publishers, Inc., San Francisco, CA, 1999.

[RB01] C. E. Riddalls and S. Bennett, *The optimal control of batched production and its effect on demand amplification*, International Journal of Production Economics **72** (2001), 159–168.

[RB06] L. Rodrigues and E. Boukas, *Piecewise-linear H_∞ controller synthesis with applications to inventory control of switched production systems*, automatica **42** (2006), 1245–1254.

[Ros89] K. Rosling, *Optimal inventory policies for assembly systems under random demands*, Operations Research **37** (1989), no. 4, 565–579.

[RV99a] N. R. S. Raghavan and N. Viswanadham, *Lead time models for analysis of supply chains*, Proceedings of the Second AEGEAN International Conference on Analysis and Modeling of Manufacturing Systems (Tinos Island, Greece), 1999.

[RV99b] ———, *Performance analysis of supply chains using queueing models*, Proceedings of POMS'99: International conference on Operations Management for the Global Economy (Delhi, India), 1999.

[Sch03] P. Schönsleben, *Integral logistics management : planning and control of comprehensive supply chains*, 2nd ed., St. Lucie Press, Boca Raton, FL, 2003.

[Sei03] D. Seifert, *Collaborative planning, forecasting and replenishment : how to create a supply chain advantage*, Amacom, New York, NY, 2003.

[She04] J. B. Sheu, *A multi-layer demand-responsive logistics control methodology for alleviating the bullwhip effect of supply chains*, European Journal of Operational Research **161** (2004), 797–811.

[SLKSL03] D. Simchi-Levi, P. Kaminsky, and E. Simchi-Levi, *Designing and managing the supply chain*, 2nd ed., McGraw-Hill/Irwin, New York, NY, 2003.

[SR05] H. X. Sun and Y. T. Ren, *The impact of forecasting methods on bullwhip effect in supply chain management*, Proceedings of the IEEE International Engineering Management Conference (St. John's, Newfoundland, Canada), vol. 1, September 2005, pp. 215–219.

[SRP+04] L. Sabato, P. Renna, G. Perrone, M. Bruccoleri, and U. La Commare, *Evaluating the impact of demand and inventory management policies on bullwhip effect in production networks*, Proceedings of the ISOMA 2004 9th International Symposium on manufacturing and applications (Seville, Spain), June 2004.

[SS01] D. Schechter and G. Sander, *Delivering the goods : the art of managing your supply chain*, John Wiley & Sons, Inc., Hoboken, NJ, 2001.

[Ste89] J. D. Sterman, *Modeling managerial behaviour: Misperceptions of feedback in a dynamic decision making experiment*, Management Science **35** (1989), no. 3, 321–339.

[SWR06] J. D. Schwartz, W. Wang, and D. E. Rivera, *Simulation-based optimization of process control policies for inventory management in supply chains*, automatica **42** (2006), 1311–1320.

[Tah03] H. A. Taha, *Operations research. An introduction*, 7th ed., Prentice Hall, New Jersey, 2003.

[TC04] S. Terzi and S. Cavalieri, *Simulation in the supply chain context: a survey*, Computers in Industry **53** (2004), 3–16.

[VBWJ05] T. E. Vollmann, W. L. Berry, D. C. Whybark, and F. R. Jacobs, *Manufacturing planning & control systems for supply chain management*, McGraw-Hill, New York, NY, 2005.

[vdVHH04] P. v. d. Vlist, J. J. E. M. Hoppenbrouwers, and H. M. H. Hegge, *Extending the enterprise through multi-level supply control*, International Journal of Production Economics **53** (2004), 209–228.

[vHSWY03] G. v. Houtum, A. Scheller-Wold, and J. Yi, *Optimal control of serial, multi-echelon inventory/production systems with periodic batching (working paper)*, Laboratoire MGSI (Modélisation et Génie des Systèmes Industriels), IUT de Montreuil, France, and Koç University, Istanbul, Turkey, 2003.

[vLB02] R. v. Landeghem and C. M. Bobeanu, *Formal modelling of supply chain: An incremental approach using Petri nets*, Proceedings of the 14th European Simulation Symposium (Dresden, Germany) (A. Verbraeck and W. Krug, eds.), 2002.

[vMP04] M. v. Mevius and R. Pibernik, *Process management in supply chains - (a) new Petri-net based approach*, Proceedings of the 37th Hawaii International Conference on System Sciences (Hawaii), vol. 3, 2004.

[VRZ00] M. Vroblefski, R. Ramesh, and S. Zionts, *Efficient lot-sizing under a differential transportation cost structure for serially distributed warehouses*, European Journal of Operational Research **127** (2000), 574–593.

[WW03] W. Walsh and M. P. Wellman, *Decentralized supply chain formation: A market protocol and competitive equilibrium analysis*, Journal of Artificial Intelligence Research **19** (2003), 513–567.

[WZ05] N. Watson and Y. Zheng, *Decentralized serial supply chains subject to order delays and information distortion: Exploiting real-time sales data*, Manufacturing & Service Operations Management **7** (2005), no. 2, 152–168.

[XDE01] K. Xu, Y. Dong, and P. T. Evers, *Towards better coordination of the supply chain*, Transportation Research Part E: Logistics and Transportation Review **37** (2001), 35–54.

[Zha04a] X. Zhang, *Evolution of arma demand in supply chains*, Manufacturing & Service Operations Management **6** (2004), no. 2, 195–198.

[Zha04b] ———, *The impact of forecasting methods on the bullwhip effect*, International Journal of Production Economics **88** (2004), 15–27.

[Zip00] P. Zipkin, *Foundations of inventory management*, McGraw-Hill/Irwin, New York, NY, 2000.

[ZT04] K. Zhu and U. W. Thonemann, *Modeling the benefits of sharing future demand information*, Operations Research **52** (2004), no. 6, 136–147.

[ZXL02] X. Zhao, J. Xie, and J. Leung, *The impact of forecasting model selection on the value of information sharing in a supply chain*, European Journal of Operational Research **142** (2002), 321–344.

Appendix A

Proofs of Lemmas and Remarks

A.1 Proof of Lemma 2.2.1

The model deals with n items ($n > 1$), whose individual inventory fluctuations do not allow shortages. Define for item $i, i = 1, 2, \ldots, n$ (see [Tah03]),

D_i : Demand rate (units per unit time)

K_i : Setup cost

h_i : Unit holding cost per unit time

y_i : Order quantity (number of units)

a_i : Storage area requirement per inventory unit

A : Maximum available storage area for all n items

Under the assumption of no shortages, the mathematical model representing the inventory situation is given as:

$$\text{Minimise TCU}(y_1, y_2, \ldots, y_n) = \sum_{i=1}^{n} \left(\frac{K_i D_i}{y_i} + \frac{h_i y_i}{2} \right)$$

subject to:

$$\sum_{i=1}^{n} a_i y_i \leq A$$

$$y_i > 0, i = 1, 2, \ldots, n$$

The steps for the solution of the model are:

- Step 1. Compute the *unconstrained* optimal values of the order quantities as:

$$y_i^* = \sqrt{\frac{2K_iD_i}{h_i}}, i = 1, 2, \ldots, n$$

- Step 2. Check if the unconstrained optimal values y_i^* satisfy the storage constraint. If this is the case, the solution $y_i^*, i = 1, 2, \ldots, n$ is optimal. Otherwise, go to step 3.

- Step 3. The storage constraint must be satisfied in equation form. Use the Lagrange multipliers method to determine the constrained optimal values of the order quantities.

In step 3, the Lagrangean function is formulated as:

$$\begin{aligned} L(\lambda, y_1, y_2, \ldots, y_n) &= TCU(y_1, y_2, \ldots, y_n) - \lambda \left(\sum_{i=1}^{n} a_i y_i - A \right) \\ &= \sum_{i=1}^{n} \left(\frac{K_iD_i}{y_i} + \frac{h_i y_i}{2} \right) - \lambda \left(\sum_{i=1}^{n} a_i y_i - A \right) \end{aligned}$$

where $\lambda < 0$ is the Lagrange multiplier.

Since the Lagrangean function is convex, the optimal values of y_i and λ are determined from the following necessary condition:

$$\begin{aligned} \frac{\partial L}{\partial y_i} &= -\frac{K_iD_i}{y_i^2} + \frac{h_i}{2} - \lambda a_i = 0 \\ \frac{\partial L}{\partial \lambda} &= -\sum_{i=1}^{n} a_i y_i + A = 0 \end{aligned}$$

The second equation shows that the storage constraint must be satisfied in equation form at the optimum.

From the first equation:

$$y_i^* = \sqrt{\frac{2K_iD_i}{h_i - 2\lambda^* \alpha_i}}$$

The formula provides that y_i^* is dependent on the value of λ^*. For $\lambda^* = 0$, y_i^* gives the unconstrained solution. Note that the application of the above method is correct in this case because $TCU(y_1, y_2, \ldots, y_n)$ is convex and the problem has a

single linear constraint and hence a convex solution space. However, this method may not be correct under other conditions or when the problem has more than one constraint.

A.2 Proof of Lemma 3.4.1

Consider the $(j+1)-th$ node state-space model ($j \geq 1$) with corresponding A-matrix given by A_{2j+1}. The proof is by induction on j. For $j = 1$, the eigenvalues of A_3 may be easily calculated as $\{1-k_1, 0, 0\}$. Thus A_3 is asymptotically stable if and only if $-1 < 1-k_1 < 1$ or equivalently if and only if $0 < k_1 < 2$. Next assume that the eigenvalues of A_{2j-1} are given by $\{1-k_1, \ldots, 1-k_{j-1}, 0, \ldots, 0\}$ (with j zero eigenvalues) for $j \geq 2$, so that A_{2j-1} is asymptotically stable if and only if $(k_1, \ldots, k_{j-1}) \in (0,\ 2)^{j-1}$. Introduce the permutation matrix Q_j, resulting from the interchange of the $(2j-1)$-th and $2j$-th rows and columns of the unit matrix I_{2j+1}. Then,

$$Q_j A_{2j+1} Q_j = \begin{pmatrix} A_{2j-1} & 0 & 0 \\ a'_{21} & 1 & 1 \\ a'_{31} & -k_j & -k_j \end{pmatrix}$$

where a'_{21} and a'_{31} are irrelevant for our present purposes. Thus, since the transformation by Q_j leaves the eigenvalues invariant, the spectrum of A_{2j+1} is given as:

$$\begin{aligned} \lambda(A_{2j+1}) &= \lambda(A_{2j-1}) \cup \{1-k_j, 0\} \\ &= \{1-k_1, \ldots, 1-k_j, 0, \ldots, 0\} \end{aligned}$$

in which the zero eigenvalue has algebraic multiplicity $j+1$, and hence A_{2j+1} is asymptotically stable if and only if $(k_1, \ldots, k_j) \in (0,\ 2)^j$. This completes the inductive argument. In general, A_{2m+1} has m real eigenvalues at $1-k_j$, $j = 1, 2, \ldots, m$, and another $m+1$ eigenvalues at the origin.

A.3 Proof of Lemma 3.4.2

It follows from standard results on eigenvalues of Kronecker products [HJ90] that $A \otimes A$ has eigenvalues $\{\lambda_i(A)\lambda_j(A) : i, j \in \{1, 2, \ldots, n\}\}$. Thus from Lemma 3.4.1, the eigenvalues of $A \otimes A$ are the m^2 products $\{(1-k_i)(1-k_j) : i, j \in \{1, 2, \ldots, m\}\}$ and zero (with multiplicity $n^2 - m^2$). Hence the eigenvalues of $I_{n^2} - A \otimes A$ are the m^2 real numbers $1 - (1-k_i)(1-k_j) = k_i + k_j - k_i k_j$ as i and j vary over the set $\{1, 2, \ldots, m\}$, and one (with multiplicity $n^2 - m^2$). Thus $I_{n^2} - A \otimes A$ is singular if and only if $(1-k_i)(1-k_j) = 1$ for some pair (i, j) such that $1 \leq i, j \leq m$. Note that this matrix is singular if $k_i = 0$ or $k_i = 2$ for some i, and certainly non-singular if all k_i lie in the interval $(0,\ 2)$.

A.4 Proof of Lemma 3.4.3

Follows immediately since $I_{n^2} - A \otimes A$ is non-singular for all $k \in \mathcal{K}_n$, while V and W have full row rank and column rank, respectively.

A.5 Proof of Remark 3.4.1

The decomposition of $Q_j A_{2j+1} Q_j$ and the fact that $A_{11} = A_{2j-1}$ follows directly from Lemma 3.4.1. Further note that

$$A_{21} = \begin{pmatrix} O_{2j-2} & -1 \\ O_{2j-2} & k_2 \end{pmatrix} \quad \text{and} \quad A_{22} = \begin{pmatrix} 1 & 1 \\ -k_j & -k_j \end{pmatrix}$$

so that both A_{21} and A_{22} have rank one. The fact that $Q_j B_{2j+1} = B_{2j+1}$ also follows immediately since the only non-zero element of B_{2j+1} is the second.

Since A is asymptotically stable for $(k_1, k_2, \ldots, k_j) \in (0, 2)^j$, the discrete-time Lyapunov equation $P - APA' - BB' = 0$ has a unique symmetric positive-semidefinite solution [DV85]. Using the indicated partitioning, this may be written as:

$$\begin{pmatrix} A_{11} & 0 \\ A_{21} & A_{22} \end{pmatrix} \begin{pmatrix} P_{11} & P_{12} \\ P_{12}' & P_{22} \end{pmatrix} \begin{pmatrix} A_{11}' & A_{21}' \\ 0 & A_{22}' \end{pmatrix} =$$
$$\begin{pmatrix} P_{11} & P_{12} \\ P_{12}' & P_{22} \end{pmatrix} + \begin{pmatrix} B_{2j-1} B_{2j-1}' & 0 \\ 0 & 0 \end{pmatrix}$$

which is equivalent to the three matrix equations:

$$\begin{aligned} P_{11} - A_{11}P_{11}A'_{11} &= B_{2j-1}B'_{2j-1} \\ P_{12} - A_{11}P_{12}A'_{22} &= A_{11}P_{11}A'_{21} \end{aligned}$$

and

$$P_{22} - A_{22}P_{22}A'_{22} = A_{21}P_{11}A'_{21} + A_{22}P'_{12}A'_{21} + A_{21}P_{12}A'_{22}$$

Note that the first of these is a discrete Lyapunov equation; since A_{11} is asymptotically stable the solution of this equation is unique, and hence $P_{11} = P_{2j-1}$. Moreover, since A_{11} and A_{22} are both asymptotically stable, the solutions of the second and third equations are also unique [DV85] and P_{22} is positive semidefinite. To show that P_{12} and P_{22} have both rank at most one, note that the second and third equations may be written as:

$$P_{12} = A_{11} \begin{pmatrix} P_{11} & P_{12} \end{pmatrix} \begin{pmatrix} A'_{21} \\ A'_{22} \end{pmatrix}$$

and

$$P_{22} = \begin{pmatrix} A_{21} & A_{22} \end{pmatrix} P \begin{pmatrix} A'_{21} \\ A'_{22} \end{pmatrix}$$

Now,

$$\begin{pmatrix} A_{21} & A_{22} \end{pmatrix} = \begin{pmatrix} O_{2j-2} & -1 & 1 & 1 \\ O_{2j-2} & k_j & -k_j & -k_j \end{pmatrix}$$

has rank one, and hence P_{12} and P_{22} have rank at most one. Finally, note that if $(k_1, k_2, \ldots, k_j) \in (0,2)^j$, A_{2j+1} is asymptotically stable and hence the Lyapunov equation:

$$P_{2j+1} - A_{2j+1}P_{2j+1}A'_{2j+1} - B_{2j+1}B'_{2j+1} = 0$$

has a unique solution. The recursive updating formula

$$P_{2j+1} = Q_j \begin{pmatrix} P_{2j-1} & P_{12} \\ P'_{12} & P_{22} \end{pmatrix} Q_j$$

now follows on noting that under the state-space transformation $A_{2j+1} \rightarrow Q_j A_{2j+1} Q_j$, $B_{2j+1} \rightarrow Q_j B_{2j+1} = B_{2j+1}$, the solution of the Lyapunov equation corresponding to (A_{2j+1}, B_{2j+1}) transforms as $P_{2j+1} \rightarrow Q_j P_{2j+1} Q_j$.

A.6 Proof of Lemma 4.3.2

The Frobenious Norm $\|\check{P}_5^{-1} - \acute{P}_5^{-1}\|_F^2$ can be defined as:

$$\begin{aligned}
\|\check{P}_5^{-1} - \acute{P}_5^{-1}\|_F^2 &= trace\{(\check{P}_5^{-1} - \acute{P}_5^{-1})(\check{P}_5^{-1} - \acute{P}_5^{-1})'\} \\
&= trace\{(\mathcal{D} - \mathbf{B}\check{k}_1 - \mathbf{A}\check{k}_1^2)(\mathcal{D}' - \mathbf{B}'\check{k}_1 - \mathbf{A}'\check{k}_1^2)\} \\
&= trace\{\mathcal{D}\mathcal{D}' - (\mathcal{D}\mathbf{B}' + \mathbf{B}\mathcal{D}')\check{k}_1 + \\
&\quad (-\mathcal{D}\mathbf{A}' - \mathbf{A}\mathcal{D}' + \mathbf{B}\mathbf{B}')\check{k}_1^2 + (\mathbf{B}\mathbf{A}' + \mathbf{A}\mathbf{B}')\check{k}_1^3 + \mathbf{A}\mathbf{A}'\check{k}_1^4\} \\
&= trace(\mathcal{D}\mathcal{D}') - trace(\mathcal{D}\mathbf{B}' + \mathbf{B}\mathcal{D}')\check{k}_1 + trace(\mathbf{B}\mathbf{B}' - \mathcal{D}\mathbf{A}' - \mathbf{A}\mathcal{D}')\check{k}_1^2 + \\
&\quad trace(\mathbf{B}\mathbf{A}' + \mathbf{A}\mathbf{B}')\check{k}_1^3 + trace(\mathbf{A}\mathbf{A}')\check{k}_1^4 \\
&= \alpha_0 + \alpha_1\check{k}_1 + \alpha_2\check{k}_1^{\,2} + \alpha_3\check{k}_1^{\,3} + \alpha_4\check{k}_1^{\,4}.
\end{aligned}$$

Appendix B

State space model computations

B.1 State space model for a three-stage series supply chain

Consider a supply chain consisting of three stages (e.g., Manufacturer, Distributor and Retailer) we can easily obtain the overall state space model by aggregating the models of all nodes. Following the notation given in chapter 3 this corresponds to setting $n = 2$. Recall that:

$$x_i(t+1) = A_i x_i(t) + B_{i,l} w_{i,l}(t) + B_{i,r} w_{i,r}(t)$$

and that $w_{i,l} = z_{i-1,r}$, $w_{i,r} = z_{i+1,l}$ then:

$$\begin{aligned} x_1(t+1) &= A_1 x_1(t) + B_{1,l} w_{1,l}(t) + B_{1,r} z_{2,l}(t) \\ &= A_1 x_1(t) + B_{1,l} w_{1,l}(t) + B_{1,r} C_{2,l} x_2(t) \end{aligned}$$

and

$$
\begin{aligned}
x_2(t+1) &= A_2x_2(t) + B_{2,l}w_{2,l}(t) + B_{2,r}w_{2,r}(t) \\
&= A_2x_2(t) + B_{2,l}z_{1,r}(t) + B_{2,r}w_{2,r}(t) \\
&= A_2x_2(t) + B_{2,l}(C_{1,r}x_1(t) + D_{1,rr}w_{1,r}(t)) + B_{2,r}w_{2,r}(t) \\
&= A_2x_2(t) + B_{2,l}C_{1,r}x_1(t) + B_{2,l}D_{1,rr}w_{1,r}(t) + B_{2,r}w_{2,r}(t) \\
&= A_2x_2(t) + B_{2,l}C_{1,r}x_1(t) + B_{2,l}D_{1,rr}z_{2,l}(t) + B_{2,r}w_{2,r}(t) \\
&= A_2x_2(t) + B_{2,l}C_{1,r}x_1(t) + B_{2,l}D_{1,rr}C_{2,l}x_2(t) + B_{2,r}w_{2,r}(t) \\
&= A_2x_2(t) + B_{2,l}C_{1,r}x_1(t) + B_{2,l}D_{1,rr}C_{2,l}x_2(t) + B_{2,r}w_{2,r}(t) \\
&= B_{2,l}C_{1,r}x_1(t) + (A_2 + B_{2,l}D_{1,rr}C_{2,l})x_2(t) + B_{2,r}w_{2,r}(t)
\end{aligned}
$$

Since $w_{2,r}(t) = x_\phi(t)$ and $x_\phi(t+1) = z_{2r}(t) = C_{2,r}x_2(t) + D_{2,rr}x_\phi(t)$

Hence the state space form for three-stage supply chain is given:

$$
\begin{pmatrix} x_1(t+1) \\ x_2(t+1) \\ x_\phi(t+1) \end{pmatrix} = \begin{pmatrix} A_1 & B_{1,r}C_{2,l} & 0 \\ B_{2,l}C_{1,r} & A_2 + B_{2,l}D_{1,rr}C_{2,l} & B_{2,r} \\ 0 & C_{2,r} & D_{2,rr} \end{pmatrix} \begin{pmatrix} x_1(t) \\ x_2(t) \\ x_\phi(t) \end{pmatrix} + \begin{pmatrix} B_{1,l} \\ 0 \\ 0 \end{pmatrix} w_{1l}(t)
$$

and the output $z_{1,l}$:

$$
z_{1,l} = \begin{pmatrix} C_{1,l} & 0 & 0 \end{pmatrix} \begin{pmatrix} x_1(t) \\ x_2(t) \\ x_\phi(t) \end{pmatrix} + \begin{pmatrix} 0 \end{pmatrix} w_{1,l}(t)
$$

and by substituting A, B, C and D matrices as have been calculated in equation 3.4.13 and 3.4.14:

$$
\begin{pmatrix} x_1(t+1) \\ x_2(t+1) \\ x_\phi(t+1) \end{pmatrix} = \begin{pmatrix} 1 & -1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & -1 & 1 \\ -k_1 & k_1 & 0 & -k_1 & 0 \\ 0 & 0 & -k_2 & k_2 & -k_2 \end{pmatrix} \begin{pmatrix} x_1(t) \\ x_2(t) \\ x_\phi(t) \end{pmatrix} + \begin{pmatrix} 0 \\ 1 \\ 0 \\ 0 \\ 0 \end{pmatrix} w_{1,l}
$$

and the output $z_{1,l}$:

$$z_{1,l}(t) = \begin{pmatrix} 0 & 1 & 0 & 0 & 0 \end{pmatrix} \begin{pmatrix} x_1(t) \\ x_2(t) \\ x_\phi(t) \end{pmatrix} + \begin{pmatrix} 0 \end{pmatrix} w_{1,l}(t)$$

B.2 State space model for a four-stage series supply chain

Intuitively, the four-nodes supply chain (e.g., Manufacturer, Distributor, intermediate Supplier, and Retailer) can be described by the following state space:

$$\begin{pmatrix} x_1(t+1) \\ x_2(t+1) \\ x_3(t+1) \\ x_\phi(t+1) \end{pmatrix} = \begin{pmatrix} A_1 & B_{1,r}C_{2,l} & 0 & 0 \\ B_{2,l}C_{1,r} & A_2 + B_{2,l}D_{1,rr}C_{2,l} & B_{2,r}C_{3,l} & 0 \\ 0 & B_{3,l}C_{2,r} & A_3 + B_{3,l}D_{2,rr}C_{3,l} & B_{3,r} \\ 0 & 0 & C_{3,r} & D_{3,rr} \end{pmatrix} \begin{pmatrix} x_1(t) \\ x_2(t) \\ x_3(t) \\ x_\phi(t) \end{pmatrix} + \begin{pmatrix} B_{1,l} \\ 0 \\ 0 \\ 0 \end{pmatrix} w_{1,l}(t)$$

and the output $z_{1,l}$:

$$z_{1,l}(t) = \begin{pmatrix} C_{1,l} & 0 & 0 & 0 \end{pmatrix} \begin{pmatrix} x_1(t) \\ x_2(t) \\ x_3(t) \\ x_\phi(t) \end{pmatrix} + \begin{pmatrix} 0 \end{pmatrix} w_{1,l}(t)$$

and the overall model of the four-stage supply chain:

$$\begin{pmatrix} x_1(t+1) \\ x_2(t+1) \\ x_3(t+1) \\ x_\phi(t+1) \end{pmatrix} = \begin{pmatrix} 1 & -1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & -1 & 0 & 1 & 0 \\ -k_1 & k_1 & 0 & -k_1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & -1 & 1 \\ 0 & 0 & -k_2 & k_2 & 0 & -k_2 & 0 \\ 0 & 0 & 0 & 0 & -k_3 & k_3 & -k_3 \end{pmatrix} \begin{pmatrix} x_1(t) \\ x_2(t) \\ x_3(t) \\ x_\phi(t) \end{pmatrix} + \begin{pmatrix} 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix} w_{1,l}(t)$$

and the output $z_{1,l}$:

$$z_{1,l}(t) = \begin{pmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 0 \end{pmatrix} \begin{pmatrix} x_1(t) \\ x_2(t) \\ x_3(t) \\ x_\phi(t) \end{pmatrix} + \begin{pmatrix} 0 \end{pmatrix} w_{1,l}(t)$$

Note that the model is driven by $O_{0,1}$, which represents customer's demand.

The general n-node model shown in Figure 3.3 can be aggregated as:

$$x(t+1) = \Phi x(t) + \Gamma w_{1,l}, \;\; x(t) = [x_1'(t)\; x_2'(t)\; \ldots\; x_n'(t)\; x_{n+1}'(t)]'$$

where,

$$\Phi = \begin{pmatrix} \Phi_{11} & \Phi_{12} & 0 & \cdots & 0 \\ \Phi_{21} & \Phi_{22} & \Phi_{23} & \ddots & \vdots \\ 0 & \Phi_{32} & \ddots & \ddots & 0 \\ \vdots & \ddots & \ddots & \Phi_{nn} & \Phi_{n\,n+1} \\ 0 & \cdots & 0 & \Phi_{n+1\,n} & \Phi_{n+1\,n+1} \end{pmatrix} \qquad \Gamma = \begin{pmatrix} \Gamma_1 \\ \Gamma_2 \\ \vdots \\ \Gamma_n \\ \Gamma_{n+1} \end{pmatrix}$$

More specifically, the elements of Φ and Γ are defined as:

$$\Phi_{ij} = \begin{cases} A_i + B_{i,l} D_{i-1,rr} C_{i,l} & \text{for } i = j; \quad i = 1, 2, \ldots, n+1 \\ B_{i,r} C_{i+1,l} & \text{for } i = j-1; \quad i = 1, 2, \ldots, n \\ B_{i,l} C_{i-1,r} & \text{for } i = j+1; \quad i = 2, 3, \ldots, n+1 \\ 0 & \text{for } |i-j| > 1. \end{cases}$$

and

$$\Gamma_i = B_{i,l} \quad \text{if } i = 1, \quad \Gamma_i = 0 \text{ otherwise.}$$

Note that this state-space model has been derived under the simplifying assumption that $D_{i,ll} = 0$ for all $i = 1, 2, \ldots, n+1$ and $D_{i,lr} = D_{i,rl} = 0$ for all $i = 1, 2, \ldots, n$ (we also define $D_{0,rr} = 0$). These relations actually hold for the concrete supply chain model which is presented in this thesis.

Appendix C

Simulation reports for chapter 6

C.1 Simulation results for Scenario 1

C.1.1 Statistical results for locations usage

The following Table C.1 depicts the usage percentages of each location per day for the simulation period of 35 days.

Table C.1: Percentage usage (%) in each location for Scenario 1

	Annealing	Cold-rolling	BWG	High-Bay
Day1:	89.9	0.0	0.0	1.1
Day2:	99.7	0.0	0.0	5.3
Day3:	100.0	0.0	0.0	9.7
Day4:	99.7	0.0	0.0	15.4
Day5:	100.0	0.0	0.0	21.2
Day6:	99.7	0.0	0.0	27.2
Day7:	100.0	0.0	0.0	32.2
Day8:	99.7	0.0	0.0	33.6
Day9:	100.0	0.0	0.0	33.6
Day10:	99.8	0.0	0.0	33.6
Day11:	99.9	0.0	0.0	33.6
Day12:	99.8	0.0	0.0	33.6
Day13:	99.9	0.0	0.0	33.6
Day14:	99.8	0.0	0.0	33.6
Day15:	99.9	6.9	0.0	33.6
Day16:	100.0	12.5	0.0	33.6
Day17:	99.7	18.1	0.0	33.5
Day18:	99.9	34.4	0.0	33.5
Day19:	99.8	3.8	25.0	31.3
Day20:	99.9	42.0	0.0	30.7
Day21:	99.8	13.9	92.0	28.2
Day22:	99.9	27.8	17.0	24.5
Day23:	99.8	43.8	0.0	23.9
Day24:	99.9	13.2	47.6	22.7
Day25:	99.8	0.0	67.7	15.5
Day26:	100.0	0.0	0.0	13.2
Day27:	99.8	0.0	0.0	14.4
Day28:	69.1	0.0	0.0	15.8
Day29:	51.2	0.0	0.0	16.3
Day30:	0.2	0.0	0.0	17.7
Day31:	0.0	0.0	0.0	17.7
Day32:	0.0	0.0	0.0	17.7
Day33:	0.0	30.6	0.0	17.6
Day34:	0.0	0.0	19.4	16.2
Day35:	0.0	0.0	0.0	15.9

C.1.2 Statistical results for crane moves at each location

Table C.2 shows the number of crane movements at each location per day for the simulation period of 35 days.

Table C.2: Crane moves at each location for Scenario 1

	Coil entry	Annealing	Cold-rolling input	Cold-rolling output	BWG
Day1:	26	20	0	0	0
Day2:	20	24	0	0	0
Day3:	19	0	0	0	0
Day4:	28	24	0	0	0
Day5:	24	0	0	0	0
Day6:	28	24	0	0	0
Day7:	19	0	0	0	0
Day8:	4	24	0	0	0
Day9:	0	0	0	0	0
Day10:	0	16	0	0	0
Day11:	0	8	0	0	0
Day12:	0	16	0	0	0
Day13:	0	8	0	0	0
Day14:	0	16	0	0	0
Day15:	0	8	20	20	0
Day16:	0	0	20	20	0
Day17:	0	24	27	25	0
Day18:	0	8	33	35	0
Day19:	0	16	4	2	12
Day20:	0	8	36	38	0
Day21:	0	16	20	20	24
Day22:	0	8	40	40	6
Day23:	0	16	48	46	0
Day24:	0	8	12	14	21
Day25:	0	16	0	0	27
Day26:	0	0	0	0	0
Day27:	0	8	0	0	0
Day28:	0	4	0	0	0
Day29:	0	4	0	0	0
Day30:	0	4	0	0	0
Day31:	0	0	0	0	0
Day32:	0	0	0	0	0
Day33:	0	0	40	40	0
Day34:	0	0	0	0	8
Day35:	0	0	0	0	0
Total:	: 168	328	300	300	98

C.2 Simulation results for Scenario 2

C.2.1 Statistical results for locations usage

Table C.3 depicts the usage percentages of each location per day for the simulation period of 29 days.

C.2.2 Statistical results for crane moves at each location

Table C.4 shows the number of crane movements at each location per day for the simulation period of 29 days.

Table C.3: Percentage usage (%) in each location for Scenario 2

	Annealing	Cold-rolling	BWG	High-Bay
Day1:	87.2	0.0	0.0	0.5
Day2:	99.7	0.0	0.0	4.4
Day3:	99.9	0.0	0.0	8.8
Day4:	99.7	0.0	0.0	14.5
Day5:	99.9	0.0	0.0	20.3
Day6:	99.7	0.0	0.0	26.3
Day7:	100.0	0.0	0.0	31.3
Day8:	99.8	0.0	0.0	32.7
Day9:	99.8	0.0	0.0	32.7
Day10:	99.8	0.0	0.0	32.7
Day11:	99.8	0.0	0.0	32.7
Day12:	99.9	6.9	0.0	32.7
Day13:	99.7	12.5	0.0	32.7
Day14:	99.9	31.9	0.0	32.6
Day15:	99.7	20.5	14.6	32.4
Day16:	99.9	48.6	10.4	29.8
Day17:	99.7	0.0	62.8	28.0
Day18:	99.8	37.5	46.2	23.4
Day19:	99.8	43.4	0.0	23.2
Day20:	100.0	12.5	44.4	22.8
Day21:	75.8	0.0	53.8	17.8
Day22:	27.1	0.0	0.0	18.3
Day23:	17.6	0.0	0.0	18.7
Day24:	0.0	15.3	0.0	19.2
Day25:	0.0	0.0	26.4	17.9
Day26:	0.0	20.1	0.0	16.5
Day27:	0.0	12.5	17.0	16.3
Day28:	0.0	0.0	17.0	13.5
Day29:	0.0	0.0	0.0	13.4

C.3 Simulation results for Scenario 3

C.3.1 Statistical results for locations usage

Table C.5 depicts the usage percentages of each location per day for the simulation period of 33 days.

C.3.2 Statistical results for crane moves at each location

Table C.6 shows the number of crane movements at each location per day for the simulation period of 33 days.

C.4 Simulation results for Scenario 4

C.4.1 Statistical results for locations usage

Table C.7 depicts the usage percentages of each location per day for the simulation period of 35 days.

Table C.4: Crane moves at each location for Scenario 2

	Coil entry	Annealing	Cold-rolling input	Cold-rolling output	BWG
Day1:	26	24	0	0	0
Day2:	20	24	0	0	0
Day3:	19	8	0	0	0
Day4:	28	24	0	0	0
Day5:	24	8	0	0	0
Day6:	28	32	0	0	0
Day7:	19	0	0	0	0
Day8:	4	16	0	0	0
Day9:	0	16	0	0	0
Day10:	0	16	0	0	0
Day11:	0	16	0	0	0
Day12:	0	8	20	20	0
Day13:	0	24	20	20	0
Day14:	0	8	40	40	0
Day15:	0	24	20	20	8
Day16:	0	8	40	40	4
Day17:	0	24	0	0	21
Day18:	0	16	60	60	9
Day19:	0	12	47	45	0
Day20:	0	0	13	15	20
Day21:	0	12	0	0	21
Day22:	0	4	0	0	0
Day23:	0	4	0	0	0
Day24:	0	0	20	20	0
Day25:	0	0	0	0	12
Day26:	0	0	26	24	0
Day27:	0	0	14	16	7
Day28:	0	0	0	0	7
Day29:	0	0	0	0	0
Total:	168	328	320	320	109

C.4.2 Statistical results for crane moves at each location

Table C.8 shows the number of crane movements at each location per day for the simulation period of 35 days.

Table C.5: Percentage usage (%) in each location for Scenario 3

	Annealing	Cold-rolling	BWG	High-Bay
Day1:	89.9	0.0	0.0	1.1
Day2:	99.7	0.0	0.0	5.3
Day3:	100.0	0.0	0.0	9.7
Day4:	99.7	0.0	0.0	15.4
Day5:	100.0	0.0	0.0	21.2
Day6:	99.7	0.0	0.0	27.2
Day7:	100.0	0.0	0.0	32.2
Day8:	99.7	0.0	0.0	33.6
Day9:	100.0	0.0	0.0	33.6
Day10:	99.8	0.0	0.0	33.6
Day11:	99.9	0.0	0.0	33.6
Day12:	99.8	0.0	0.0	33.6
Day13:	99.9	0.0	0.0	33.6
Day14:	99.8	6.9	0.0	33.6
Day15:	99.9	13.2	0.0	33.6
Day16:	100.0	31.3	0.0	33.5
Day17:	99.7	20.5	25.0	32.6
Day18:	99.9	43.4	0.0	30.7
Day19:	99.8	5.2	46.2	30.2
Day20:	99.9	27.8	62.8	25.5
Day21:	99.8	6.9	0.0	24.1
Day22:	99.9	43.1	3.8	23.8
Day23:	99.8	15.3	83.7	19.3
Day24:	99.9	0.0	18.1	15.5
Day25:	99.8	0.0	0.0	14.1
Day26:	100.0	0.0	0.0	14.1
Day27:	99.8	0.0	0.0	15.3
Day28:	69.1	2.8	0.0	16.7
Day29:	51.2	12.5	0.0	17.2
Day30:	0.2	0.0	18.1	17.1
Day31:	0.0	30.6	0.0	16.6
Day32:	0.0	0.0	19.4	16.2
Day33:	0.0	0.0	0.0	15.0

Table C.6: Crane moves at each location for Scenario 3

	Coil entry	Annealing	Cold-rolling input	Cold-rolling output	BWG
Day1:	26	20	0	0	0
Day2:	20	24	0	0	0
Day3:	19	0	0	0	0
Day4:	28	24	0	0	0
Day5:	24	0	0	0	0
Day6:	28	24	0	0	0
Day7:	19	0	0	0	0
Day8:	4	24	0	0	0
Day9:	0	0	0	0	0
Day10:	0	16	0	0	0
Day11:	0	8	0	0	0
Day12:	0	16	0	0	0
Day13:	0	8	0	0	0
Day14:	0	16	20	20	0
Day15:	0	8	22	20	0
Day16:	0	0	38	40	0
Day17:	0	24	20	20	12
Day18:	0	8	33	32	0
Day19:	0	16	7	8	12
Day20:	0	8	40	40	18
Day21:	0	16	9	7	0
Day22:	0	8	51	53	3
Day23:	0	16	20	20	33
Day24:	0	8	0	0	8
Day25:	0	16	0	0	0
Day26:	0	0	0	0	0
Day27:	0	8	0	0	0
Day28:	0	4	5	3	0
Day29:	0	4	15	17	0
Day30:	0	4	0	0	8
Day31:	0	0	40	40	0
Day32:	0	0	0	0	8
Day33:	0	0	0	0	0
Total:	168	328	320	320	102

Table C.7: Percentage usage (%) in each location for Scenario 4

	Annealing	Cold-rolling	BWG	High-Bay
Day1:	89.9	0.0	0.0	1.1
Day2:	99.7	0.0	0.0	1.5
Day3:	100.0	0.0	0.0	8.0
Day4:	99.8	0.0	0.0	10.2
Day5:	99.9	0.0	0.0	17.9
Day6:	99.8	0.0	0.0	22.1
Day7:	99.9	0.0	0.0	29.3
Day8:	99.7	0.0	0.0	32.7
Day9:	100.0	0.0	0.0	33.6
Day10:	99.8	0.0	0.0	33.6
Day11:	99.9	0.0	0.0	33.6
Day12:	99.8	0.0	0.0	33.6
Day13:	99.9	0.0	0.0	33.6
Day14:	99.8	0.0	0.0	33.6
Day15:	99.9	6.9	0.0	33.6
Day16:	100.0	12.5	0.0	33.6
Day17:	99.7	22.2	0.0	33.5
Day18:	99.8	30.2	2.1	33.5
Day19:	99.9	8.3	22.9	31.1
Day20:	99.9	37.5	4.9	30.7
Day21:	99.8	16.0	92.0	27.9
Day22:	99.9	25.7	12.2	24.4
Day23:	99.8	47.6	0.0	23.8
Day24:	99.9	9.4	52.4	22.4
Day25:	99.8	0.0	62.8	15.2
Day26:	100.0	0.0	0.0	13.2
Day27:	99.8	0.0	0.0	14.5
Day28:	72.8	0.0	0.0	15.7
Day29:	47.7	0.0	0.0	16.4
Day30:	0.0	0.0	0.0	17.7
Day31:	0.0	0.0	0.0	17.7
Day32:	0.0	3.1	0.0	17.7
Day33:	0.0	27.4	0.0	17.6
Day34:	0.0	0.0	19.4	16.1
Day35:	0.0	0.0	0.0	15.9

Table C.8: Crane moves at each location for Scenario 4

	Coil entry	Annealing	Cold-rolling input	Cold-rolling output	BWG
Day1:	26	20	0	0	0
Day2:	0	16	0	0	0
Day3:	39	8	0	0	0
Day4:	0	16	0	0	0
Day5:	52	8	0	0	0
Day6:	0	16	0	0	0
Day7:	47	8	0	0	0
Day8:	0	24	0	0	0
Day9:	4	0	0	0	0
Day10:	0	16	0	0	0
Day11:	0	8	0	0	0
Day12:	0	16	0	0	0
Day13:	0	8	0	0	0
Day14:	0	16	0	0	0
Day15:	0	8	20	20	0
Day16:	0	0	20	20	0
Day17:	0	24	32	30	0
Day18:	0	16	28	30	2
Day19:	0	8	7	5	10
Day20:	0	8	33	35	2
Day21:	0	16	24	22	24
Day22:	0	8	36	38	4
Day23:	0	16	52	50	0
Day24:	0	8	8	10	23
Day25:	0	16	0	0	25
Day26:	0	0	0	0	0
Day27:	0	8	0	0	0
Day28:	0	4	0	0	0
Day29:	0	8	0	0	0
Day30:	0	0	0	0	0
Day31:	0	0	0	0	0
Day32:	0	0	5	3	0
Day33:	0	0	35	37	0
Day34:	0	0	0	0	8
Day35:	0	0	0	0	0
Total:	168	328	300	300	98

Appendix D

MATLAB programme code for chapter 6

This Appendix provides all the code lines developed in MATLAB environment for the modelling and simulation of Bridgnorth Aluminium production process studied in chapter 6.

```
input_from_hotline = deliver_process(run_index_local,no_coils_loaded,state_buff,param,input_file);
input_from_hotline = check_empty(input_from_hotline);
no_coils_temp=size(input_from_hotline,1);
no_coils_loaded=no_coils_loaded+no_coils_temp;
state_buff=update_buff(state_buff,input_from_hotline,output_to_highbay);
%-----------------------
'[state_highbay,out_to_buff1_n,out_to_in_cold_roll_buff,out_to_in_BWG_buff,in_from_buff,in_from_anneal_buff,...
in_from_out_cold_roll_buff,state_out_anneal_buff,state_out_cold_roll_buff,index_list_coils_cr,...
real_crane_moves]=...
update_highbay_new(state_highbay,run_index,state_buff1_n,state_in_cold_roll_buff,state_in_BWG_buff,...
state_out_anneal_buff,state_out_cold_roll_buff,index_list_coils_cr,param);
tot_real_crane_moves=sum(real_crane_moves);
%-----------------------
'[state_in_BWG_buff,output_to_BWG]=update_in_BWG_buff(state_in_BWG_buff,out_to_in_BWG_buff,BWG_flag_empty);
'[state_BWG,time_BWG,out_to_out_buff]=update_bwg(state_BWG,time_BWG,output_to_BWG);
state_out_buff=update_output_buff(state_out_buff,out_to_out_buff);
%-----------------------
'[state_buff1_n,output_to_anneal]=update_buff1_n(state_buff1_n,out_to_buff1_n,anneal_flag_empty);
'[state_anneal,index_state_anneal,time_anneal,output_from_anneal]= ...
update_anneal1_n(state_anneal,index_state_anneal,time_anneal,output_to_anneal,state_out_anneal_buff,param);
state_out_anneal_buff=update_out_anneal_buff(state_out_anneal_buff,output_from_anneal,output_to_highbay);
%-----------------------
'[state_in_cold_roll_buff,output_to_cold_roll]= ...
update_in_cold_roll_buff(state_in_cold_roll_buff,out_to_in_cold_roll_buff,cold_roll_flag_empty);
'[state_cold_roll,time_cold_roll,output_from_cold_roll]= ...
update_cold_roll(state_cold_roll,time_cold_roll,output_to_cold_roll,state_out_cold_roll_buff,param);
state_out_cold_roll_buff=update_out_cold_roll_buff(state_out_cold_roll_buff,output_from_cold_roll,output_to_highbay);
```

Next a brief description of each of these functions is given together with their inputs and outputs.

- Function `deliver_process`: Defines the coils which have completed the Hot-Line process and can be entered into the High-Bay by the crane. These coils initially update the input buffer and await to be allocated to the High-Bay storage area. The function `deliver_process` according to the state of buffer (`state_buff`), loading rate of coils in the entry of High-Bay and the capacity of input buffer - specified in parameters `param` - checks initially if the buffer

capacity has been reached. If this is not the case, the function reads the excel input file `input_file` and takes the first coil on the list that is to be entered. Due to scheduling restrictions of the number of coils dispatched to the High-Bay entry each day, `deliver_process` checks with the aid of `no_coils_loaded` whether the number of coils loaded in a specific day has reached the predefined number set by the user in the spreadsheet file (.xls). Table D.1 gives a brief description of inputs and outputs for `deliver_process`.

Table D.1: Inputs and outputs of function *deliver_process*

`input_from_hotline=deliver_process(run_index_local,no_coils_loaded,state_buff,param,input_file)`
Inputs:
`run_index_local`: Specifies current run index
`no_coils_loaded`: The number of coils loaded at the beginning of time step
`state_buff`: The number of coils in buffer before entering the High-Bay
`param`: The structure of parameters
`input_file`: The name of the spreadsheet input file
Outputs:
`input_from_hotline`: The coils loaded to the input buffer before High-Bay

- Function `update_buff`: This function updates the state of the buffer which is located between the Hot-Line and the High-Bay in Litho centre and can be called either with `input_from_hotline=[]` or `output_to_highbay=[]` or even from both []. The function `update_buff` does not define its output, which is "pulled in" by the High-Bay. It is the responsibility of the High-Bay (through function `update_highbay` defined later) to check that requested row-size of `output_to_highbay` does not exceed buffer's state-dimension. Similarly, it is the responsibility of the Hot-Line (via function `update_buff`) to ensure it does not load the buffer when the later is at full capacity. Table D.2 summarises the properties of the function `update_buff`.

Table D.2: Inputs and outputs of function *update_buff*

`state_buff=update_buff(state_buff,input_from_hotline,output_to_highbay)`
Inputs:
`state_buff`: List of coils at the beginning of current time-step
`input_from_hotline`: Empty matrix when there is no coils from HotLine or list of coils
`output_to_highbay` or []: Empty matrix (no input requested from High-Bay) or list of coils
Outputs:
`state_buff`: List of coils at the end of current time-step

- Function `update_highbay_new` updates the High-Bay state and outputs. This is the most complex function of the software tool since it updates the state of

the High-Bay according to its complex interrelations with the other processes. The function operates as follows. Firstly, it checks the number of coils inside the High-Bay storage area at the beginning of each time interval and secondly, whether a list of coils for cold-rolling (initially 20 or 40) has been identified in an earlier step. Next, it is determined which of the cold-rolled coils have cooled (so they can be moved to `out_to_in_BWG_buff` and then proceed to BWG) and also, which of the coils that they have been annealed have been also cooled (so they can move to `out_to_in_cold_roll_buff`), and which coils need to be loaded to annealing machine through `out_to_buff1_n`.

Note that till this stage the function specifies the coil which could potentially be moved to the input cold rolling buffer, and its index. Also, in case when an empty matched list is identified, new list of matched coils for cold rolling is created (possibly empty) indexed with respect to input argument `state_highbay`. If there is a non-empty matched list, then the output list either stays the same (if no coil is moved to input cold rolling buffer) or its first row is deleted.

Finally, the function determines whether there are coils to be moved to/from input and/or output buffers of the annealing machine. The function checks capacities of input buffer of High-bay. If at least one quartet of coils can be moved, they are loaded in the buffer (after clearing their time stamp) and the state of output buffer of annealing machine is updated by constructing a new state and a new-list. Else the function performs in/out loading operations in sequence according of to priorities list specified by user in GUI (subject to capacity constraints), until all permissible crane movements have been exhausted or all tasks have been performed. The syntax of `update_highbay_new` function together with its inputs and outputs is given in Table D.3.

- Function `update_in_BWG_buff` updates the state and output of the buffer

Table D.3: Inputs and outputs of function *update_highbay_new*

[state_highbay,out_to_buff1_n,out_to_in_cold_roll_buff,out_to_in_BWG_buff,in_from_buff,in_from_anneal_buff,... in_from_out_cold_roll_buff,state_out_anneal_buff,state_out_cold_roll_buff,index_list_coils_cr,... real_crane_moves]=...update_highbay_new(state_highbay,run_index,state_buff1_n,state_in_cold_roll_buff,state_in_BWG_buff,... state_out_anneal_buff,state_out_cold_roll_buff,index_list_coils_cr,param)
Inputs:
state_highbay: List of coils being stored in High-Bay or empty matrix. These coils are listed vertically
run_index: Current iteration index (integer)
state_buff1_n: List of coils intended for annealing at the beginning of time step
state_in_cold_roll_buff: List of coils intended for cold rolling at the beginning of time step
state_in_BWG_buff: List of coils intended for BWG at the beginning of time step
state_out_anneal_buff: List of coils that have finished annealing and waiting to return into the High-Bay at the beginning of time step. Note that this state is updated internally
state_out_cold_roll_buff: List of coils that have finished cold rolling and waiting to return into the High-Bay at the beginning of time step. Note that this state is updated internally
index_list_coils_cr: Index of coils in identified matched list in state_highbay
param: The structure of parameters
Outputs:
state_highbay: List of coils being stored inside High-Bay at the end of time-interval (listed vertically)
out_to_buff1_n: List of matching coil quartets loaded to buff1_n (input side of anneal_n)
out_to_in_cold_roll_buff: List of coils loaded to input cold rolling buffer
out_to_in_BWG_buff: List of coils loaded to input BWG buffer
in_from_buff: List of coils loaded into High-Bay from the buffer after Hotline
in_from_anneal_buff: List of coils loaded from annealing buffer (outside of annealing machine)
in_from_out_cold_roll_buff: List of coils loaded to High-Bay from cold rolling output buffer
state_out_anneal_buff: List of coils that have finished annealing and waiting to return into the High-Bay at the end of time step. Note that this state is updated internally
state_out_cold_roll_buff: List of coils that have finished cold rolling and waiting to return into the High-Bay at the end of time step. Note that this state is updated internally
index_list_coils_cr: Index of coils in identified matched list in state_highbay
real_crane_moves: Real number of crane movements

which is located between High-Bay and BWG. This function checks first if there is any coil currently processed to BWG. If there is no such coil then function checks whether any coil is currently waiting to be loaded to BWG (coils that have finished cold-rolling process and they have waited enough for cooling purposes). In case that many coils are found in High-Bay that can be transferred to BWG, the function selects the coil which is is on top of the list. Table D.4 summarises the properties of the function `update_in_BWG_buff`.

Table D.4: Inputs and outputs of function *update_in_BWG_buff*

[state_in_BWG_buff,output_to_BWG]=update_in_BWG_buff(state_in_BWG_buff,out_to_in_BWG_buff,BWG_flag_empty)
Inputs:
state_in_BWG_buff: List of coils in buffer at current time step
out_to_in_BWG_buff: Empty matrix if there is no output from High-Bay or list of coils
BWG_flag_empty: Flag indicates whether BWG is busy or not. (1: BWG machine empty, 0: BWG machine busy)
Outputs:
state_in_BWG_buff: list of coils after transition (end of current time step)
output_to_BWG: Empty matrix if there is no transfer from BWG, or else a coil to be transferred to BWG.

- Function `update_bwg` updates the state and output of BWG process. If BWG is empty the function loads a coil from its input buffer `output_to_BWG` else it updates its state internally for the BWG work cycle (30 minutes). When the coil completes the BWG process it is loaded to the output buffer `out_to_out_buff`. Table D.5 summarises the properties of the function `update_bwg`.

Table D.5: Inputs and outputs of function *update_bwg*

[state_BWG,time_BWG,out_to_out_buff]=update_bwg(state_BWG,time_BWG,output_to_BWG)
Inputs:
state_BWG: Coil being processed in BWG or an empty matrix at the beginning of time step.
time_BWG: Empty matrix if BWG empty; If BWG is busy, it gives the time-step since the beginning of BWG process.
output_to_BWG: List of coils in the buffer located between High-Bay and BWG.
Outputs:
state_BWG: Coil inside BWG machine at the end of time-interval.
time_BWG: Updates the BWG cycle time. If BWG is loaded from empty the time is reset to 1. If BWG is busy then the time clock increases by one. In case where BWG process is just finished, time_BWG becomes an empty matrix.
out_to_out_buff: Output buffer. It is empty if BWG process still in progress, or BWG not busy, else it loads a coil that has passed the BWG process.

- Function `update_output_buff` updates the output buffer after the BWG process. Note that this buffer collects all coils (arranged in a list) that have completed the whole process and thus it can be deemed as the output buffer of the simulation model. Table D.6 shows the inputs and output of the function `update_output_buff`.

Table D.6: Inputs and outputs of function *update_output_buff*

state_out_buff=update_output_buff(state_out_buff,out_to_out_buff)
Inputs:
state_out_buff: List of coils in output buffer at the beginning of current time step.
out_to_out_buff: Coil that has just finished BWG process.
Outputs:
state_out_buff: List of coils in output buffer at the end of current time-interval.

- Function `update_buff1_n` updates the state of the buffer which is located between High-Bay and annealing machines. This function checks first with the aid of input `out_to_buff1_n` if there are coils than can be loaded into one or more annealing machines. The input `anneal_flag_empty` indicates whether there are one or more empty annealing machines to load the coils from `out_to_buff1_n`. The summary of the function `update_buff1_n` is given in Table D.7.

Table D.7: Inputs and outputs of function *update_buff1_n*

[state_buff1_n,output_to_anneal]=update_buff1_n(state_buff1_n,out_to_buff1_n,anneal_flag_empty)
Inputs:
state_buff1_n: List of coils in buffer at current time step.
out_to_buff1_n: Empty matrix if there is no output from High-Bay or list of coils.
anneal_flag_empty: A n-dimension row vector where n is the number of annealing machines. If anneal_flag_empty(i)=1, it means that the i-th annealing machine is empty and if anneal_flag_empty(i)=0 the i-th annealing machine is busy.
Outputs:
state_buff1_n: List of coils after transition (end of current time step).
output_to_anneal: Empty matrix if there is no transfer to annealing machine, else a list of coils to be transferred.

- Function `update_anneal1_n` updates the state of annealing machines. The number of annealing machines is specified by the user in GUI, and it is passed

to the function through the input `param`. The function `update_anneal1_n` operates as follows. It checks if there is any annealing machine free and loads a quartet of coils from input `output_to_anneal` into it. It also updates the state of the annealing machines `state_anneal` and loads the coils that have completed the annealing process to the output `output_from_anneal` according to the state of the output annealing buffer `state_out_anneal_buff`. Table D.8 summarises the inputs and outputs of the function `update_anneal1_n`.

Table D.8: Inputs and outputs of function *update_anneal1_n*

`[state_anneal,index_state_anneal,time_anneal,output_from_anneal]=...` `update_anneal1_n(state_anneal,index_state_anneal,time_anneal,output_to_anneal,state_out_anneal_buff,param)`
Inputs:
`state_anneal`: List of coils being annealed or empty matrix. Coils are listed vertically and the state dimension of the i-th annealing machine is specified by the input `index_state_anneal(i)`.
`index_state_anneal`: Number of coils in i-th annealing machine for input `state_anneal`.
`time_anneal`: A n-vector (where n is the number of parallel annealing machines) whose i-th entry specifies the time step since beginning of annealing in i-th machine in case i-th machine is busy. If i-th machine is empty then `time_anneal(i)=-1`.
`output_to_anneal`: List of coils from the buffer located between High-Bay and annealing machines.
`state_out_anneal_buff`: Current state of the output buffer of annealing machines. Used to calculate the output so that no overflow occurs. `param`: Parameters structure
Outputs:
`state_anneal`: List of coils inside annealing machines at the end of time interval; Coils are listed vertically and the state dimension of the i-th annealing machine is specified by the output `index_state_anneal(i)`.
`index_state_anneal`: Number of coils in i-th annealing machine for output `state_anneal`.
`time_anneal`: A n-dimensional vector. If the i-th machine is busy then `time_anneal(i)=time_anneal(i)+1`. If the annealing process has finished and there is no loaded output (for not causing overflow to `state_out_anneal_buff`), `time_anneal(i)=time_anneal(i)` (time is frozen).When a quartet of coils is loaded to the i-th, then `time_anneal(i)` reset to 1. If the i-th annealing machine is empty at the end of interval then `time_anneal(i)=-1`.
`output_from_anneal`: List of coils contribution from the i-th annealing machine is an empty matrix if annealing is still in progress or annealing in i-th machine has finished but output will cause overflow to `state_out_anneal_buff`. Else output is stacked to the `output_from_anneal` array in machine order.

- Function `update_out_anneal_buff` updates the state of the buffer which is located between annealing machines and High-Bay. Function should be called either with `output_from_anneal=[]` or `output_to_highbay=[]` (or both []). Note that this buffer does not define its output, which is "pulled in" by the High-Bay. It is the responsibility of the High-Bay (through the function `update_highbay`) to check that requested row-size of `output_to_highbay` does not exceed buffer's state dimension. Similarly, it is the responsibility of the annealing machine (via `update_anneal_n`) to ensure that row-size of `output_from_anneal` pushed into the buffer will not cause overflow (i.e., buffer's state dimension + `output_from_anneal` <= buffer's capacity). The inputs and output of the function `update_out_anneal_buff` are given in Table D.9.

- Function `update_in_cold_roll_buff` updates the state and output in the

Table D.9: Inputs and outputs of function *update_out_anneal_buff*

state_out_anneal_buff=update_out_anneal_buff(state_out_anneal_buff,output_from_anneal,output_to_highbay)
Inputs:
state_out_anneal_buff: List of coils at current time step.
output_from_anneal: Empty matrix if there is no output from annealing machines or list of coils.
output_to_highbay: Empty matrix if there is no input requested from High-Bay or list of coils.
Outputs:
state_out_anneal_buff: List of coils in buffer after transition (end of current time step).

buffer which is located between High-Bay and cold-rolling machine. This function checks first if there is any coil from High-Bay waiting to be loaded into the cold rolling machine via input `out_to_in_cold_roll_buff`. The input `cold_roll_flag_empty` indicates whether cold rolling machine is empty or not. The summary of the function `update_in_cold_roll_buff` is given in Table D.10

Table D.10: Inputs and outputs of function *update_in_cold_roll_buff*

[state_in_cold_roll_buff,output_to_cold_roll]= ... update_in_cold_roll_buff(state_in_cold_roll_buff,out_to_in_cold_roll_buff,cold_roll_flag_empty)
Inputs:
state_in_cold_roll_buff: List of coils in buffer at current time step.
out_to_in_cold_roll_buff: Empty matrix if there is no output from High-Bay or list of coils.
cold_roll_flag_empty: An integer indicates if cold rolling machine is busy or not (1:Cold rolling machine empty; 0:Cold-rolling machine busy) .
Outputs:
state_in_cold_roll_buff: List of coils after transition (end of current time step).
output_to_cold_roll: Empty matrix if there is no transfer to cold rolling machine, else a list of coils to be transferred.

- Function `update_cold_roll` updates the state of cold rolling machine. This function operates as follows. It checks if the cold rolling machine is free and loads a coil from input `output_to_cold_roll` into it. It also updates the state of the cold rolling `state_cold_roll` and loads the coil that has completed the cold rolling process to the output `output_from_cold_roll` according to the state of the output cold rolling buffer `state_out_cold_roll_buff`. Table D.11 summarises the inputs and outputs of the function `update_cold_roll`.

- Function `update_out_cold_roll_buff` updates the state of the buffer which is located between cold rolling machine and High-Bay. This function should be called either with `output_from_cold_roll=[]` or `output_to_highbay=[]` or both []. Note that this buffer does not define its output, which is "pulled in" by the High-Bay. It is the responsibility of the High-Bay (through function `update_highbay`) to check that requested row-size of `output_to_highbay` does

Table D.11: Inputs and outputs of function *update_cold_roll*

[state_cold_roll,time_cold_roll,output_from_cold_roll]= ... update_cold_roll(state_cold_roll,time_cold_roll,output_to_cold_roll,state_out_cold_roll_buff,param))
Inputs:
state_cold_roll: List of coils being cold-rolled or empty matrix.
time_cold_roll: Specifies time since the last start of cold-rolling process.
output_to_cold_roll: Coil from the buffer located between High-Bay and cold rolling machine or empty matrix.
state_out_cold_roll_buff: Current state of the output buffer of the cold rolling machine. It is used to calculate the output so that no overflow occurs.
param: Parameters structure.
Outputs:
state_cold_roll: Returns the current coil which is being cold-rolled at the end of time interval or empty matrix.
time_cold_roll: Time since last cold rolling begun at the end of current time step. If machine is busy then time_cold_roll=time_cold_roll+1. If cold rolling process is completed and there is no loaded output, time_cold_roll=time_cold_roll (time is frozen). When a coil is loaded,then time_cold_roll reset to 1. If the machine is empty at the end of interval then time_cold_roll=-1.
output_from_cold_roll: Cold-rolled coil loaded to output buffer state_out_cold_roll_buff or empty matrix.

not exceed buffer's state-dimension. Similarly, it is the responsibility of the cold-rolling machine (via `update_cold_roll` function) to ensure that row-size of `output_from_cold_roll` pushed into the buffer will not cause overflow (i.e., that buffer's state-dimension + `output_from_cold_roll` <= buffer's capacity).

The inputs and output of the function `update_out_cold_roll_buff` are given in Table D.12.

Table D.12: Inputs and outputs of function *update_out_cold_roll_buff*

state_out_cold_roll_buff=update_out_cold_roll_buff(state_out_cold_roll_buff,output_from_cold_roll,output_to_highbay)
Inputs:
state_out_cold_roll_buff: List of coils at start of current time step.
output_from_cold_roll: Empty matrix if there is no output from cold rolling machine or list of coils.
output_to_highbay: Empty matrix if there is no input requested from High-Bay or list of coils.
Outputs:
state_out_cold_roll_buff: List of coils in buffer after transition (end of current time step).

The following code lines are consist of M-files and functions which are presented in alphabetical order.

▶ Function anneal_stamp:

```
  function slabs=anneal_stamp(slabs)
% function slabs=anneal_stamp(slabs)
% Update anneal stamp on input array of slabs
%
  no_slabs=size(slabs,1); % no slabs
  if no_slabs == 0 | size(slabs,2) == 0
     slabs=[];
     return
  end
%
  for i=1:no_slabs
      current_slab=slabs(i); % get current slab
      flag_current_slab=current_slab.flags; % get flags field
      flag_current_slab(2)=flag_current_slab(2)+1; % increment by one each time slab is annealed
      current_slab = setfield(current_slab,'flags',flag_current_slab);
      slabs(i)=current_slab; % substitute back
  end
%
%--------------- end of anneal_stamp.m ------------------------------------------------------
```

▶ Function check_empty:

```
  function out_list=check_empty(in_list)
% function out_list=check_empty(in_list)
% checks if input list is empty

if ((size(in_list,1)==0) | (size(in_list,2)==0))
   out_list=[];
else
   out_list=in_list;
end
%
%---------------------- end of check_empty.m ----------
```

▶ M-file check_out_variables:

```
% check_out_variables
% called from update_highbay_new.m

if exist('out_to_in_cold_roll_buff') ~= 1
   out_to_in_cold_roll_buff=[];
end
%
if exist('in_from_out_cold_roll_buff') ~= 1
   in_from_out_cold_roll_buff=[];
end
%
if exist('out_to_in_BWG_buff') ~= 1
   out_to_in_BWG_buff=[];
end
%
if exist('in_from_buff') ~= 1
   in_from_buff=[];
end
%
%--------- end of check_out_variables.m ---------
```

▶ Function cold_roll_stamp:

```
  function out_slabs=cold_roll_stamp(in_slabs)
% function out_slabs=cold_roll_stamp(in_slabs)
% Update cold-rolling stamp on input array of slabs
%
  in_slabs=check_empty(in_slabs);
  if isempty(in_slabs)
     out_slabs=[];
     return
  end
%
  no_slabs=size(in_slabs,1);
  out_slabs=in_slabs;
  for i=1:no_slabs
      current_slab=in_slabs(i,:); % get current slab
      flag_current_slab=current_slab.flags; % get flags field
      flag_current_slab(3)=flag_current_slab(3)+1; % increment flag after cold-roll process
      current_slab = setfield(current_slab,'flags',flag_current_slab);
      out_slabs(i,:)=current_slab; % substitute back
  end
%
%------------------------------ end of cold_roll_stamp.m ----------------------------------
```

▶ Function default_parameters:

```
function param=default_parameters
% function param=default_parameters
% Specifies parameters for the process - can be called from anywhere in the code

%-------------------------------------------------------------------------------
% SIMULATION PARAMETERS:
%
simulation_step=5; % minutes
simulation_time=168; % hours
%-------------------------------------------------------------------------------
% NO MACHINES PARAMETERS:
%
no_anneal_machines=3; % no of parallel annealing machines
%-------------------------------------------------------------------------------
% CAPACITIES:
%
high_bay_capacity=440; %440; % no of racks
buff_capacity=20; % between hot-line and high-bay
buff1_n_capacity=8; % 2 racks of 4 quartet-slabs each
in_BWG_buff_capacity=1; % capacity of BWG input buffer
capacity_BWG_buff=1; % capacity of BWG buffer
out_anneal_buff_capacity=4; % out-side of anneal machines
in_cold_roll_buff_capacity=1; % any two slabs
out_cold_roll_buff_capacity=1; % any two slabs
%-------------------------------------------------------------------------------------------
% COOLING TIME PARAMETERS (in HighBay):
%
cool_time_after_anneal=12; %72*12; = 72 hours, no of 5-minute periods
cool_time_after_coldroll=288; % 24*12; = 24 hours, no of 5-minute periods
%-------------------------------------------------------------------------------------------
%
crane_moves=4; % max no of'crane moves allowable during simulation step
priorities=[1 2 3]; % [1 2 3] priorities order= cold rolling(in-out), out to BWG, in to HB from hotline
input_file='schedule1'; % input data file
```

```
load_rate=2; % coils loaded to input buffer every load_rate time steps
plot_flag=0; % 0=No plots, 1=Complete process, 2=Highbay
stats_plot_flag=1; % 0=No plots, 1=Plots
pause_time=0.1; % Pause-time for plots
out_file_flag=1; % 1=Save output stats file, 0=No output file
out_file_name='results'; % output file name
initial_conditions_flag=0; % 1=Use Initial Data, 0=No Initial Conditions
initial_conditions_file='initial_data'; % initial conditions file (mat file)

% ASSEMBLE OVERALL param STRUCTURE
%
param= struct('simulation_time',simulation_time,...
       'simulation_step',simulation_step, ...
       'no_anneal_machines',no_anneal_machines, ...
       'high_bay_capacity',high_bay_capacity,...
       'buff_capacity',buff_capacity, ...
       'buff1_n_capacity',buff1_n_capacity,...
       'in_BWG_buff_capacity',in_BWG_buff_capacity,...
       'capacity_BWG_buff',capacity_BWG_buff,...
       'out_anneal_buff_capacity',out_anneal_buff_capacity, ...
       'in_cold_roll_buff_capacity',in_cold_roll_buff_capacity,...
       'out_cold_roll_buff_capacity',out_cold_roll_buff_capacity, ...
       'cool_time_after_anneal',cool_time_after_anneal,...
       'cool_time_after_coldroll',cool_time_after_coldroll, ...
       'crane_moves',crane_moves,...
       'priorities',priorities,...
       'input_file',input_file, ...
       'load_rate',load_rate, ...
       'plot_flag',plot_flag, ...
       'pause_time',pause_time, ...
       'stats_plot_flag',stats_plot_flag, ...
       'out_file_flag',out_file_flag, ...
       'out_file_name',out_file_name, ...
       'initial_conditions_flag',initial_conditions_flag, ...
       'initial_conditions_file',initial_conditions_file);
%
% ---------------------------------------- end of default_parameters.m ----------------------------------
```

▶ Function deliver_process:

```
  function out_from_dp = deliver_process(run_index,no_coils_loaded,state_buff,param,input_file)
% function out_from_dp = deliver_process(run_index,no_coils_loaded,state_buff,param,input_file)
%
% Defines coils entering input buffer (before High-Bay)

% INPUTS
% run_index: current run index
% no_coils_loaded: no coils loaded at beginning of time step
% state_buff: number of coils in buffer (before HighBay)
% param: parameter structure

% OUPUTS
% out_from_dp: coils loaded to buffer before High-Bay

%-------------------------------------------------------------------------------------------------------
% Get parameters
%
simulation_time=param.simulation_time;
simulation_step=param.simulation_step;
buff_capacity=param.buff_capacity;
%input_file=param.input_file;
load_rate=param.load_rate; % load loaded to input every load_rate steps
%-------------------------------------------------------------------------------------------------------
if load_rate == 1
   index1=1;
else
   index1=rem(run_index,load_rate); % is 1 when run_index=1,load_rate+1,2*load_rate+1, etc
end
%
state_buff=check_empty(state_buff); index2=size(state_buff,1);
current_day=floor((run_index*simulation_step)/(24*60))+1; % current day
[slabs_list,day_index]=read_from_xls_all(input_file);
slabs_list=check_empty(slabs_list);
%
if (size(slabs_list,1) >= no_coils_loaded+1)
   out_from_dp=slabs_list(no_coils_loaded+1,:);
else
   out_from_dp=[];
   return
end
%
if ((index1 ~= 1) | (index2 >= buff_capacity) |
(day_index(no_coils_loaded+1) > current_day))
   out_from_dp=[];
end
%
%------------------------------ end of deliver_process ----------------------------------
```

▶ Function determine_cold_roll_time:

```
function total_time_for_rolling=determine_cold_roll_time(coil)
% function total_time_for_rolling=determine_cold_roll_time(coil)
% Function determines cold-rolling time according to coil type
%
coil=check_empty(coil); if isempty(coil)
   total_time_for_rolling=[];
```

```
   return
end
%
coil=coil(1,:); top_slab_gauge=coil.gauge;
top_slab_width=coil.width; no_passes=coil.flags(3);

if all(top_slab_gauge=='thic')  | ( all(top_slab_gauge=='stan')  &
all(top_slab_width=='wide'))
   if no_passes==0
      total_time_for_rolling=2; %(10 minutes)
   elseif no_passes==1
       total_time_for_rolling=3; %(15 minutes)
   elseif no_passes==2
       total_time_for_rolling=4; %(20 minutes)
   else
       total_time_for_rolling=[]; %not-possible
   end

elseif all(top_slab_gauge=='stan')  &  (all(top_slab_width=='medi')
| all(top_slab_width=='narr'))
       if no_passes==0
          total_time_for_rolling=3; %(14.5 minutes)
       elseif no_passes==1
          total_time_for_rolling=4; %(17.5 minutes)
       elseif no_passes==2
          total_time_for_rolling=6; %(27.5 minutes we consider 30 minutes)
       else
          total_time_for_rolling=[]; %not-possible
       end

elseif all(top_slab_gauge=='thin')  &  all(top_slab_width=='wide')
       if no_passes==0
          total_time_for_rolling=2; %(10 minutes)
       elseif no_passes==1
          total_time_for_rolling=3; %(15 minutes)
       elseif no_passes==2
          total_time_for_rolling=4; %(20 minutes)
       elseif no_passes==3
          total_time_for_rolling=6; %(27.5 minutes we consider 30 minutes)
       else
          total_time_for_rolling=[]; %not-possible
       end

elseif all(top_slab_gauge=='thin')  &  (all(top_slab_width=='medi')
| all(top_slab_width=='narr'))
       if no_passes==0
          total_time_for_rolling=3; %(14.5 minutes)
       elseif no_passes==1
          total_time_for_rolling=4; %(17.5 minutes)
       elseif no_passes==2
          total_time_for_rolling=6; %(27.5 minutes we consider 30 minutes)
       else
          total_time_for_rolling=[]; %not-possible
       end
else
       total_time_for_rolling=[]; %not-possible
       disp('Error in determine_cold_roll_time.m ...')
end
%
%---------------------- end of determine_cold_roll_time.m ---------------------------------------
```

▶ Function determine_coil_status:

```
function
[flag1,flag_ready]=determine_coil_status(coil,time_in_highbay,param)
% function [flag1,flag_ready]=determine_coil_status(coil,time_in_highbay,param)
% Function determines whether coil that has been annealed has to be cold-rolled and if it has been
% cooled:
% flag1=1 if next stage is cold-rolling, flag1=0, next stage is BWG
% flag_ready=1, coil can exit HighBay, flag_ready=0, coil needs extra time to cool
%
coil=check_empty(coil); if isempty(coil)
   flag1=[];
   flag_ready=[];
   return
end
%
cool_time_after_anneal=param.cool_time_after_anneal;
cool_time_after_coldroll=param.cool_time_after_coldroll;
%
coil=coil(1,:); top_slab_gauge=coil.gauge;
top_slab_width=coil.width; no_passes=coil.flags(3);

if all(top_slab_gauge=='thic')  | ( all(top_slab_gauge=='stan')  &
all(top_slab_width=='wide'))
   %
   if no_passes==0 % annnealed, no cold-roll passes
   %
   flag1=1;
   if time_in_highbay < cool_time_after_anneal;
      flag_ready=0;
   else
      flag_ready=1;
   end
   %
elseif no_passes == 1 | no_passes == 2
    %
    flag1=1;
   if (time_in_highbay < cool_time_after_coldroll)
      flag_ready=0;
```

```
      else
         flag_ready=1;
      end
      %
   elseif no_passes == 3
       %
       flag1=0; % no more cold-rolling
       if (time_in_highbay < cool_time_after_coldroll)
         flag_ready=0;
      else
         flag_ready=1;
      end
      %
   else
      disp('Error in determine_cold_status.m ...');
      flag1=[];flag_ready=[];
      return
   end
%
elseif (all(top_slab_gauge=='stan')  &  (all(top_slab_width=='medi')
| all(top_slab_width=='narr')))
    %
    if no_passes==0 % annnealed, no cold-roll passes
      %
      flag1=1;
      if time_in_highbay < cool_time_after_anneal;
         flag_ready=0;
      else
         flag_ready=1;
      end
      %
  elseif no_passes == 1
      %
      flag1=1;
      if (time_in_highbay < cool_time_after_coldroll)
         flag_ready=0;
      else
         flag_ready=1;
      end
      %
  elseif no_passes == 2
       %
       flag1=0; % no more cold-rolling
       if (time_in_highbay < cool_time_after_coldroll)
         flag_ready=0;
       else
         flag_ready=1;
       end
       %
  else
      disp('Error in determine_cold_status.m ...');
      flag1=[];flag_ready=[];
      return
  end
%
elseif (all(top_slab_gauge=='thin')  &  all(top_slab_width=='wide'))

    if no_passes==0 % annnealed, no cold-roll passes
      %
      flag1=1;
      if time_in_highbay < cool_time_after_anneal;
         flag_ready=0;
      else
         flag_ready=1;
      end
      %
   elseif (no_passes == 1 | no_passes == 2 | no_passes==3)
      %
      flag1=1;
      if (time_in_highbay < cool_time_after_coldroll)
         flag_ready=0;
      else
         flag_ready=1;
      end
      %
   elseif no_passes == 4
       %
       flag1=0; % no more cold-rolling
       if (time_in_highbay < cool_time_after_coldroll)
         flag_ready=0;
       else
         flag_ready=1;
       end
       %
   else
      disp('Error in determine_cold_status.m ...');
      flag1=[];flag_ready=[];
      return
   end
%
elseif (all(top_slab_gauge=='thin')  &  (all(top_slab_width=='medi')
| all(top_slab_width=='narr')))

    if no_passes==0 % annnealed, no cold-roll passes
       flag1=1;
       %
       if (time_in_highbay < cool_time_after_anneal)
          flag_ready=0;
```

```
   else
      flag_ready=1;
   end
   %
elseif no_passes == 1 | no_passes == 2
   %
   flag1=1;
   if (time_in_highbay < cool_time_after_coldroll)
     flag_ready=0;
   else
     flag_ready=1;
   end
   %
elseif no_passes == 3
   %
   flag1=0; % no more cold-rolling
   if (time_in_highbay < cool_time_after_coldroll)
     flag_ready=0;
   else
     flag_ready=1;
   end
else
   disp('Error in determine_cold_status.m ...');
   flag1=[];flag_ready=[];
      return
   end
%
else
   disp('Error in determine_cold_status.m ...');
   flag1=[];
   flag_ready=[];
   return
end
%
%----------------------- end of determine_cold_status.m ----------------------------------------
```

▶ Function determine_slabs_annealed:

```
function
slabs=determine_slabs_annealed(slab_list,index_slab_list,index_machine)
% function slabs=determine_slabs_annealed(slab_list,index_slab_list,index_machine)
% Given n annealing machines in parallel loaded with slab_list, indexed by
% index_slab_list, function determines slabs at the (index-machine)-th
% machine.

  no_slabs=size(slab_list,1);
  if no_slabs==0 | size(slab_list,2)==0
     no_slabs=0;
     slabs=[];
     return
  end
  %
  if no_slabs ~= sum(index_slab_list) | index_machine ~= round(index_machine) | index_machine <=0
     slabs=[];
     disp('Error in determine_slabs_annealed.m ...')
     return
  end
  %
  if index_slab_list(index_machine) ~= 0
     if index_machine == 1
        slabs=slab_list(1:index_slab_list(1));
     else
        slabs=slab_list(sum(index_slab_list(1:index_machine-1))+1:sum(index_slab_list(1:index_machine-1))+index_slab_list(index_machine));
     end
  else
     slabs=[];
  end
  %
  if size(slabs,1)==0 | size(slabs,2)==0
     slabs=[];
  end
%
%------------------------------ end of determine_slabs_annealed.m -------------------------------
```

▶ Function determine_slabs_colour:

```
function groups=determine_slabs_colour(slabs_in)
% function groups=determine_slabs_colour(slabs_in)
% Function produces a list of 6 (distinct) colour codes corresponding to
% slabs_in
% Group 1: No slabs not annealed
% Group 2: No slabs annneal CR passes =0
% Group 3: CR passes = 1
% Group 4: CR passes = 2
% Group 5: CR passes = 3
% Group 6: CR passes = 4

slabs_in=check_empty(slabs_in); if isempty(slabs_in)
   groups=zeros(1,6);
   return
end
%
no_slabs_in=size(slabs_in,1); groups=zeros(1,6); for i=1:no_slabs_in
    current_slab=slabs_in(i,:);
    anneal_flag=current_slab.flags(2);
    cr_flag=current_slab.flags(3);
```

```
    if anneal_flag == 0
       groups(1)=groups(1)+1;
    elseif anneal_flag == 1
       if cr_flag == 0
          groups(2)=groups(2)+1;
       elseif cr_flag == 1
          groups(3)=groups(3)+1;
       elseif cr_flag == 2
          groups(4)=groups(4)+1;
       elseif cr_flag == 3
          groups(5)=groups(5)+1;
       elseif cr_flag == 4
          groups(6)=groups(6)+1;
       end
   end
end
%
% ---------------------- end determine_slabs_colour ------------------
```

▶ Function determine_slabs_colour1:

```
function
array_out=determine_slabs_colour1(slabs_list,run_index,param)
% function array_out=determine_slabs_colour1(slabs_list,run_index,param)
%
% Function groups HighBay slabs_list in following 11 categories:
%
%  1. Not yet annealed
%  2. Annealed and Hot
%  3. Annealed and Cold
%  4. Cold-rolled once and Hot
%  5. Cold-rolled once and Cold
%  6. Cold-rolled twice and Hot
%  7. Cold-rolled twice and Cold
%  8. Cold-rolled 3 times and Hot
%  9. Cold-rolled 3 times and Cold
% 10. Cold-rolled 4 times and Hot
% 11. Cold-rolled 4 times and Cold
%
% The total numbers of each category are recorded in the corresponding
% entry of array_out.

%-------------------------------------------------------------------------
% Decode parameters
cool_time_after_anneal=param.cool_time_after_anneal;
cool_time_after_coldroll=param.cool_time_after_coldroll;
%-----------------
slabs_list=check_empty(slabs_list); no_slabs=size(slabs_list,1);
array_out=zeros(6,2);
%
for i=1:no_slabs
      %
      current_slab=slabs_list(i,:);
      time_of_entry=current_slab.timer;
      current_slab_flags=current_slab.flags;
      current_slab_anneal_flag=current_slab_flags(2);
      current_slab_cold_roll_flag=current_slab_flags(3);
      current_slab_gauge=current_slab.gauge;
      current_slab_width=current_slab.width;
      %
      if (current_slab_anneal_flag == 0)  % check if coil has not been annealed
          array_out(1,1)=array_out(1,1)+1;
          %
      elseif (current_slab_anneal_flag == 1) & (current_slab_cold_roll_flag == 0)% slab has been annealed but not CR'd
          %
          time_in_highbay=run_index-time_of_entry;
          if time_in_highbay < cool_time_after_anneal
             array_out(2,1)=array_out(2,1)+1;
          else
             array_out(2,2)=array_out(2,2)+1;
          end
          %
      elseif (current_slab_cold_roll_flag == 1)
          %
          time_in_highbay=run_index-time_of_entry;
          [flag1,flag_ready]=determine_cold_status(current_slab,time_in_highbay,param);
          if (flag_ready==0) % Coil Hot
              array_out(3,1)=array_out(3,1)+1;
          elseif (flag_ready==1) % Coil Cold
              array_out(3,2)=array_out(3,2)+1;
          end
          %
      elseif (current_slab_cold_roll_flag == 2)
          %
          time_in_highbay=run_index-time_of_entry;
          [flag1,flag_ready]=determine_cold_status(current_slab,time_in_highbay,param);
          if (flag_ready==0) % Coil Hot
              array_out(4,1)=array_out(4,1)+1;
          elseif (flag_ready==1) % Coil Cold
              array_out(4,2)=array_out(4,2)+1;
          end
          %
      elseif (current_slab_cold_roll_flag == 3)
          %
          time_in_highbay=run_index-time_of_entry;
          [flag1,flag_ready]=determine_cold_status(current_slab,time_in_highbay,param);
          if (flag_ready==0) % Coil Hot
              array_out(5,1)=array_out(5,1)+1;
```

```
        elseif (flag_ready==1) % Coil Cold
            array_out(5,2)=array_out(5,2)+1;
        end
        %
    elseif (current_slab_cold_roll_flag == 4)
        %
        time_in_highbay=run_index-time_of_entry;
        [flag1,flag_ready]=determine_cold_status(current_slab,time_in_highbay,param);
        if (flag_ready==0) % Coil Hot
            array_out(6,1)=array_out(6,1)+1;
        elseif (flag_ready==1) % Coil Cold
            array_out(6,2)=array_out(6,2)+1;
        end
        %
    end
%
end % for
%
%----------------------------- end of determine_slabs_colour1.m ------------
```

▶ Function determine_slabs_for_anneal:

```
function [out_slabs,index_out]=determine_slabs_for_anneal(in_slabs);
% function [out_slabs,index_out]=determine_slabs_for_anneal(in_slabs);
% select 4 slabs of compatible thickness for annealing machine starting
% from top. If no 4 compatible types exist out_slabs=[];

 no_in_slabs=size(in_slabs,1); %
 if no_in_slabs==0
    out_slabs=[];
    index_out=[];
    return
 end
%
index_out=1:no_in_slabs;
%
while ~isempty(in_slabs)
        %
        no_slabs=size(in_slabs,1);
        top_slab=in_slabs(1,:); % get top slab
        top_slab_width=top_slab.width;
        top_special_flag=top_slab.special_flag;
        %
           index_array1=[];
           index_array2=[];

           for i=1:no_slabs
               if in_slabs(i).width == top_slab_width & in_slabs(i).special_flag == top_special_flag
                   index_array1=[index_array1 ; i];
               else
                   index_array1 = index_array1;
               end
           end
           index_array2=index_array1';

           if size(index_array2,2) < 4 % no 4 special flags exist
              in_slabs(index_array2,:)=[]; % eliminate from list
              index_out(index_array2)=[];% ... and from index set
           else
              out_slabs=in_slabs(index_array2(1:4),:);
              index_out=index_out(index_array2(1:4));
              return
           end
        %
   end
%
   out_slabs=[];
   index_out=[];
%
%----------------------- end of determine_slabs_for_anneal.m -------------------
```

▶ Function determine_slabs_for_rolling1:

```
function
[out_slabs,index_out]=determine_slabs_for_rolling1(in_slabs);
% function [out_slabs,index_out]=determine_slabs_for_rolling1(in_slabs);
% select up to 40 or up to 20  grouped coils for rolling machine starting
% from top. If no 40 or 20 coils compatible types exist, out_slabs=[];
%
% in_slabs:  list of slabs that have been annealed and stored to High-Bay for
%            cooling
% out_slabs: selected list of matching slabs from in_slabs list (empty if
%            no match found)
% index_out: indexes of matching slabs in in_slabs list (empty if no match
%            found)

 in_slabs=check_empty(in_slabs);
 if isempty(in_slabs)
    out_slabs=[];
    index_out=[];
    return
 end
 %
 no_in_slabs=size(in_slabs,1); %
 %
```

```
group1=[]; % yellow
ind_group1=[];
%
group2=[]; % light green
ind_group2=[];
%
group3=[]; % orange
ind_group3=[];
%
group4=[]; % dark green
ind_group4=[];
%
for i=1:no_in_slabs
    slab=in_slabs(i,:);
    %
    if all(slab.gauge=='thic')  | (all(slab.gauge=='stan')  & all(slab.width=='wide'))
       group1=[group1 ; slab];
       ind_group1=[ind_group1 i];
    elseif all(slab.gauge=='stan')  &  (all(slab.width=='medi')  | all(slab.width=='narr'))
       group2=[group2 ; slab];
       ind_group2=[ind_group2 i];
    elseif all(slab.gauge=='thin')  &  all(slab.width=='wide')
       group3=[group3 ; slab];
       ind_group3=[ind_group3 i];
     elseif all(slab.gauge=='thin')  &  (all(slab.width=='medi')  | all(slab.width=='narr'))
       group4=[group4 ; slab];
       ind_group4=[ind_group4 i];
     else
       out_slabs=[];
       index_out=[];
       return
    end
end
%
if size(ind_group2,2) >= 40
   out_slabs=group2(1:40,:);
   index_out=ind_group2(1:40);
elseif size(ind_group1,2) >= 20
   out_slabs=group1(1:20,:);
   index_out=ind_group1(1:20);
elseif size(ind_group3,2) >= 20
   out_slabs=group3(1:20,:);
   index_out=ind_group3(1:20);
elseif size(ind_group4,2) >= 20
   out_slabs=group4(1:20,:);
   index_out=ind_group4(1:20);
else
   out_slabs=[];
   index_out=[];
end
%
%------------------------------------ end of determine_slabs_for_rolling1.m --------------------------------
```

▶ Function determine_stock_codes:

```
  function stock_codes=determine_stock_codes(slabs_in)
% function stock_codes=determine_stock_codes(slabs_in)
% Function produces a list of (distinct) stock codes corresponding to
% slabs_in

no_slabs_in=size(slabs_in,1); if no_slabs_in == 0 |
size(slabs_in,2)==0
   stock_codes=[];
   return
end
%
stock_codes=[]; for i=1:no_slabs_in
    current_slab=slabs_in(i);
    current_stock_code=current_slab.stock_code;
    if ~any(stock_codes == current_stock_code);
        stock_codes=[stock_codes current_stock_code];
    end
end
%
% ---------------------- end determine_stock_codes -------------------
```

▶ Function diplay_list:

```
  function display_list(slabs)
% function display_list(slabs)

slabs=check_empty(slabs); no_slabs=size(slabs,1);

for i=1:no_slabs
    slab=slabs(i,:);
    type=slab.type;
    width=slab.width;
    gauge=slab.gauge;
    flag2=slab.flags(2);
    flag3=slab.flags(3);
    disp(['Type=',type,' Width=', width,' Gauge=',gauge,' Flag2=',int2str(flag2),' Flag3=',int2str(flag3)]);
end
%
%----------------------------- end display_list.m ---------------------------------------------------------
```

▶ Function min2hoursdays:

```
    function outstr=min2hoursdays(no_mins)
% function outstr=min2hoursdays(no_mins)

no_minutes=mod(no_mins,60); no_hours=floor(no_mins/60); if no_hours
>= 24
   no_days=floor(no_hours/24);
   no_hours=mod(no_hours,24);
else
   no_days=0;
end outstr=['[',num2str(no_days,2),' days; ',num2str(no_hours,2),'
hours; ',num2str(no_minutes,2),' min',']'];
%
%----------------------- end of min2hoursdays.m ---------------------------------------------------
```

▶ Function obtain_anneal_time:

```
  function anneal_time=obtain_anneal_time(slab_type)
% function anneal_time=obtain_anneal_time(slab_type)
% function defines annealing time in minutes according to slab-type
% special_demand_flag=1 for special slab types

  slab_width=slab_type.width;
  special_flag=slab_type.special_flag;
  if (all(slab_width == 'wide') & special_flag == 0)
     anneal_time=645; %129; % 10.75 hours or 645=5*129 minutes
  elseif (all(slab_width == 'medi') & special_flag == 0)
     anneal_time=585; %117; % 9.75 hours or 585=5*117 minutes
  elseif (all(slab_width == 'narr') & special_flag == 0)
     anneal_time=525; %105; % 8.75 hours or 525=5*105 minutes
  elseif (all(slab_width == 'wide') & special_flag == 1)
     anneal_time=765; %153; % 12.75 hours or 765=5*153 minutes
  elseif (all(slab_width == 'medi') & special_flag == 1)
     anneal_time=735; %147; % 12.25 hours or 735=5*147 minutes
  elseif (all(slab_width == 'narr') & special_flag == 1)
     anneal_time=675; %135; % 11.25 hours or 675=5*135 minutes
  else
     disp('Error in obtain_anneal_time.m ...')
     anneal_time=[];
     return
  end
%
%------------------------ end of obtain_anneal_time.m -----------------
```

▶ Function obtain_bwg_time:

```
  function bwg_time=obtain_bwg_time(slab_type)
% function bwg_time=obtain_bwg_time(slab_type)
% function defines bwg time in minutes according to slab-type
%
  slab_gauge=slab_type.gauge;
%
  if all(slab_gauge == 'thic')
     bwg_time=7; % 35 minutes (actual 35)
  elseif all(slab_gauge == 'stan')
     bwg_time=8; % 40 minutes (actual 41)
  elseif all(slab_gauge == 'thin')
     bwg_time=14; % 70 minutes (actual 69)
  else
     disp('Error in obtain_bwg_time.m ...')
     bwg_time=[];
     return
  end
%
%------------------------ end of obtain_bwg_time.m -----------------
```

▶ Function pad_blanks:

```
  function out_str=pad_blanks(in_str,n)
% function out_str=pad_blanks(in_str,n)
% Pad in_str whith leading blanks up to length n

nc=size(in_str,2); if nc >=n
   out_str=in_str;
else
   out_str=[blanks(n-nc) in_str];
end
%
%------------------ end of pad_blanks.m -------------
```

▶ M-file priorities123:

```
% priorities123
% script file called from inside update_highbay_new

% first priority (a): Load input buffer of cold-rolling machine
if ~isempty(slab_cr)
    out_to_in_cold_roll_buff=slab_cr;
    out_to_in_cold_roll_buff=remove_time_stamp(out_to_in_cold_roll_buff);
    new_state(slab_cr_index)=[];
    new_state=check_empty(new_state);
    % update list of matched coils destined for cold-rolling treatment
    %index_list_coils_cr
    if size(index_list_coils_cr,2)==1 %size(index_coils_cr,2)==1
       index_list_coils_cr=[]; % only coil in list is removed
```

```
    else
       index_list_coils_cr(1)=[];
       no_coils_in_list=size(index_list_coils_cr,2);
       % move up remaining coils in list by one place
       index_list_coils_cr=index_list_coils_cr-ones(1,no_coils_in_list);
    end
    available_places=available_places+1;
    crane_moves=crane_moves-1;
    real_crane_moves(3)=real_crane_moves(3)+1;
else
    out_to_in_cold_roll_buff=[];
end
%
% first priority (b): Bring in coils from out-cold-roll-buffer
no_slabs_out_cold_roll_buff=size(state_out_cold_roll_buff,1);
index1=min([no_slabs_out_cold_roll_buff available_places
crane_moves]);
%real_crane_moves=real_crane_moves+index1;% max(size(slab_cr,1),no_slabs_out_cold_roll_buff); % some trips are free!
%
if index1 > 0 % some coils have moved in from out-cold-roll-buffer
   %
   in_from_out_cold_roll_buff=state_out_cold_roll_buff(1:index1,:);
   in_from_out_cold_roll_buff=set_time_stamp(in_from_out_cold_roll_buff,time_step);
   new_state=[new_state ; in_from_out_cold_roll_buff];
   state_out_cold_roll_buff=update_out_cold_roll_buff(state_out_cold_roll_buff,[],in_from_out_cold_roll_buff);
   available_places=available_places-index1;
   crane_moves=crane_moves-index1;
   real_crane_moves(4)=real_crane_moves(4)+index1;
   if crane_moves == 0
      out_to_in_BWG_buff=[];
      in_from_buff=[];
      check_out_variables;
      return
   end
   %
elseif index1 == 0 % no coils moved in from out-cold-roll-buffer
   %
   in_from_out_cold_roll_buff=[];
   %
   if crane_moves==0
      out_to_in_BWG_buff=[];
      in_from_buff=[];
      in_from_out_cold_roll_buff=[];
      check_out_variables;
      return
   end
end
%-------------------------------------------------------------
% second priority: Load out to BWG input buffer
index2=min(no_out_to_in_BWG_buff,crane_moves);
if index2 > 0 % some slabs can be moved out to in-BWG-buffer
   out_to_in_BWG_buff=out_to_in_BWG_buff(1:index2,:);
   index_slabs_to_in_BWG_buff=index_slabs_to_in_BWG_buff(1:index2,:);
   out_to_in_BWG_buff=remove_time_stamp(out_to_in_BWG_buff);
   new_state(index_slabs_to_in_BWG_buff,:)=[];
   index_list_coils_cr=update_list(index_list_coils_cr,index_slabs_to_in_BWG_buff);
   available_places=available_places+index2;
   crane_moves=crane_moves-index2;
   real_crane_moves(5)=real_crane_moves(5)+index2;
   if crane_moves == 0
      in_from_buff=[];
      check_out_variables;
      return
   end
   %
elseif index2 == 0 % no coils loaded out to BWG input buffer
   %
   out_to_in_BWG_buff=[];
   %
   if crane_moves==0
      in_from_buff=[];
      check_out_variables;
      return
   end
end
%-----------------------------------------------
% 3rd priority: Load in coils from input buffer

state_buff=check_empty(state_buff);
no_slabs_in_buff=size(state_buff,1);
available_places_virtual=max(0,available_places-max_no_circulating_coils);
index3=min([no_slabs_in_buff available_places_virtual crane_moves]);
%
if index3 > 0 % some coils can be moved into HighBay from input buffer
   in_from_buff=state_buff(1:index3,:);
   state_buff=update_buff(state_buff,[],in_from_buff);
   in_from_buff=set_time_stamp(in_from_buff,time_step);
   new_state=[new_state ; in_from_buff];
   available_places=available_places-index3;
   crane_moves=crane_moves-index3;
   real_crane_moves(1)=real_crane_moves(1)+index3;
   %
elseif index3 == 0 % No coils loaded into HB from input buffer
   in_from_buff=[];
end
%
check_out_variables;
%------------------------------ priorities123.m -------------------------
```

▶ M-file priorities132:

```
% priorities132
% script file called from inside update_highbay_new
% CR, Entry, BWG

% first priority (a): Load input buffer of cold-rolling machine
if ~isempty(slab_cr)
    out_to_in_cold_roll_buff=slab_cr;
    out_to_in_cold_roll_buff=remove_time_stamp(out_to_in_cold_roll_buff);
    new_state(slab_cr_index)=[];
    new_state=check_empty(new_state);
    % update list of matched coils destined for cold-rolling treatment
    %index_list_coils_cr
    if size(index_list_coils_cr,2)==1 %size(index_coils_cr,2)==1
       index_list_coils_cr=[]; % only coil in list is removed
    else
       index_list_coils_cr(1)=[];
       no_coils_in_list=size(index_list_coils_cr,2);
       % move up remaining coils in list by one place
       index_list_coils_cr=index_list_coils_cr-ones(1,no_coils_in_list);
    end
    available_places=available_places+1;
    crane_moves=crane_moves-1;
    real_crane_moves(3)=real_crane_moves(3)+1;
else
    out_to_in_cold_roll_buff=[];
end
% out_to_in_cold_roll_buff defined
%
% first priority (b): Bring in coils from out-cold-roll-buffer
no_slabs_out_cold_roll_buff=size(state_out_cold_roll_buff,1);
index1=min([no_slabs_out_cold_roll_buff available_places
crane_moves]);
%real_crane_moves=max(size(slab_cr,1),no_slabs_out_cold_roll_buff); % some trips are free!
%
if index1 > 0 % some coils have moved in from out-cold-roll-buffer
   %
   in_from_out_cold_roll_buff=state_out_cold_roll_buff(1:index1,:);
   in_from_out_cold_roll_buff=set_time_stamp(in_from_out_cold_roll_buff,time_step);
   new_state=[new_state ; in_from_out_cold_roll_buff];
   state_out_cold_roll_buff=update_out_cold_roll_buff(state_out_cold_roll_buff,[],in_from_out_cold_roll_buff);
   available_places=available_places-index1;
   crane_moves=crane_moves-index1;
   real_crane_moves(4)=real_crane_moves(4)+index1;
   if crane_moves == 0
      out_to_in_BWG_buff=[];
      in_from_buff=[];
      check_out_variables;
      return
   end
   %
elseif index1 == 0 % no coils moved in from out-cold-roll-buffer
   %
   in_from_out_cold_roll_buff=[];
   %
   if crane_moves==0
      out_to_in_BWG_buff=[];
      in_from_buff=[];
      %in_from_out_cold_roll_buff=[];
      check_out_variables;
      return
   end
end
% in_from_out_cold_roll_buff defined
%-------------------------------------------
% 2nd priority: Load in coils from input buffer

state_buff=check_empty(state_buff);
no_slabs_in_buff=size(state_buff,1);
available_places_virtual=max(0,available_places-max_no_circulating_coils);
index3=min([no_slabs_in_buff available_places_virtual crane_moves]);
%
if index3 > 0 % some coils can be moved into HighBay from input buffer
   in_from_buff=state_buff(1:index3,:);
   state_buff=update_buff(state_buff,[],in_from_buff);
   in_from_buff=set_time_stamp(in_from_buff,time_step);
   new_state=[new_state ; in_from_buff];
   available_places=available_places-index3;
   crane_moves=crane_moves-index3;
   real_crane_moves(1)=real_crane_moves(1)+index3;
   if crane_moves == 0
      out_to_in_BWG_buff=[];
      check_out_variables;
      return
   end
   %
elseif index3 == 0 % No coils loaded into HB from input buffer
   in_from_buff=[];
   if crane_moves == 0
      out_to_in_BWG_buff=[];
      check_out_variables;
      return
   end
end
% in_from_buff defined
%-----------------------------------------------
% third priority: Load out to BWG input buffer
```

```
index2=min(no_out_to_in_BWG_buff,crane_moves);
if index2 > 0 % some slabs can be moved out to in-BWG-buffer
   out_to_in_BWG_buff=out_to_in_BWG_buff(1:index2,:);
   index_slabs_to_in_BWG_buff=index_slabs_to_in_BWG_buff(1:index2,:);
   out_to_in_BWG_buff=remove_time_stamp(out_to_in_BWG_buff);
   new_state(index_slabs_to_in_BWG_buff,:)=[];
   index_list_coils_cr=update_list(index_list_coils_cr,index_slabs_to_in_BWG_buff);
   available_places=available_places+index2;
   crane_moves=crane_moves-index2;
   real_crane_moves(5)=real_crane_moves(5)+index2;
   if crane_moves == 0
      %in_from_buff=[];
      check_out_variables;
      return
   end
   %
elseif index2 == 0 % no coils loaded out to BWG input buffer
   %
   out_to_in_BWG_buff=[];
   %
   if crane_moves==0
      %in_from_buff=[];
      check_out_variables;
      return
   end
end check_out_variables;
%
%---------------------------------- end of priorities132.m ------------------
```

▶ M-file priorities213:

```
% priorities213
% script file called from inside update_highbay_new
% BWG, CR, Entry

% first priority: Load out to BWG input buffer
index2=min(no_out_to_in_BWG_buff,crane_moves);
if index2 > 0 % some slabs can be moved out to in-BWG-buffer
   out_to_in_BWG_buff=out_to_in_BWG_buff(1:index2,:);
   index_slabs_to_in_BWG_buff=index_slabs_to_in_BWG_buff(1:index2,:);
   out_to_in_BWG_buff=remove_time_stamp(out_to_in_BWG_buff);
   new_state(index_slabs_to_in_BWG_buff,:)=[];
   index_list_coils_cr=update_list(index_list_coils_cr,index_slabs_to_in_BWG_buff);
   available_places=available_places+index2;
   crane_moves=crane_moves-index2;
   real_crane_moves(5)=real_crane_moves(5)+index2;
   if crane_moves == 0
      in_from_buff=[];
      out_to_in_cold_roll_buff=[];
      in_from_out_cold_roll_buff=[];
      check_out_variables;
      return
   end
   %
elseif index2 == 0 % no coils loaded out to BWG input buffer
   %
   out_to_in_BWG_buff=[];
   %
   if crane_moves==0
      in_from_buff=[];
      out_to_in_cold_roll_buff=[];
      in_from_out_cold_roll_buff=[];
      check_out_variables;
      return
   end
end
% out_to_in_BWG_buff defined
%-------------------------------------------------------------------------
% second priority (a): Load input buffer of cold-rolling machine
if ~isempty(slab_cr)
    out_to_in_cold_roll_buff=slab_cr;
    out_to_in_cold_roll_buff=remove_time_stamp(out_to_in_cold_roll_buff);
    new_state(slab_cr_index)=[];
    new_state=check_empty(new_state);
    % update list of matched coils destined for cold-rolling treatment
    %index_list_coils_cr
    if size(index_list_coils_cr,2)==1 %size(index_coils_cr,2)==1
       index_list_coils_cr=[]; % only coil in list is removed
    else
       index_list_coils_cr(1)=[];
       no_coils_in_list=size(index_list_coils_cr,2);
       % move up remaining coils in list by one place
       index_list_coils_cr=index_list_coils_cr-ones(1,no_coils_in_list);
    end
    available_places=available_places+1;
    crane_moves=crane_moves-1;
    real_crane_moves(3)=real_crane_moves(3)+1;
else
    out_to_in_cold_roll_buff=[];
end
%out_to_in_cold_roll_buff defined
%
% second priority (b): Bring in coils from out-cold-roll-buffer
no_slabs_out_cold_roll_buff=size(state_out_cold_roll_buff,1);
index1=min([no_slabs_out_cold_roll_buff available_places
crane_moves]);
%real_crane_moves=max(size(slab_cr,1),no_slabs_out_cold_roll_buff); % some trips are free!
%
```

```
if index1 > 0 % some coils have moved in from out-cold-roll-buffer
   %
   in_from_out_cold_roll_buff=state_out_cold_roll_buff(1:index1,:);
   in_from_out_cold_roll_buff=set_time_stamp(in_from_out_cold_roll_buff,time_step);
   new_state=[new_state ; in_from_out_cold_roll_buff];
   state_out_cold_roll_buff=update_out_cold_roll_buff(state_out_cold_roll_buff,[],in_from_out_cold_roll_buff);
   available_places=available_places-index1;
   crane_moves=crane_moves-index1;
   real_crane_moves(4)=real_crane_moves(4)+index1;
   if crane_moves == 0
      % out_to_in_BWG_buff=[];
      in_from_buff=[];
      check_out_variables;
      return
   end
   %
elseif index1 == 0 % no coils moved in from out-cold-roll-buffer
   %
   in_from_out_cold_roll_buff=[];
   %
   if crane_moves==0
      % out_to_in_BWG_buff=[];
      in_from_buff=[];
      check_out_variables;
      return
   end
end
% in_from_out_cold_roll_buff defined
%-------------------------------------------------------------------------

% 3rd priority: Load in coils from input buffer

state_buff=check_empty(state_buff);
no_slabs_in_buff=size(state_buff,1);
available_places_virtual=max(0,available_places-max_no_circulating_coils);
index3=min([no_slabs_in_buff available_places_virtual crane_moves]);
%
if index3 > 0 % some coils can be moved into HighBay from input buffer
   in_from_buff=state_buff(1:index3,:);
   state_buff=update_buff(state_buff,[],in_from_buff);
   in_from_buff=set_time_stamp(in_from_buff,time_step);
   new_state=[new_state ; in_from_buff];
   available_places=available_places-index3;
   crane_moves=crane_moves-index3;
   real_crane_moves(1)=real_crane_moves(1)+index3;
   %
elseif index3 == 0 % No coils loaded into HB from input buffer
   in_from_buff=[];
end check_out_variables;
%------------------------------ priorities213.m ------------------------
```

▶ M-file priorities231:

```
% priorities231
% script file called from inside update_highbay_new
% BWG, Entry, CR

% first priority: Load out to BWG input buffer
index2=min(no_out_to_in_BWG_buff,crane_moves);
if index2 > 0 % some slabs can be moved out to in-BWG-buffer
   out_to_in_BWG_buff=out_to_in_BWG_buff(1:index2,:);
   index_slabs_to_in_BWG_buff=index_slabs_to_in_BWG_buff(1:index2,:);
   out_to_in_BWG_buff=remove_time_stamp(out_to_in_BWG_buff);
   new_state(index_slabs_to_in_BWG_buff,:)=[];
   index_list_coils_cr=update_list(index_list_coils_cr,index_slabs_to_in_BWG_buff);
   available_places=available_places+index2;
   crane_moves=crane_moves-index2;
   real_crane_moves(5)=real_crane_moves(5)+index2;
   if crane_moves == 0
      in_from_buff=[];
      out_to_in_cold_roll_buff=[];
      in_from_out_cold_roll_buff=[];
      check_out_variables;
      return
   end
   %
elseif index2 == 0 % no coils loaded out to BWG input buffer
   %
   out_to_in_BWG_buff=[];
   %
   if crane_moves==0
      in_from_buff=[];
      out_to_in_cold_roll_buff=[];
      in_from_out_cold_roll_buff=[];
      check_out_variables;
      return
   end
end
%
%-------------------------------------------------------------------------
% 2nd priority: Load in coils from input buffer

state_buff=check_empty(state_buff);
no_slabs_in_buff=size(state_buff,1);
available_places_virtual=max(0,available_places-max_no_circulating_coils);
index3=min([no_slabs_in_buff available_places_virtual crane_moves]);
%
if index3 > 0 % some coils can be moved into HighBay from input buffer
```

```
   in_from_buff=state_buff(1:index3,:);
   state_buff=update_buff(state_buff,[],in_from_buff);
   in_from_buff=set_time_stamp(in_from_buff,time_step);
   new_state=[new_state ; in_from_buff];
   available_places=available_places-index3;
   crane_moves=crane_moves-index3;
   real_crane_moves(1)=real_crane_moves(1)+index3;
   %
   if crane_moves == 0
      in_from_out_cold_roll_buff=[];
      out_to_in_cold_roll_buff=[];
      check_out_variables;
      return
   end

elseif index3 == 0 % No coils loaded into HB from input buffer
   in_from_buff=[];
   %
   if crane_moves == 0
      in_from_out_cold_roll_buff=[];
      out_to_in_cold_roll_buff=[];
      check_out_variables;
      return
   end
end
% in_from_buff is defined!
%-------------------------------------------------------------------------

% third priority (a): Load input buffer of cold-rolling machine
if ~isempty(slab_cr)
    out_to_in_cold_roll_buff=slab_cr;
    out_to_in_cold_roll_buff=remove_time_stamp(out_to_in_cold_roll_buff);
    new_state(slab_cr_index)=[];
    new_state=check_empty(new_state);
    % update list of matched coils destined for cold-rolling treatment
    %index_list_coils_cr
    if size(index_list_coils_cr,2)==1 %size(index_coils_cr,2)==1
       index_list_coils_cr=[]; % only coil in list is removed
    else
       index_list_coils_cr(1)=[];
       no_coils_in_list=size(index_list_coils_cr,2);
       % move up remaining coils in list by one place
       index_list_coils_cr=index_list_coils_cr-ones(1,no_coils_in_list);
    end
    available_places=available_places+1;
    crane_moves=crane_moves-1;
    real_crane_moves(3)=real_crane_moves(3)+1;
else
    out_to_in_cold_roll_buff=[];
end
%
% second priority (b): Bring in coils from out-cold-roll-buffer
no_slabs_out_cold_roll_buff=size(state_out_cold_roll_buff,1);
index1=min([no_slabs_out_cold_roll_buff available_places
crane_moves]);
% real_crane_moves=max(size(slab_cr,1),no_slabs_out_cold_roll_buff); % some trips are free!
%
if index1 > 0 % some coils have moved in from out-cold-roll-buffer
   %
   in_from_out_cold_roll_buff=state_out_cold_roll_buff(1:index1,:);
   in_from_out_cold_roll_buff=set_time_stamp(in_from_out_cold_roll_buff,time_step);
   new_state=[new_state ; in_from_out_cold_roll_buff];
   state_out_cold_roll_buff=update_out_cold_roll_buff(state_out_cold_roll_buff,[],in_from_out_cold_roll_buff);
   available_places=available_places-index1;
   crane_moves=crane_moves-index1;
   real_crane_moves(4)=real_crane_moves(4)+index1;
   if crane_moves == 0
      check_out_variables;
      return
   end
   %
elseif index1 == 0 % no coils moved in from out-cold-roll-buffer
   %
   in_from_out_cold_roll_buff=[];
   %
   if crane_moves==0
      check_out_variables;
      return
   end
end check_out_variables;
%------------------------------ priorities231.m -------------------------
```

▶ M-file priorities312:

```
% priorities312
% script file called from inside update_highbay_new
% Entry, CR, BWG

% 1rst priority: Load in coils from input buffer

state_buff=check_empty(state_buff);
no_slabs_in_buff=size(state_buff,1);
available_places_virtual=max(0,available_places-max_no_circulating_coils);
index3=min([no_slabs_in_buff available_places_virtual crane_moves]);
%
if index3 > 0 % some coils can be moved into HighBay from input buffer
   in_from_buff=state_buff(1:index3,:);
```

```
   state_buff=update_buff(state_buff,[],in_from_buff);
   in_from_buff=set_time_stamp(in_from_buff,time_step);
   new_state=[new_state ; in_from_buff];
   available_places=available_places-index3;
   crane_moves=crane_moves-index3;
   real_crane_moves(1)=real_crane_moves(1)+index3;
   %
   if crane_moves == 0
      in_from_out_cold_roll_buff=[];
      out_to_in_cold_roll_buff=[];
      out_to_in_BWG_buff=[];
      check_out_variables;
      return
   end

elseif index3 == 0 % No coils loaded into HB from input buffer
   in_from_buff=[];
   %
   if crane_moves == 0
      in_from_out_cold_roll_buff=[];
      out_to_in_cold_roll_buff=[];
      out_to_in_BWG_buff=[];
      check_out_variables;
      return
   end

end
% in_from_buff is defined!
%-------------------------------------------------------------------------
% second priority (a): Load input buffer of cold-rolling machine
if ~isempty(slab_cr)
    out_to_in_cold_roll_buff=slab_cr;
    out_to_in_cold_roll_buff=remove_time_stamp(out_to_in_cold_roll_buff);
    new_state(slab_cr_index)=[];
    new_state=check_empty(new_state);
    % update list of matched coils destined for cold-rolling treatment
    %index_list_coils_cr
    if size(index_list_coils_cr,2)==1 %size(index_coils_cr,2)==1
       index_list_coils_cr=[]; % only coil in list is removed
    else
       index_list_coils_cr(1)=[];
       no_coils_in_list=size(index_list_coils_cr,2);
       % move up remaining coils in list by one place
       index_list_coils_cr=index_list_coils_cr-ones(1,no_coils_in_list);
    end
    available_places=available_places+1;
    crane_moves=crane_moves-1;
    real_crane_moves(3)=real_crane_moves(3)+1;
else
    out_to_in_cold_roll_buff=[];
end
%
% second priority (b): Bring in coils from out-cold-roll-buffer
no_slabs_out_cold_roll_buff=size(state_out_cold_roll_buff,1);
index1=min([no_slabs_out_cold_roll_buff available_places
crane_moves]);
% real_crane_moves=max(size(slab_cr,1),no_slabs_out_cold_roll_buff); % some trips are free!
%
if index1 > 0 % some coils have moved in from out-cold-roll-buffer
   %
   in_from_out_cold_roll_buff=state_out_cold_roll_buff(1:index1,:);
   in_from_out_cold_roll_buff=set_time_stamp(in_from_out_cold_roll_buff,time_step);
   new_state=[new_state ; in_from_out_cold_roll_buff];
   state_out_cold_roll_buff=update_out_cold_roll_buff(state_out_cold_roll_buff,[],in_from_out_cold_roll_buff);
   available_places=available_places-index1;
   crane_moves=crane_moves-index1;
   real_crane_moves(4)=real_crane_moves(4)+index1;
   if crane_moves == 0
      out_to_in_BWG_buff=[];
      check_out_variables;
      return
   end
   %
elseif index1 == 0 % no coils moved in from out-cold-roll-buffer
   %
   in_from_out_cold_roll_buff=[];
   %
   if crane_moves==0
      out_to_in_BWG_buff=[];
      check_out_variables;
      return
   end
       %
end
%-------------------------------------------------------------------------
% third priority: Load out to BWG input buffer
index2=min(no_out_to_in_BWG_buff,crane_moves);
if index2 > 0 % some slabs can be moved out to in-BWG-buffer
   out_to_in_BWG_buff=out_to_in_BWG_buff(1:index2,:);
   index_slabs_to_in_BWG_buff=index_slabs_to_in_BWG_buff(1:index2,:);
   out_to_in_BWG_buff=remove_time_stamp(out_to_in_BWG_buff);
   new_state(index_slabs_to_in_BWG_buff,:)=[];
   index_list_coils_cr=update_list(index_list_coils_cr,index_slabs_to_in_BWG_buff);
   available_places=available_places+index2;
   crane_moves=crane_moves-index2;
   real_crane_moves(5)=real_crane_moves(5)+index2;
   if crane_moves == 0
      check_out_variables;
```

```
      return
   end
   %
elseif index2 == 0 % no coils loaded out to BWG input buffer
   %
   out_to_in_BWG_buff=[];
   %
   if crane_moves==0
      check_out_variables;
      return
   end
end check_out_variables;
%-------------------------------- prioritiies312.m ------------------------
```

▶ M-file prioritiies321:

```
% prioritiies321
% script file called from inside update_highbay_new
% Entry, BWG, CR

% first priority: Load in coils from input buffer
state_buff=check_empty(state_buff);
no_slabs_in_buff=size(state_buff,1);
available_places_virtual=max(0,available_places-max_no_circulating_coils);
index3=min([no_slabs_in_buff available_places_virtual crane_moves]);
%
if index3 > 0 % some coils can be moved into HighBay from input buffer
   in_from_buff=state_buff(1:index3,:);
   state_buff=update_buff(state_buff,[],in_from_buff);
   in_from_buff=set_time_stamp(in_from_buff,time_step);
   new_state=[new_state ; in_from_buff];
   available_places=available_places-index3;
   crane_moves=crane_moves-index3;
   real_crane_moves(1)=real_crane_moves(1)+index3;
   %
   if crane_moves == 0
      in_from_out_cold_roll_buff=[];
      out_to_in_cold_roll_buff=[];
      out_to_in_BWG_buff=[];
      check_out_variables;
      return
   end

elseif index3 == 0 % No coils loaded into HB from input buffer
   in_from_buff=[];
   %
   if crane_moves == 0
      in_from_out_cold_roll_buff=[];
      out_to_in_cold_roll_buff=[];
      out_to_in_BWG_buff=[];
      check_out_variables;
      return
   end
end
% in_from_buff is defined!
%-------------------------------------------------------------------------
% second priority: Load out to BWG input buffer
index2=min(no_out_to_in_BWG_buff,crane_moves);
if index2 > 0 % some slabs can be moved out to in-BWG-buffer
   out_to_in_BWG_buff=out_to_in_BWG_buff(1:index2,:);
   index_slabs_to_in_BWG_buff=index_slabs_to_in_BWG_buff(1:index2,:);
   out_to_in_BWG_buff=remove_time_stamp(out_to_in_BWG_buff);
   new_state(index_slabs_to_in_BWG_buff,:)=[];
   index_list_coils_cr=update_list(index_list_coils_cr,index_slabs_to_in_BWG_buff);
   available_places=available_places+index2;
   crane_moves=crane_moves-index2;
   real_crane_moves(5)=real_crane_moves(5)+index2;
   if crane_moves == 0
      out_to_in_cold_roll_buff=[];
      in_from_out_cold_roll_buff=[];
      check_out_variables;
      return
   end
   %
elseif index2 == 0 % no coils loaded out to BWG input buffer
   %
   out_to_in_BWG_buff=[];
   %
   if crane_moves==0
      out_to_in_cold_roll_buff=[];
      in_from_out_cold_roll_buff=[];
      check_out_variables;
      return
   end
end
%
%-------------------------------------------------------------------------
% third priority (a): Load input buffer of cold-rolling machine
if ~isempty(slab_cr)
    out_to_in_cold_roll_buff=slab_cr;
    out_to_in_cold_roll_buff=remove_time_stamp(out_to_in_cold_roll_buff);
    new_state(slab_cr_index)=[];
    new_state=check_empty(new_state);
    % update list of matched coils destined for cold-rolling treatment
    %index_list_coils_cr
    if size(index_list_coils_cr,2)==1 %size(index_coils_cr,2)==1
       index_list_coils_cr=[]; % only coil in list is removed
    else
```

```
      index_list_coils_cr(1)=[];
      no_coils_in_list=size(index_list_coils_cr,2);
      % move up remaining coils in list by one place
      index_list_coils_cr=index_list_coils_cr-ones(1,no_coils_in_list);
   end
   available_places=available_places+1;
   crane_moves=crane_moves-1;
   real_crane_moves(3)=real_crane_moves(3)+1;
else
   out_to_in_cold_roll_buff=[];
end
%
% second priority (b): Bring in coils from out-cold-roll-buffer
no_slabs_out_cold_roll_buff=size(state_out_cold_roll_buff,1);
index1=min([no_slabs_out_cold_roll_buff available_places
crane_moves]);
% real_crane_moves=max(size(slab_cr,1),no_slabs_out_cold_roll_buff); % some trips are free!
%
if index1 > 0 % some coils have moved in from out-cold-roll-buffer
   %
   in_from_out_cold_roll_buff=state_out_cold_roll_buff(1:index1,:);
   in_from_out_cold_roll_buff=set_time_stamp(in_from_out_cold_roll_buff,time_step);
   new_state=[new_state ; in_from_out_cold_roll_buff];
   state_out_cold_roll_buff=update_out_cold_roll_buff(state_out_cold_roll_buff,[],in_from_out_cold_roll_buff);
   available_places=available_places-index1;
   crane_moves=crane_moves-index1;
   real_crane_moves(4)=real_crane_moves(4)+index1;
   if crane_moves == 0
      check_out_variables;
      return
   end
   %
elseif index1 == 0 % no coils moved in from out-cold-roll-buffer
   %
   in_from_out_cold_roll_buff=[];
   %
   if crane_moves==0
      check_out_variables;
      return
   end
       %
end check_out_variables;
%------------------------------ priorities321.m -------------------------
```

▶ Function read_from_xls:

```
function slabs_list=read_from_xls(filename,day_no)
% function slabs_list=read_from_xls(filename,day_no)
% Function reads data from xls file (row day_no) and converts it into
% list structure

a=xlsread(filename); nr=size(a,1); nc=size(a,2);
%
if round(nc/2) ~= nc/2
   disp('Error in read_from_xls.m ...')
   slabs_list=[];
   return
end
%
if day_no > nr
   slabs_list=[];
   return
end
%-------------------------------------------------------------------------
%SLAB_TYPES
slab_type_1 =
struct('type','type_01','width','wide','special_flag',0,'gauge','thic','timer',-1,'flags',[0
0 0 0]); slab_type_2 =
struct('type','type_02','width','medi','special_flag',0,'gauge','thic','timer',-1,'flags',[0
0 0 0]); slab_type_3 =
struct('type','type_03','width','narr','special_flag',0,'gauge','thic','timer',-1,'flags',[0
0 0 0]); slab_type_4 =
struct('type','type_04','width','wide','special_flag',0,'gauge','stan','timer',-1,'flags',[0
0 0 0]);
%
slab_type_5 =
struct('type','type_05','width','medi','special_flag',0,'gauge','stan','timer',-1,'flags',[0
0 0 0]); slab_type_6 =
struct('type','type_06','width','narr','special_flag',0,'gauge','stan','timer',-1,'flags',[0
0 0 0]);
%
slab_type_7 =
struct('type','type_07','width','wide','special_flag',0,'gauge','thin','timer',-1,'flags',[0
0 0 0]);

slab_type_8 =
struct('type','type_08','width','medi','special_flag',0,'gauge','thin','timer',-1,'flags',[0
0 0 0]); slab_type_9 =
struct('type','type_09','width','narr','special_flag',0,'gauge','thin','timer',-1,'flags',[0
0 0 0]);

slab_type_10 =
struct('type','type_10','width','wide','special_flag',1,'gauge','thic','timer',-1,'flags',[0
0 0 0]); slab_type_11 =
struct('type','type_11','width','medi','special_flag',1,'gauge','thic','timer',-1,'flags',[0
0 0 0]); slab_type_12 =
struct('type','type_12','width','narr','special_flag',1,'gauge','thic','timer',-1,'flags',[0
0 0 0]); slab_type_13 =
```

```
struct('type','type_13','width','wide','special_flag',1,'gauge','stan','timer',-1,'flags',[0
0 0 0]);

slab_type_14 =
struct('type','type_14','width','medi','special_flag',1,'gauge','stan','timer',-1,'flags',[0
0 0 0]); slab_type_15 =
struct('type','type_15','width','narr','special_flag',1,'gauge','stan','timer',-1,'flags',[0
0 0 0]);

slab_type_16 =
struct('type','type_16','width','wide','special_flag',1,'gauge','thin','timer',-1,'flags',[0
0 0 0]);

slab_type_17 =
struct('type','type_17','width','medi','special_flag',1,'gauge','thin','timer',-1,'flags',[0
0 0 0]); slab_type_18 =
struct('type','type_18','width','narr','special_flag',1,'gauge','thin','timer',-1,'flags',[0
0 0 0]);
%-------------------------------------------------------------------------
n=round(nc/2); no_slabs_array=[]; slab_types1=[]; for i=1:n
    type_no=a(day_no,(i-1)*2+1);
    no_slabs=a(day_no,(i-1)*2+2);
    %
    if ((~isnan(type_no)) & (~isnan(no_slabs)) & (type_no > 0) & (type_no <= 18) & (no_slabs == round(no_slabs)) & (no_slabs > 0))
        string_temp=['slab_type_',int2str(type_no)];
        slab=eval(string_temp);
        slab_types1=[slab_types1 ; slab];
        no_slabs_array=[no_slabs_array no_slabs];
    end
    %
end
%
if isempty(no_slabs_array) | isempty(slab_types1)
   slabs_list=[];
else
   slabs_list=specify_slabs(no_slabs_array',slab_types1);
end
%
%---------------------------------- end of read_from_xls.m -----------
```

▶ Function read_from_xls_all:

```
  function [slabs_list,day_index]=read_from_xls_all(filename)
% function [slabs_list,day_index]=read_from_xls_all(filename)
% Function reads data from xls file all data and converts it into
% a consecutive list structure. Day index is corresponding day-index
% array (row-vector)

a=xlsread(filename); nr=size(a,1); nc=size(a,2);
%
if round(nc/2) ~= nc/2
   disp('Error in read_from_xls.m ...')
   slabs_list=[];
   day_index=[];
   return
end
%
slabs_list=[]; day_index=[];
%
for i=1:nr % no of days
    slabs_list_day_i=read_from_xls(filename,i);
    slabs_list_day_i=check_empty(slabs_list_day_i);
    no_slabs_day_i=size(slabs_list_day_i,1); % no slabs
    if no_slabs_day_i ~= 0
       slabs_list=[slabs_list ; slabs_list_day_i]; % append
       day_index=[day_index i*ones(1,no_slabs_day_i)];
    end
end
%
%---------------------------------- end of read_from_xls_all.m -----------
```

▶ Function remove_time_stamp:

```
  function slabs=remove_time_stamp(slabs)
% function slabs=remove_time_stamp(slabs)
% Remove time-stamp from slabs leaving HighBay
%
  no_slabs=size(slabs,1); % no slabs
  if no_slabs == 0 | size(slabs,2) == 0
     slabs=[];
     return
  end
%
  for i=1:no_slabs
      current_slab=slabs(i); % get current slab
      timer_current_slab=current_slab.timer; % get flags field
      timer_current_slab=-1; % set to -1 after exiting HB
      current_slab = setfield(current_slab,'timer',timer_current_slab);
      slabs(i)=current_slab; % substitute back
  end
%
%---------------------------------- end of remove_time_stamp.m -----------------------------------
```

▶ M-file run_process_ic:

```
run_process_ic
%
cur_dir=pwd;
data_dir=[cur_dir,'\data']; % data directory
%-------------------------------------------------------------------------------
% DEFINE PROCESS PARAMETERS
%
  %param=default_parameters;

  simulation_step=param.simulation_step; % minutes
  max_iter=ceil((param.simulation_time)*60/simulation_step); % maximum number of iterations
  simulation_step=param.simulation_step;
  no_anneal_machines=param.no_anneal_machines; % number of annealing machines in parallel
  high_bay_capacity=param.high_bay_capacity;
  buff_capacity=param.buff_capacity;
  capacity_buff1_n=param.buff1_n_capacity; % capacity of ANNEAL buffer (in-side)
  out_anneal_buff_capacity=param.out_anneal_buff_capacity;
  in_cold_roll_buff_capacity=param.in_cold_roll_buff_capacity;
  out_cold_roll_buff_capacity=param.out_cold_roll_buff_capacity;
  plot_flag=param.plot_flag; % 0=No plots, 1=Complete process, 2=Highbay
  pause_time=param.pause_time; % Pause-time for plots
  %------
  input_file=param.input_file;
  input_file=[data_dir,'\',input_file];
  %------
  stats_plot_flag=param.stats_plot_flag; % 0=No plots, 1=Plots
  out_file_flag=param.out_file_flag; % 1=Save output stats file, 0=No output file
  %------
  out_file_name=param.out_file_name; % output file name
  out_file_name=[data_dir,'\',out_file_name];
  %------
  initial_conditions_flag=param.initial_conditions_flag; % 1=Use Initial Data, 0=No Initial Conditions
  %------
  initial_conditions_file=param.initial_conditions_file; % initial conditions file (mat file)
  initial_conditions_file=[data_dir,'\',initial_conditions_file];
%
% load_rate=param.load_rate; % not used
% cool_time_after_anneal % not used at the moment
% cool_time_after_coldroll % not used
% crane_moves % not used
% priorities % not used
%-------------------------------------------------------------------------------
% DEFINE INPUT AND INITIAL CONDITIONS
if initial_conditions_flag == 1 % use initial data (import from file)
   if exist([initial_conditions_file,'.mat'])==2
      eval(['load ',initial_conditions_file,'.mat;']);
      % check for compatibility with parameters
      state_out_buff=[]; % reset to empty
   else
      disp('Specified file containing initial conditions does not exist ...');
      return
   end
   %
else % zero initial conditions
   %
run_index=0; % iteration index
state_buff=[]; % slab_types; % load slab_types->buff
%groups_buff=determine_slabs_colour(state_buff); % determine stock-codes
%
state_buff1_n=[];
state_anneal=[]; % state of annealing machine empty
state_out_anneal_buff=[];
%
state_cold_roll=[];
state_in_cold_roll_buff=[];
state_out_cold_roll_buff=[];
%
state_in_BWG_buff=[];
state_BWG=[];
state_out_buff=[]; % output buffer initially empty
%
state_highbay=[];
%------------------------
index_state_anneal=zeros(1,no_anneal_machines); % all annealing machines initially empty
time_anneal=-ones(1,no_anneal_machines); % all machines empty waiting to be loaded
% set anneal_flag_empty
anneal_flag_empty=zeros(1,no_anneal_machines);
for i=1:no_anneal_machines
    if index_state_anneal(i)==0
       anneal_flag_empty(i)=1;
    else
       anneal_flag_empty(i)=0;
    end
end
%
   cold_roll_flag_empty=1;
   time_cold_roll=-1;
   %
   BWG_flag_empty=1;
   time_BWG=-1;
   index_list_coils_cr=[];
   %
end
%-------------------------------------------------------------------------------
% ITERATE
%
run_index_local=0; % iteration index
no_coils_loaded=0;
```

```
iterations_per_hour=round(60/simulation_step); % No steps per hour
no_hours=ceil(max_iter*simulation_step/60); % No of hours
current_hour=1; total_crane_moves_per_hour=zeros(1,no_hours);
array_crane_moves_per_hour=zeros(5,no_hours);
anneal_machine_occupancy=zeros(1,no_hours);
bwg_occupancy=zeros(1,no_hours);
cold_roll_machine_occupancy=zeros(1,no_hours);
highbay_occupancy=zeros(1,no_hours);
%
while 1
      %
      run_index=run_index+1;
      run_index_local=run_index_local+1;
      disp(['Current time step: ',int2str(run_index_local)]);
      %
      %---------------------------
      input_from_hotline = deliver_process(run_index_local,no_coils_loaded,state_buff,param,input_file);
      input_from_hotline = check_empty(input_from_hotline);
      no_coils_temp=size(input_from_hotline,1);
      no_coils_loaded=no_coils_loaded+no_coils_temp; % update
      state_buff=update_buff(state_buff,input_from_hotline,[]);
      %---------------------------
      [state_highbay,out_to_buff1_n,out_to_in_cold_roll_buff,out_to_in_BWG_buff,in_from_buff,in_from_anneal_buff,...
      in_from_out_cold_roll_buff,state_out_anneal_buff,state_out_cold_roll_buff,state_buff,index_list_coils_cr, ...
      real_crane_moves]= ...
      update_highbay_new(state_highbay,run_index,state_buff1_n,state_in_cold_roll_buff,state_in_BWG_buff, ...
      state_out_anneal_buff,state_out_cold_roll_buff,state_buff,index_list_coils_cr,param);
      tot_real_crane_moves=sum(real_crane_moves);
      %---------------------------
      [state_in_BWG_buff,output_to_BWG]=update_in_BWG_buff(state_in_BWG_buff,out_to_in_BWG_buff,BWG_flag_empty);
      [state_BWG,time_BWG,out_to_out_buff]=update_bwg(state_BWG,time_BWG,output_to_BWG);
      state_out_buff=update_output_buff(state_out_buff,out_to_out_buff);
      %---------------------------
      [state_buff1_n,output_to_anneal]=update_buff1_n(state_buff1_n,out_to_buff1_n,anneal_flag_empty);
      [state_anneal,index_state_anneal,time_anneal,output_from_anneal]= ...
      update_anneal1_n(state_anneal,index_state_anneal,time_anneal,output_to_anneal,state_out_anneal_buff,param);
      state_out_anneal_buff=update_out_anneal_buff(state_out_anneal_buff,output_from_anneal,[]);
      %---------------------------
      [state_in_cold_roll_buff,output_to_cold_roll]= ...
      update_in_cold_roll_buff(state_in_cold_roll_buff,out_to_in_cold_roll_buff,cold_roll_flag_empty);
      [state_cold_roll,time_cold_roll,output_from_cold_roll]= ...
      update_cold_roll(state_cold_roll,time_cold_roll,output_to_cold_roll,state_out_cold_roll_buff,param);
      state_out_cold_roll_buff=update_out_cold_roll_buff(state_out_cold_roll_buff,output_from_cold_roll,[]);
      %---------------------------
      % Update empty flags
      %
      for i=1:no_anneal_machines
          if time_anneal(i)==-1 % i-th anneal machine is empty
             anneal_flag_empty(i)=1;
          else % i-th annealing machine busy
             anneal_flag_empty(i)=0;
          end
      end
      %
      if time_cold_roll == -1 % cold-roll machine is empty
         cold_roll_flag_empty=1;
      else
          cold_roll_flag_empty=0;
      end

      if isempty(time_BWG) % BWG is empty
         BWG_flag_empty=1;
      else
          BWG_flag_empty=0;
      end
      %---------------------------
      % UPDATE STATISTICS
      %---------------------------
      %
      array_crane_moves_per_hour(:,current_hour)=array_crane_moves_per_hour(:,current_hour)+real_crane_moves';
      total_crane_moves_per_hour(current_hour)=total_crane_moves_per_hour(current_hour)+tot_real_crane_moves;
      anneal_machine_occupancy(current_hour)=anneal_machine_occupancy(current_hour)+size(state_anneal,1);
      bwg_occupancy(current_hour)=bwg_occupancy(current_hour)+size(state_BWG,1);
      cold_roll_machine_occupancy(current_hour)=cold_roll_machine_occupancy(current_hour)+size(state_cold_roll,1);
      highbay_occupancy(current_hour)=highbay_occupancy(current_hour)+size(state_highbay,1);
      %
      if rem(run_index_local,iterations_per_hour)==0 %
         current_hour=current_hour+1;
      end
      %---------------------------
      % Stoping condition 1
      if run_index_local >= max_iter % maximum iteration index exceeded
         break;break;
      end
      %---------------------------
      % Plots
      %---------------------------
      if plot_flag == 1 % plots for complete process
         %
         % Input buffer to HB
         no_slabs_in_buff=size(state_buff,1); % no slubs in input HB buffer
         array_slabs_in_buff_sc=determine_slabs_colour(state_buff);
         %-------------------
         % High-Bay
         no_slabs_highbay=size(state_highbay,1); % no slabs in HB
         array_slabs_highbay_sc=determine_slabs_colour(state_highbay);
         %-------------------
         % Annealing
```

```
no_slabs_buff1_n=size(state_buff1_n,1); % no slabs in input buff to anneal machine
array_slabs_buff1_n_sc=determine_slabs_colour(state_buff1_n);
%
array_slabs_annealed_sc=[];
for i=1:no_anneal_machines
    slabs_i=determine_slabs_annealed(state_anneal,index_state_anneal,i);
    array_slabs_annealed_i_sc=determine_slabs_colour(slabs_i);
    array_slabs_annealed_sc=[array_slabs_annealed_sc ; array_slabs_annealed_i_sc];
end
%
no_slabs_buff1_n=size(state_buff1_n,1); % no slabs being annealed
array_slabs_buff1_n_sc=determine_slabs_colour(state_buff1_n);
%
no_slabs_out_anneal_buff=size(state_out_anneal_buff,1); % no slabs in out-amnneal buff
array_slabs_out_anneal_buff_sc=determine_slabs_colour(state_out_anneal_buff);
%--------------------
% Cold-rolling
no_slabs_in_cold_roll_buff=size(state_in_cold_roll_buff,1); % no slabs in in-cold-roll buffer
array_in_cold_roll_buff_sc=determine_slabs_colour(state_in_cold_roll_buff);
%
no_slabs_cold_roll=size(state_cold_roll,1); % no slabs being cold-rolled
array_slabs_cold_roll_sc=determine_slabs_colour(state_cold_roll);
%
no_slabs_out_cold_roll_buff=size(state_out_cold_roll_buff,1); % no slabs in out cold-roll buffer
array_slabs_out_cold_roll_buff_sc=determine_slabs_colour(state_out_cold_roll_buff);
%----------------------
% BWG
no_slabs_in_BWG_buff=size(state_in_BWG_buff,1); % no slabs in input BWG buffer
array_slabs_in_BWG_buff_sc=determine_slabs_colour(state_in_BWG_buff);
%
no_slabs_BWG=size(state_BWG,1); % no slabs in BWG
array_slabs_BWG_sc=determine_slabs_colour(state_BWG);
%
no_slabs_out_buff=size(state_out_buff,1); % no slabs in output buffer
array_slabs_out_buff_sc=determine_slabs_colour(state_out_buff);
%-------------------------
% ARRAYS FOR PLOTS
%
array_plot=[array_slabs_in_buff_sc; array_slabs_highbay_sc; array_slabs_buff1_n_sc ; array_slabs_annealed_sc;  array_slabs_out_anneal_buff_sc];
array_plot=[array_plot ; array_in_cold_roll_buff_sc ; array_slabs_cold_roll_sc; array_slabs_out_cold_roll_buff_sc];
array_plot=[array_plot ; array_slabs_in_BWG_buff_sc ; array_slabs_BWG_sc ; array_slabs_out_buff_sc];
%
[slabs_to_enter,day_index]=read_from_xls_all(input_file); % full schedule
current_day=floor((run_index_local*simulation_step)/(24*60))+1; % current day
no_coils_day=zeros(1,current_day); % initialize
for i=1:current_day
    no_coils_day_i=(day_index==i);
    no_coils_day(i)=sum(no_coils_day_i);
end
total_no_coils=sum(no_coils_day);
no_coils_to_enter=total_no_coils-no_coils_loaded;
%
row1=0*array_plot(1,:);
row1(1,1)=no_coils_to_enter;
array_plot=[row1 ; array_plot];
%
figure(1)
%
barh(array_plot,'stacked'), colormap(cool)
ylab=['1:SCH ',' 2:IB ',' 3:HB ',' 4-',int2str(5+no_anneal_machines),':ANNEAL '];
ylab=[ylab,int2str(6+no_anneal_machines),'-',int2str(8+no_anneal_machines),':CR '];
ylab=[ylab,int2str(9+no_anneal_machines),'-',int2str(11+no_anneal_machines),':BWG'];
set(gca,'XGrid','on')
xlabel('Number of coils')
ylabel(ylab)
string_time=min2hoursdays(run_index_local*simulation_step);
title(['Iteration no = ',int2str(run_index_local),' ',string_time]);
axis([0 high_bay_capacity 0 12+no_anneal_machines]);
%
pause(pause_time)
%
elseif plot_flag == 2 % HighBay plots
%
    [slabs_to_enter,day_index]=read_from_xls_all(input_file); % full schedule
    current_day=floor((run_index_local*simulation_step)/(24*60))+1; % current day
    no_coils_day=zeros(1,current_day); % initialize
    for i=1:current_day
        no_coils_day_i=(day_index==i);
        no_coils_day(i)=sum(no_coils_day_i);
    end
    total_no_coils=sum(no_coils_day);
    no_coils_to_enter=total_no_coils-no_coils_loaded;
%

    array_plot=determine_slabs_colour1(state_highbay,run_index_local,param);
    row1=[no_coils_to_enter 0];
    array_plot=[row1 ; array_plot];

    figure(1)
%
    barh(array_plot,'stacked'), colormap(cool)
    ylab=['1:Schedule ',' 2:NA ',' 3:Ann ',' 4: CR-1 ',' 5:CR-2 ',' 6:CR-3 ','7:CR-4'];
    set(gca,'XGrid','on')
    xlabel('Number of coils')
    ylabel(ylab)
    string_time=min2hoursdays(run_index_local*simulation_step);
    title(['Iteration no = ',int2str(run_index_local),' ',string_time]);
    axis([0 high_bay_capacity 0 8]);
```

```
        %
            pause(pause_time)
      end
end
%
% STATISTICS
%
max_no_crane_moves_per_hour=max(total_crane_moves_per_hour);
min_no_crane_moves_per_hour=min(total_crane_moves_per_hour);
average_no_crane_moves_per_hour=mean(total_crane_moves_per_hour);
%
anneal_machine_occupancy=(100*anneal_machine_occupancy)/(iterations_per_hour*no_anneal_machines*4); % percentage occupancy
max_anneal_machine_occupancy=max(anneal_machine_occupancy);
min_anneal_machine_occupancy=min(anneal_machine_occupancy);
average_anneal_machine_occupancy=mean(anneal_machine_occupancy);
%
bwg_occupancy=(100*bwg_occupancy)/iterations_per_hour; % percentage occupancy
max_bwg_occupancy=max(bwg_occupancy);
min_bwg_occupancy=min(bwg_occupancy);
average_bwg_occupancy=mean(bwg_occupancy);
%
cold_roll_machine_occupancy=(100*cold_roll_machine_occupancy)/iterations_per_hour; % percentage occupancy
max_cold_roll_machine_occupancy=max(cold_roll_machine_occupancy);
min_cold_roll_machine_occupancy=min(cold_roll_machine_occupancy);
average_cold_roll_machine_occupancy=mean(cold_roll_machine_occupancy);
%
highbay_occupancy=(100*highbay_occupancy)/(high_bay_capacity*iterations_per_hour); % percentage occupancy
max_highbay_occupancy=max(highbay_occupancy);
min_highbay_occupancy=min(highbay_occupancy);
average_highbay_occupancy=mean(highbay_occupancy);
%
% STATISTICS per day
%
no_days=ceil(no_hours/24); hours_in_last_day=rem(no_hours,24);
array_crane_moves_per_day=zeros(5,no_days);
anneal_machine_occupancy_per_day=zeros(1,no_days);
bwg_occupancy_per_day=zeros(1,no_days);
cold_roll_machine_occupancy_per_day=zeros(1,no_days);
highbay_occupancy_per_day=zeros(1,no_days);
%
for i=1:no_days
    if (i < no_days | hours_in_last_day == 0)
       total_crane_moves_per_day(i)=sum(total_crane_moves_per_hour(24*(i-1)+1:24*i));
       anneal_machine_occupancy_per_day(i)=sum(anneal_machine_occupancy(24*(i-1)+1:24*i))/24;
       bwg_occupancy_per_day(i)=sum(bwg_occupancy(24*(i-1)+1:24*i))/24;
       cold_roll_machine_occupancy_per_day(i)=sum(cold_roll_machine_occupancy(24*(i-1)+1:24*i))/24;
       highbay_occupancy_per_day(i)=sum(highbay_occupancy(24*(i-1)+1:24*i))/24;
       array_crane_moves_per_day(:,i)=(sum((array_crane_moves_per_hour(:,24*(i-1)+1:24*i))'))';
    else
       total_crane_moves_per_day(no_days)=...
       sum(total_crane_moves_per_hour(24*(no_days-1)+1:24*(no_days-1)+1+hours_in_last_day));
       anneal_machine_occupancy_per_day(no_days)= ...
       sum(anneal_machine_occupancy(24*(no_days-1)+1:24*(no_days-1)+1+hours_in_last_day))/hours_in_last_day;
       bwg_occupancy_per_day(no_days)= ...
       sum(bwg_occupancy(24*(no_days-1)+1:24*(no_days-1)+1+hours_in_last_day))/hours_in_last_day;
       cold_roll_machine_occupancy_per_day(no_days)= ...
       sum(cold_roll_machine_occupancy(24*(no_days-1)+1:24*(no_days-1)+1+hours_in_last_day))/hours_in_last_day;
       highbay_occupancy_per_day(no_days)= ...
       sum(highbay_occupancy(24*(no_days-1)+1:24*(no_days-1)+1+hours_in_last_day))/hours_in_last_day;
       indd=24*(no_days-1)+1:24*(no_days-1)+1+hours_in_last_day;
       array_crane_moves_per_day(:,no_days)=(sum((array_crane_moves_per_hour(:,indd))'))';
   end
end array_crane_moves_per_day=array_crane_moves_per_day';
%
% STATISTICS plots
%
if stats_plot_flag == 1
   %
   figure(2)
   no_points=size(total_crane_moves_per_hour,2);
   plot(1:no_points,total_crane_moves_per_hour,'-',1:no_points,average_no_crane_moves_per_hour*ones(1,no_points),'--');
   title('Crane moves per hour and average')
   xlabel('No hours');
   grid
%
   figure(3)
   no_points=size(anneal_machine_occupancy,2);
   plot(1:no_points,anneal_machine_occupancy,'-',1:no_points,average_anneal_machine_occupancy*ones(1,no_points),'--');
   title('Percentage ANNEALING furnace occupancy')
   xlabel('No hours');
   grid
%
   figure(4)
   no_points=size(bwg_occupancy,2);
   plot(1:no_points,bwg_occupancy,'-',1:no_points,average_bwg_occupancy*ones(1,no_points),'--');
   title('Percentage BWG Occupancy')
   xlabel('No hours');
   grid
%
   figure(5)
   no_points=size(cold_roll_machine_occupancy,2);
   plot(1:no_points,cold_roll_machine_occupancy,'-',1:no_points,average_cold_roll_machine_occupancy*ones(1,no_points),'--');
   title('Percentage COLD-ROLLING machine occupancy')
   xlabel('No hours');
   grid
%
   figure(6)
   no_points=size(highbay_occupancy,2);
```

```
   plot(1:no_points,highbay_occupancy,'-',1:no_points,average_highbay_occupancy*ones(1,no_points),'--');
   title('Percentage HIGHBAY occupancy')
   xlabel('No hours');
   grid
%
end
%

thr_buff=size(state_out_buff,1);

out_string1=simul_summary1(no_days,param,anneal_machine_occupancy_per_day,bwg_occupancy_per_day,...
cold_roll_machine_occupancy_per_day,highbay_occupancy_per_day,array_crane_moves_per_day,thr_buff);
disp(out_string1);

% stat_data=[max_no_crane_moves_per_hour min_no_crane_moves_per_hour average_no_crane_moves_per_hour;
%            max_anneal_machine_occupancy min_anneal_machine_occupancy average_anneal_machine_occupancy;
%            max_bwg_occupancy min_bwg_occupancy average_bwg_occupancy;
%            max_cold_roll_machine_occupancy min_cold_roll_machine_occupancy average_cold_roll_machine_occupancy;
%            max_highbay_occupancy min_highbay_occupancy average_highbay_occupancy];
% %
% out_string=simul_summary(param,stat_data);
% disp(out_string);
%-------------------------------------------------------------------------
% Save final state conditions
%
var_str1=' run_index state_buff state_highbay state_buff1_n
state_in_cold_roll_buff state_in_BWG_buff state_out_anneal_buff ';
var_str2='state_out_cold_roll_buff state_buff index_list_coils_cr
BWG_flag_empty state_BWG time_BWG state_out_buff ';
var_str3='anneal_flag_empty state_anneal index_state_anneal
time_anneal state_out_anneal_buff cold_roll_flag_empty ';
var_str4='state_cold_roll time_cold_roll'; var_str=[var_str1
var_str2 var_str3 var_str4];
%
flag_log=input('Save final states for future simulation? (y/n) :
','s'); if (flag_log == 'y' | flag_log == 'Y')
    while 1
          file_save=input('Enter file name: ','s');
          if exist([data_dir,'\',file_save,'.mat'])==2
             flag_overwrite=input('File exists - overwrite? (y/n) : ','s');
             if (flag_overwrite == 'y' | flag_overwrite == 'Y')
                 eval(['save ',data_dir,'\',file_save,' ',var_str,';']);
                 return;
             end
          else
             eval(['save ',data_dir,'\',file_save,' ',var_str,';']);
             return;
          end
    end
end
%
%------------------------------------------- end run_process_ic.m ---------------------------------------------
```

▶ Function set_time_stamp:

```
  function out_slabs=set_time_stamp(slabs,time_index)
% function out_slabs=set_time_stamp(slabs,time_index)
% Set current time-stamp for slabs entering HighBay
%
  slabs=check_empty(slabs);
  if isempty(slabs)
     out_slabs=[];
     return
  end
%
  no_slabs=size(slabs,1); % no slabs
%
  time_index=round(time_index);
  out_slabs=[];
  for i=1:no_slabs
      current_slab=slabs(i); % get current slab
      timer_current_slab=current_slab.timer; % get flags field
      timer_current_slab=time_index; % set to current time index when entering HB
      current_slab = setfield(current_slab,'timer',timer_current_slab);
      out_slabs=[out_slabs ; current_slab];
  end
%
%----------------------------- end of set_time_stamp.m -----------------------------
```

▶ Function simul_summary:

```
function out_string=simul_summary(param,stat_data);
% function out_string=simul_summary(param,stat_data);
% Produces string summarizing simulation parameters and results
%
now=DATESTR(clock,'dd-mmm-yyyy HH:MM:SS');
sfe='-------------------------------------------------------------------------';
s1=['SIMULATION RUN at: ',now,' with following parameters:']; s2='
'; s3=['Input file: ',param.input_file]; if
param.initial_conditions_flag == 1
   s4=['Initial conditions file :',param.initial_conditions_file];
else
   s4='';
end s5=['Simulation time: ',int2str(param.simulation_time),'hours'];
s6=['Simulation step:',int2str(param.simulation_step),' min'];
s7=['No of annealing machines: ',int2str(param.no_anneal_machines)];
```

```
s8=['Load rate from hotline: 1 coil every
',int2str((param.load_rate)*(param.simulation_step)),' min'];
s9=['Maximum no of crane moves: ',int2str(param.crane_moves),' every
',int2str(param.simulation_step),' min'];
%
%priorities=[1 2 3]; % priorities order= cold rolling(in-out),in to HB, out to BWG
flags_pr=param.priorities; pr_vec=['Cold-Roll i/o  '
        'Out to BWG     '
        'In to HighBay  '];
pr1=flags_pr(1); pr2=flags_pr(2); pr3=flags_pr(3);
%
s10=['Priorities:   1: ',pr_vec(pr1,:),' 2:',pr_vec(pr2,:),'
3:',pr_vec(pr3,:)];
%
s11=s2; s12=['Summary of Statistical results'];
s13=['------------------------------'];
s14=['Crane moves per hour          MAX:',pad_blanks(num2str(stat_data(1,1),'%4.1f'),6),'    MIN:',pad_blanks(num2str(stat_data(1,2),'%4.1f'),6),'
s15=['Anneal machine occupancy (%)  MAX:',pad_blanks(num2str(stat_data(2,1),'%4.1f'),6),'    MIN:',pad_blanks(num2str(stat_data(2,2),'%4.1f'),6),'
s16=['BWG occupancy (%)             MAX:',pad_blanks(num2str(stat_data(3,1),'%4.1f'),6),'    MIN:',pad_blanks(num2str(stat_data(3,2),'%4.1f'),6),'
s17=['CR machine occupancy (%)      MAX:',pad_blanks(num2str(stat_data(4,1),'%4.1f'),6),'    MIN:',pad_blanks(num2str(stat_data(4,2),'%4.1f'),6),'
s18=['HighBay occupancy (%)         MAX:',pad_blanks(num2str(stat_data(5,1),'%4.1f'),6),'    MIN:',pad_blanks(num2str(stat_data(5,2),'%4.1f'),6),'
%
out_string=strvcat(sfe,s1,s2,s3,s4,s5,s6,s7,s8,s9,s10,s11,s12,s13,s14,s15,s16,s17,s18,sfe);
%
%------------------------------ end of simul_summary.m --------------------------------
```

▶ Function simul_summary1:

```
function
out_string=simul_summary1(no_days,param,anneal_machine_occupancy_per_day,...
          bwg_occupancy_per_day,cold_roll_machine_occupancy_per_day,highbay_occupancy_per_day,crane_moves_per_day,thr_buff)
% function out_string=simul_summary1(no_days,param,anneal_machine_occupancy_per_day, ...
%          bwg_occupancy_per_day,cold_roll_machine_occupancy_per_day,highbay_occupancy_per_day,crane_moves_per_day,thr_buff)
%
% Produces string summarizing simulation parameters and simulation
% statistics
%
%
% max_no_crane_moves_per_day=max(total_crane_moves_per_day);
% min_no_crane_moves_per_day=min(total_crane_moves_per_day);
% average_no_crane_moves_per_day=mean(total_crane_moves_per_day);
% %
% max_anneal_machine_occupancy_per_day=max(anneal_machine_occupancy_per_day);
% min_anneal_machine_occupancy_per_day=min(anneal_machine_occupancy_per_day);
average_anneal_machine_occupancy_per_day=mean(anneal_machine_occupancy_per_day);
% %
% max_bwg_occupancy_per_day=max(bwg_occupancy_per_day);
% min_bwg_occupancy_per_day=min(bwg_occupancy_per_day);
average_bwg_occupancy_per_day=mean(bwg_occupancy_per_day);
% %
% max_cold_roll_machine_occupancy_per_day=max(cold_roll_machine_occupancy_per_day);
% min_cold_roll_machine_occupancy_per_day=min(cold_roll_machine_occupancy_per_day);
average_cold_roll_machine_occupancy_per_day=mean(cold_roll_machine_occupancy_per_day);
% %
% max_highbay_occupancy_per_day=max(highbay_occupancy_per_day);
% min_highbay_occupancy_per_day=min(highbay_occupancy_per_day);
average_highbay_occupancy_per_day=mean(highbay_occupancy_per_day);
%
if no_days > 1
    total_array_crane_moves_per_day=sum(crane_moves_per_day);
else
    total_array_crane_moves_per_day=crane_moves_per_day;
end

%-------------------------------------------------------------
%
now=DATESTR(clock,'dd-mmm-yyyy HH:MM:SS');
sfe='-------------------------------------------------------------------------------------------';
s1=['SIMULATION RUN at: ',now,' with the following parameters:'];
s2='  '; s3=['Input file: ',param.input_file]; if
param.initial_conditions_flag == 1
   s4=['Initial conditions file :',param.initial_conditions_file];
else
   s4='';
end s5=['Simulation time: ',int2str(param.simulation_time),'
hours']; s6=['Simulation step: ',int2str(param.simulation_step),'
min']; s7=['No of annealing machines:
',int2str(param.no_anneal_machines)]; s8=['Load rate from hotline: 1
coil every ',int2str((param.load_rate)*(param.simulation_step)),'
min']; s9=['Maximum no of crane moves:
',int2str(param.crane_moves),' every
',int2str(param.simulation_step),' min'];
%
%priorities=[1 2 3]; % priorities order= cold rolling(in-out),in to HB, out to BWG
flags_pr=param.priorities; x1=find(flags_pr==1);
x2=find(flags_pr==2); x3=find(flags_pr==3);

pr_vec=['Cold-Roll i/o  '
        'Out to BWG     '
        'In to HighBay  '];
pr1=pr_vec(x1,:); pr2=pr_vec(x2,:); pr3=pr_vec(x3,:);
%

s10=['Priorities:   1:',pr1,' 2:',pr2,' 3:',pr3];
%s10=['Priorities:   1: ',pr_vec(pr1,:),' 2:',pr_vec(pr2,:),' 3:',pr_vec(pr3,:)];
%
s11=s2;
s12=['Summary of Statistical results: USAGE (%)'];
```

```
s13=['----------------------------------------'];
%
s_temp_1='                    ANNEAL      TSM       BWG     HIGHBAY';
s_temp_2='                    ------      ---       ---     -------';
s_temp=strvcat(s_temp_1,s_temp_2);
%
for i=1:no_days s_temp_i=[...
'Day ',pad_blanks(num2str(i,'%3.0f'),5), '    :     ',pad_blanks(num2str(anneal_machine_occupancy_per_day(i),'%4.1f'),6),'    ', ...
pad_blanks(num2str(cold_roll_machine_occupancy_per_day(i),'%4.1f'),6),'    ', ...
pad_blanks(num2str(bwg_occupancy_per_day(i),'%4.1f'),6),'    ', ...
pad_blanks(num2str(highbay_occupancy_per_day(i),'%4.1f'),6)];
s_temp=strvcat(s_temp,s_temp_i); end
s_temp_1=['TOTAL       :     ',pad_blanks(num2str(average_anneal_machine_occupancy_per_day,'%4.1f'),6),'    ',...
pad_blanks(num2str(average_cold_roll_machine_occupancy_per_day,'%4.1f'),6),'    ',...
pad_blanks(num2str(average_bwg_occupancy_per_day,'%4.1f'),6),'    ', ...
pad_blanks(num2str(average_highbay_occupancy_per_day,'%4.1f'),6)];
s_temp_usage=strvcat(s_temp,s_temp_1);
%
s_blank='     ';
%

s14=s2; s15=['Summary of Statistical results: CRANE MOVES AT EACH
LOCATION'];
s16=['------------------------------------------------------------'];
```

```
s_temp_1='                  COIL-ENTRY    ANNEALING      TSM-IN
TSM-OUT        BWG'; s_temp_2='                  ----------
---------      ------       -------        ---';
s_temp=strvcat(s_temp_1,s_temp_2);
%
for i=1:no_days s_temp_i=[...
'Day ',pad_blanks(num2str(i,'%3.0f'),5), '    :     ',pad_blanks(num2str(crane_moves_per_day(i,1),'%5.0f'),6),'    ', ...
pad_blanks(num2str(crane_moves_per_day(i,2),'%5.0f'),9),'    ', ...
pad_blanks(num2str(crane_moves_per_day(i,3),'%5.0f'),9),'    ', ...
pad_blanks(num2str(crane_moves_per_day(i,4),'%5.0f'),9),'    ', ...
pad_blanks(num2str(crane_moves_per_day(i,5),'%5.0f'),9)];
s_temp=strvcat(s_temp,s_temp_i); end
%
s_temp_1=['TOTAL       :     ',pad_blanks(num2str(total_array_crane_moves_per_day(1),'%5.0f'),6),'    ',...
pad_blanks(num2str(total_array_crane_moves_per_day(2),'%5.0f'),9),'    ',...
pad_blanks(num2str(total_array_crane_moves_per_day(3),'%5.0f'),9),'    ', ...
pad_blanks(num2str(total_array_crane_moves_per_day(4),'%5.0f'),9),'    ',...
pad_blanks(num2str(total_array_crane_moves_per_day(5),'%5.0f'),9)];
%
s_temp_crane=strvcat(s_temp,s_temp_1);
%
sthr=['Throughput: ',int2str(thr_buff),' coils in
',int2str(no_days),' days'];
out_string=strvcat(sfe,s1,s2,s3,s4,s5,s6,s7,s8,s9,s10,s11,s12,s13,s_temp_usage,s_blank,s_blank,s14,s15,s16,s_temp_crane,sfe,sthr);
%
%------------------------------ end of simul_summary.m ---------------------------------
```

▶ Function Simulation_gui:

```
function varargout = Simulation_gui(varargin)

gui_Singleton = 1; gui_State = struct('gui_Name',       mfilename,
...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @Simulation_gui_OpeningFcn, ...
                   'gui_OutputFcn',  @Simulation_gui_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin & isstr(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end

% --- Executes just before Simulation_gui is made visible.
function Simulation_gui_OpeningFcn(hObject, eventdata, handles,
varargin)

handles.anneal=3; handles.cooling=72; handles.rate=20;
handles.crane=60;

handles.prior_anneal=1;  % This option is not currently in use
handles.prior_entry=2; handles.prior_tsm=3; handles.prior_bwg=4;

handles.input='schedule1'; handles.initial_file='initial_data';
handles.initial_flag=0; handles.popup=1; handles.plot=2;
handles.simulation=30;  %Initial simulation time 30 days (or 8640 iterations)

% Choose default command line output for Simulation_gui
handles.output = hObject;

% Update handles structure
guidata(hObject, handles);

% UIWAIT makes Simulation_gui wait for user response (see UIRESUME)
% uiwait(handles.figure1);

% --- Outputs from this function are returned to the command line.
```

```
function varargout = Simulation_gui_OutputFcn(hObject, eventdata,
handles)

% Get default command line output from handles structure
varargout{1} = handles.output;

% --- Executes during object creation, after setting all properties.
function edit_anneal_CreateFcn(hObject, eventdata, handles)

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc
    set(hObject,'BackgroundColor','white');
else
    set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end

function edit_anneal_Callback(hObject, eventdata, handles)

handles.anneal=str2double(get(hObject,'String')); guidata(hObject,
handles);

% --- Executes during object creation, after setting all properties.
function edit_cooling_CreateFcn(hObject, eventdata, handles)

if ispc
    set(hObject,'BackgroundColor','white');
else
    set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end

function edit_cooling_Callback(hObject, eventdata, handles)

handles.cooling=str2double(get(hObject,'String')); guidata(hObject,
handles);

if ispc
    set(hObject,'BackgroundColor','white');
else
    set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end

function edit_entry_Callback(hObject, eventdata, handles)

handles.rate=str2double(get(hObject,'String')); guidata(hObject,
handles);

% --- Executes during object creation, after setting all properties.
function edit_crane_CreateFcn(hObject, eventdata, handles)

if ispc
    set(hObject,'BackgroundColor','white');
else
    set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end

function edit_crane_Callback(hObject, eventdata, handles)

handles.crane=str2double(get(hObject,'String')); guidata(hObject,
handles);

% --- Executes during object creation, after setting all properties.
function edit_prior_anneal_CreateFcn(hObject, eventdata, handles)

if ispc
    set(hObject,'BackgroundColor','white');
else
    set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end

function edit_prior_anneal_Callback(hObject, eventdata, handles)

% --- Executes during object creation, after setting all properties.
function edit_prior_entry_CreateFcn(hObject, eventdata, handles)

if ispc
    set(hObject,'BackgroundColor','white');
else
    set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end

function edit_prior_entry_Callback(hObject, eventdata, handles)

handles.prior_entry=str2double(get(hObject,'String'));
guidata(hObject, handles);

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc
    set(hObject,'BackgroundColor','white');
else
    set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end

function edit_prior_tsm_Callback(hObject, eventdata, handles)

handles.prior_tsm=str2double(get(hObject,'String'));
```

```
guidata(hObject, handles);

% --- Executes during object creation, after setting all properties.
function edit_prior_bwg_CreateFcn(hObject, eventdata, handles)

if ispc
    set(hObject,'BackgroundColor','white');
else
    set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end

function edit_prior_bwg_Callback(hObject, eventdata, handles)

handles.prior_bwg=str2double(get(hObject,'String'));
guidata(hObject, handles);


function edit_file_name_CreateFcn(hObject, eventdata, handles)

if ispc
    set(hObject,'BackgroundColor','white');
else
    set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end

function edit_file_name_Callback(hObject, eventdata, handles)

handles.input=get(hObject,'String'); guidata(hObject, handles);
[handles.input,path] = uigetfile('*.mat; *.xls', 'Choose your
files'); guidata(hObject, handles);


function edit_plot_selection_CreateFcn(hObject, eventdata, handles)

if ispc
    set(hObject,'BackgroundColor','white');
else
    set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end

function edit_plot_selection_Callback(hObject, eventdata, handles)

handles.plot=str2double(get(hObject,'String')); guidata(hObject,
handles);


function buttoncancel_Callback(hObject, eventdata, handles)

delete(handles.figure1)

% --- Executes on button press in button_run.
function button_run_Callback(hObject, eventdata, handles)

param=default_parameters;

%--UPDATE ANNEAL PARAMETERS----------------------------------
%
param.no_anneal_machines=handles.anneal;
%-----------------------------------------------------------

%--UPDATE COOLING PARAMETERS---------------------------------
%
handles.cooling=floor(handles.cooling*12);
param.cool_time_after_coldroll=handles.cooling;
%-----------------------------------------------------------

%--UPDATE CRANE PARAMETERS-----------------------------------
%
handles.crane=max(1,floor(handles.crane/12));
param.crane_moves=handles.crane;
%-----------------------------------------------------------

%--UPDATE ENTRY RATE PARAMETERS------------------------------
%
handles.rate=max(1,floor(handles.rate/5));
param.load_rate=handles.rate;
%-----------------------------------------------------------

%--UPDATE PRIORITIES PARAMETERS------------------------------
%
handles.prior_tsm=(handles.prior_tsm-1);
handles.prior_bwg=(handles.prior_bwg-1);
handles.prior_entry=(handles.prior_entry-1);

param.priorities(1)=handles.prior_tsm;
param.priorities(2)=handles.prior_bwg;
param.priorities(3)=handles.prior_entry;
%-----------------------------------------------------------

%--UPDATE INPUT EXCEL FILE CHOICE----------------------------
%
param.input_file=handles.input;
%-----------------------------------------------------------

%--UPDATE INITIAL CONDITIONS PARAMETERS----------------------
%
param.initial_conditions_file=handles.initial_file;
param.initial_conditions_flag=handles.initial_flag;
```

```
%-----------------------------------------------------------

%--UPDATE PLOTTING PARAMETERS---------------------------------
%
param.plot_flag=handles.plot;
%-----------------------------------------------------------

%--UPDATE SIMULATION PARAMETERS-------------------------------
%
handles.simulation=handles.simulation*24;
param.simulation_time=handles.simulation;
%-----------------------------------------------------------
run_process_ic

% --- Executes during object creation, after setting all properties.
function edit_initial_CreateFcn(hObject, eventdata, handles)

if ispc
    set(hObject,'BackgroundColor','white');
else
    set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end

function edit_initial_Callback(hObject, eventdata, handles)

function edit_simulation_days_CreateFcn(hObject, eventdata, handles)

if ispc
    set(hObject,'BackgroundColor','white');
else
    set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end function edit_simulation_days_Callback(hObject, eventdata,
handles)

handles.simulation=str2double(get(hObject,'String'));
guidata(hObject, handles);

function popupmenu_Callback(hObject, eventdata, handles)

handles.popup=get(hObject,'Value'); guidata(hObject, handles);

if handles.popup==1
     handles.initial_file=handles.initial_file;
     handles.initial_flag=0;
     guidata(hObject, handles);
elseif handles.popup==2
     [handles.initial_file,path] = uigetfile('*.mat', 'Choose your file with initial conditions');
     if handles.initial_file==0
        handles.initial_file='initial_data';
        guidata(hObject, handles);
     else
     handles.initial_file(size(handles.initial_file,2)-3:size(handles.initial_file,2))=[];
     handles.initial_flag=1;
     guidata(hObject, handles);
     end
end
% --- Executes during object creation, after setting all properties.
function popupmenu_CreateFcn(hObject, eventdata, handles)

if ispc
    set(hObject,'BackgroundColor','white');
else
    set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end

function pushbutton_Callback(hObject, eventdata, handles)

[handles.input,path] = uigetfile('*.xls', 'Choose your input
schedule file'); if handles.input==0
   handles.input='schedule1';
   guidata(hObject, handles);
else
handles.input(size(handles.input,2)-3:size(handles.input,2))=[];
guidata(hObject, handles); end

function figure1_CreateFcn(hObject, eventdata, handles)
%------------------------------ end of Simulation_gui.m --------------------------------
```

▶ Function specify_slabs:

```
  function slab_types_out=specify_slabs(no_slabs_array,slab_types);
% function slab_types_out=specify_slabs(no_slabs_array,slab_types);
% FUNCTION SPECIFY_SLABS
% Creates structure of required slabs according to column-vector of integers
% no_slabs_array and array of structures slab_types
%
 no_orders=size(no_slabs_array,1);
 %
 if size(no_slabs_array,2) ~= 1
    slab_types_out=[];
    return
 end
%
 for i=1:no_orders
     if no_slabs_array(i) ~= round(no_slabs_array(i)) | no_slabs_array(i) <= 0
        slab_types_out=[];
        return
     end
```

```
 end
%
  slab_types_out=[];
  for i=1:no_orders
      for j=1:no_slabs_array(i)
          slab_types_out=[slab_types_out ; slab_types(i,:)];
      end
  end
%
%----------------------------- end of specify_slabs.m --------------------
```

▶ Function split_stock_codes:

```
  function array_out=split_stock_codes(slabs_in,stock_codes)
% function array_out=split_stock_codes(slabs_in,stock_codes)
%
% Given an list of slabs_in and an array of 1xm stock_codes, function
% produces array_out of size 1xm whose m-th element is the number of
% slabs in slabs_in with stock_code(i)
%

no_stock_codes=size(stock_codes,2); if no_stock_codes==0;
   array_out=[];
   return
end
%
array_out=zeros(1,no_stock_codes); no_slabs=size(slabs_in,1);
%
if no_slabs == 0 | size(slabs_in,2)==0
   no_slabs=0;
   return
end
%
for i=1:no_slabs
    current_slab=slabs_in(i);
    current_stock_code=current_slab.stock_code;
    for j=1:no_stock_codes
        if current_stock_code==stock_codes(j)
           array_out(j)=array_out(j)+1;
       end
   end
end
%---------------------- end of split_stock_codes.m ----------------------
```

▶ Function update_anneal1_n:

```
  function [new_state,index_new_state,new_time,output_from_anneal]= ...
  update_anneal1_n(old_state,index_old_state,old_time,input1,state_out_anneal_buff,param);
% function [new_state,index_new_state,new_time,output_from_anneal]= ...
% update_anneal1_n(old_state,index_old_state,old_time,input1,state_out_anneal_buff,param);
%
% INPUTS:
%
%       old_state:              list of slabs being annealed or empty matrix; these are
%                               listed vertically and the state-dimension of the i-th annealing machine
%                               is specified by index_old_state(i)
%       index_old_state:        no of slabs in i-th annealing machine for old_state
%       old_time:               n-vector (where n=no of parallel annealing machines) whose i-th entry
%                               specifies time-step since beginning of annealing in i-th machine if i-th machine
%                               is busy; if i-th machine is empty then old_time(i)=-1
%       input1:                 list of slabs from buffer buff1_n output (could be empty)
%       state_out_anneal_buff   current state of out_anneal_buffer (list of slabs or empty matric). Used to
%                               calculate output so that no overflow occurs.
%
% OUTPUTS:
%
%       new_state:              list of slabs inside anneal-machine at end of time-interval; these are listed
%                               vertically and the state-dimension of the i-th machine is specified by index_new_state(i)
%       index_new_state:        no of slabs in ith annealing machine for new_state
%       new_time:               n-dimensional vector; For the i-th machine:
%                                   If busy->busy: new_time(i)=old_time(i)+1
%                                   If finished annealing but not loaded output (for not causing overflow to out_anneal_buff)
%                                      new_time(i)=old_time(i) (time frozen)
%                                   If loaded from empty:  new_time(i) reset to 1
%                                   If empty at end of interval: new_time(i)=-1
%       output_from_anneal:    list of slabs: contribution from i-th machine is empty matrix if annealing is still in progress or
%                              annealing in i-th machine has finished but output will cause overflow to out_anneal_buff; else
%                              output is stacked to output_from_anneal array in machine order.
%

%-------------------------------------------------------------------------------------------
% Get parameters
%
  out_anneal_buff_capacity=param.out_anneal_buff_capacity;
%-------------------------------------------------------------------------------------------
  n=max(size(index_old_state));
  %
  if size(old_state,1) ~= sum(index_old_state)
     new_state=[];
     index_new_state=[];
     new_time=[];
     output_from_anneal=[];
     disp('Error in update_anneal_n.m ...')
     return
  end
  %
```

```
state_out_anneal_buff=check_empty(state_out_anneal_buff);
no_slabs_out_anneal_buff=size(state_out_anneal_buff,1);
%
new_state=[]; % initialize
index_new_state=zeros(1,n); % initialize
new_time=zeros(1,n); % initialize
output_from_anneal=[]; % initialize
current_input_list=input1; % initialize
out_slabs_count=0; % initialize
%
if isempty(input1); % no input from buffer
   for i=1:n % for each machine
       if index_old_state(i) ~= 0 % machine busy
          %
          if i==1
             old_state_i=old_state(1:index_old_state(1));
          else
             old_state_i=old_state(sum(index_old_state(1:i-1))+1:sum(index_old_state(1:i-1))+index_old_state(i));
          end % here old_state_i is known
          %old_state_i=determine_slabs_annealed(old_state,index_old_state,i);
          %
          top_slab_i=old_state_i(1,:);
          total_time_for_anneal_i=obtain_anneal_time(top_slab_i);
   %
          if old_time(i)+1 < total_time_for_anneal_i % annealing not finished at end of period
             new_state=[new_state ; old_state_i];
             new_time(i)=old_time(i)+1; % increment clock
             index_new_state(i)=index_old_state(i);
          else % process ends at end of time-interval
             if out_slabs_count + index_old_state(i) + no_slabs_out_anneal_buff <= out_anneal_buff_capacity % no overflow
                new_time(i)=-1;
                index_new_state(i)=0;
                output_from_anneal=[output_from_anneal ; old_state_i]; % stamp before returning
                out_slabs_count=out_slabs_count+index_old_state(i); % update count of out slabs
             else % ouput will cause overflow to out_anneal_buff
                new_state=[new_state ; old_state_i];
                index_new_state(i)=index_old_state(i);
                new_time(i)=old_time(i); % freeze time - try to output next time round
             end
          end
       else % machine is empty
          new_time(i)=-1;
       end % if
    end %for
else % some input waiting to be loaded
    for i=1:n % for each machine
        if index_old_state(i) ~= 0 % machine busy
           %
           if i==1
              old_state_i=old_state(1:index_old_state(1));
           else
              old_state_i=old_state(sum(index_old_state(1:i-1))+1:sum(index_old_state(1:i-1))+index_old_state(i));
           end % here old_state_i is known
           %old_state_i=determine_slabs_annealed(old_state,index_old_state,i);
           %
           % No input-> state transition at end of interval, possible state->output transition
           top_slab_i=old_state_i(1,:);
           total_time_for_anneal_i=obtain_anneal_time(top_slab_i);
           %
           if old_time(i)+1 < total_time_for_anneal_i % annealing not finished at end of period
              new_state=[new_state ; old_state_i];
              index_new_state(i)=index_old_state(i);
              new_time(i)=old_time(i)+1; % increment clock
           else % process ends at end of time-interval
              if out_slabs_count+index_old_state(i)+no_slabs_out_anneal_buff <= out_anneal_buff_capacity % no overflow
                 new_time(i)=-1;
                 index_new_state(i)=0;
                 output_from_anneal=[output_from_anneal ; old_state_i]; % stamp before returning
                 out_slabs_count=out_slabs_count+index_old_state(i); % update count of out slabs
              else % ouput will cause overflow to out_anneal_buff
                 new_state=[new_state ; old_state_i];
                 index_new_state(i)=index_old_state(i);
                 new_time(i)=old_time(i); % freeze time - try to output annealed slabs next time round
             end
           end
           %
    else % i-th machine is empty
           %
           if i==1
              old_state_i=old_state(1:index_old_state(1));
           else
              old_state_i=old_state(sum(index_old_state(1:i-1))+1:sum(index_old_state(1:i-1))+index_old_state(i));
           end % here old_state_i is known
           %old_state_i=determine_slabs_annealed(old_state,index_old_state,i);
           %
           % No state->output transition possible, check for input->state transition
           %
           [in_slabs,index_slabs_in]=determine_slabs_for_anneal(current_input_list); % check for compatibility
           current_input_list(index_slabs_in)=[]; % eliminate imported slabs to i-th machine
           current_input_list=check_empty(current_input_list);
           new_state=[new_state ; in_slabs]; % update new global state
           index_new_state(i)=size(in_slabs,1); % update with number of slabs
           if isempty(in_slabs)
              new_time(i)=-1; % keep time at -1
           else
              new_time(i)=1; % start clock
           end
    end %if
```

```
    end %for
 end %if
%
 output_from_anneal=anneal_stamp(output_from_anneal); % update anneal stamp
%
%---------------------------------------- end update_anneal1_n.m ---------------------------------------
```

▶ Function update_anneal1_n:

```
  function [new_state,index_new_state,new_time,output_from_anneal]= ...
  update_anneal1_n(old_state,index_old_state,old_time,input1,state_out_anneal_buff,param);
% function [new_state,index_new_state,new_time,output_from_anneal]= ...
% update_anneal1_n(old_state,index_old_state,old_time,input1,state_out_anneal_buff,param);
%
% INPUTS:
%
%       old_state:              list of slabs being annealed or empty matrix; these are
%                               listed vertically and the state-dimension of the i-th annealing machine
%                               is specified by index_old_state(i)
%       index_old_state:        no of slabs in i-th annealing machine for old_state
%       old_time:               n-vector (where n=no of parallel annealing machines) whose i-th entry
%                               specifies time-step since beginning of annealing in i-th machine if i-th machine
%                               is busy; if i-th machine is empty then old_time(i)=-1
%       input1:                 list of slabs from buffer buff1_n output (could be empty)
%       state_out_anneal_buff   current state of out_anneal_buffer (list of slabs or empty matric). Used to
%                               calculate output so that no overflow occurs.
%
% OUTPUTS:
%
%       new_state:              list of slabs inside anneal-machine at end of time-interval; these are listed
%                               vertically and the state-dimension of the i-th machine is specified by index_new_state(i)
%       index_new_state:        no of slabs in ith annealing machine for new_state
%       new_time:               n-dimensional vector; For the i-th machine:
%                                   If busy->busy: new_time(i)=old_time(i)+1
%                                   If finished annealing but not loaded output (for not causing overflow to out_anneal_buff)
%                                      new_time(i)=old_time(i) (time frozen)
%                                   If loaded from empty:  new_time(i) reset to 1
%                                   If empty at end of interval: new_time(i)=-1
%       output_from_anneal:     list of slabs: contribution from i-th machine is empty matrix if annealing is still in progress or
%                               annealing in i-th machine has finished but output will cause overflow to out_anneal_buff; else
%                               output is stacked to output_from_anneal array in machine order.
%

%-------------------------------------------------------------------------------------------------
% Get parameters
%
  out_anneal_buff_capacity=param.out_anneal_buff_capacity;
%-------------------------------------------------------------------------------------------------
  n=max(size(index_old_state));
  %
  if size(old_state,1) ~= sum(index_old_state)
     new_state=[];
     index_new_state=[];
     new_time=[];
     output_from_anneal=[];
     disp('Error in update_anneal_n.m ...')
     return
end
%
state_out_anneal_buff=check_empty(state_out_anneal_buff);
no_slabs_out_anneal_buff=size(state_out_anneal_buff,1);
%
new_state=[]; % initialize
index_new_state=zeros(1,n); % initialize
new_time=zeros(1,n); % initialize
output_from_anneal=[]; % initialize
current_input_list=input1; % initialize
out_slabs_count=0; % initialize
%
if isempty(input1); % no input from buffer
   for i=1:n % for each machine
       if index_old_state(i) ~= 0 % machine busy
          %
          if i==1
             old_state_i=old_state(1:index_old_state(1));
          else
             old_state_i=old_state(sum(index_old_state(1:i-1))+1:sum(index_old_state(1:i-1))+index_old_state(i));
          end % here old_state_i is known
          %old_state_i=determine_slabs_annealed(old_state,index_old_state,i);
          %
          top_slab_i=old_state_i(1,:);
          total_time_for_anneal_i=obtain_anneal_time(top_slab_i);
   %
          if old_time(i)+1 < total_time_for_anneal_i % annealing not finished at end of period
             new_state=[new_state ; old_state_i];
             new_time(i)=old_time(i)+1; % increment clock
             index_new_state(i)=index_old_state(i);
          else % process ends at end of time-interval
             if out_slabs_count + index_old_state(i) + no_slabs_out_anneal_buff <= out_anneal_buff_capacity % no overflow
                new_time(i)=-1;
                index_new_state(i)=0;
                output_from_anneal=[output_from_anneal ; old_state_i]; % stamp before returning
                out_slabs_count=out_slabs_count+index_old_state(i); % update count of out slabs
             else % ouput will cause overflow to out_anneal_buff
                new_state=[new_state ; old_state_i];
                index_new_state(i)=index_old_state(i);
                new_time(i)=old_time(i); % freeze time - try to output next time round
             end
```

```
          end
       else % machine is empty
          new_time(i)=-1;
       end % if
    end %for
else % some input waiting to be loaded
    for i=1:n % for each machine
        if index_old_state(i) ~= 0 % machine busy
           %
           if i==1
              old_state_i=old_state(1:index_old_state(1));
           else
              old_state_i=old_state(sum(index_old_state(1:i-1))+1:sum(index_old_state(1:i-1))+index_old_state(i));
           end % here old_state_i is known
           %old_state_i=determine_slabs_annealed(old_state,index_old_state,i);
           %
           % No input-> state transition at end of interval, possible state->output transition
           top_slab_i=old_state_i(1,:);
           total_time_for_anneal_i=obtain_anneal_time(top_slab_i);
           %
           if old_time(i)+1 < total_time_for_anneal_i % annealing not finished at end of period
              new_state=[new_state ; old_state_i];
              index_new_state(i)=index_old_state(i);
              new_time(i)=old_time(i)+1; % increment clock
           else % process ends at end of time-interval
              if out_slabs_count+index_old_state(i)+no_slabs_out_anneal_buff <= out_anneal_buff_capacity % no overflow
                 new_time(i)=-1;
                 index_new_state(i)=0;
                 output_from_anneal=[output_from_anneal ; old_state_i]; % stamp before returning
                 out_slabs_count=out_slabs_count+index_old_state(i); % update count of out slabs
              else % ouput will cause overflow to out_anneal_buff
                 new_state=[new_state ; old_state_i];
                 index_new_state(i)=index_old_state(i);
                 new_time(i)=old_time(i); % freeze time - try to output annealed slabs next time round
             end
           end
           %
        else % i-th machine is empty
           %
           if i==1
              old_state_i=old_state(1:index_old_state(1));
           else
              old_state_i=old_state(sum(index_old_state(1:i-1))+1:sum(index_old_state(1:i-1))+index_old_state(i));
           end % here old_state_i is known
           %old_state_i=determine_slabs_annealed(old_state,index_old_state,i);
           %
           % No state->output transition possible, check for input->state transition
           %
           [in_slabs,index_slabs_in]=determine_slabs_for_anneal(current_input_list); % check for compatibility
           current_input_list(index_slabs_in)=[]; % eliminate imported slabs to i-th machine
           current_input_list=check_empty(current_input_list);
           new_state=[new_state ; in_slabs]; % update new global state
           index_new_state(i)=size(in_slabs,1); % update with number of slabs
           if isempty(in_slabs)
              new_time(i)=-1; % keep time at -1
           else
              new_time(i)=1; % start clock
           end
        end %if
    end %for
 end %if
 %
 output_from_anneal=anneal_stamp(output_from_anneal); % update anneal stamp
 %
 %---------------------------------------- end update_anneal1_n.m ----------------------------------------
```

▶ Function update_anneal1_n:

```
  function [new_state,index_new_state,new_time,output_from_anneal]= ...
  update_anneal1_n(old_state,index_old_state,old_time,input1,state_out_anneal_buff,param);
% function [new_state,index_new_state,new_time,output_from_anneal]= ...
% update_anneal1_n(old_state,index_old_state,old_time,input1,state_out_anneal_buff,param);
%
% INPUTS:
%
%       old_state:              list of slabs being annealed or empty matrix; these are
%                               listed vertically and the state-dimension of the i-th annealing machine
%                               is specified by index_old_state(i)
%       index_old_state:        no of slabs in i-th annealing machine for old_state
%       old_time:               n-vector (where n=no of parallel annealing machines) whose i-th entry
%                               specifies time-step since beginning of annealing in i-th machine if i-th machine
%                               is busy; if i-th machine is empty then old_time(i)=-1
%       input1:                 list of slabs from buffer buff1_n output (could be empty)
%       state_out_anneal_buff   current state of out_anneal_buffer (list of slabs or empty matric). Used to
%                               calculate output so that no overflow occurs.
%
% OUTPUTS:
%
%       new_state:              list of slabs inside anneal-machine at end of time-interval; these are listed
%                               vertically and the state-dimension of the i-th machine is specified by index_new_state(i)
%       index_new_state:        no of slabs in ith annealing machine for new_state
%       new_time:               n-dimensional vector; For the i-th machine:
%                                   If busy->busy: new_time(i)=old_time(i)+1
%                                   If finished annealing but not loaded output (for not causing overflow to out_anneal_buff)
%                                      new_time(i)=old_time(i) (time frozen)
%                                   If loaded from empty:  new_time(i) reset to 1
%                                   If empty at end of interval: new_time(i)=-1
%       output_from_anneal:   list of slabs: contribution from i-th machine is empty matrix if annealing is still in progress or
```

```
%                          annealing in i-th machine has finished but output will cause overflow to out_anneal_buff; else
%                          output is stacked to output_from_anneal array in machine order.
%
%-------------------------------------------------------------------------------------------------
% Get parameters
%
  out_anneal_buff_capacity=param.out_anneal_buff_capacity;
%-------------------------------------------------------------------------------------------------
  n=max(size(index_old_state));
  %
  if size(old_state,1) ~= sum(index_old_state)
     new_state=[];
     index_new_state=[];
     new_time=[];
     output_from_anneal=[];
     disp('Error in update_anneal_n.m ...')
     return
  end
  %
  state_out_anneal_buff=check_empty(state_out_anneal_buff);
  no_slabs_out_anneal_buff=size(state_out_anneal_buff,1);
  %
  new_state=[]; % initialize
index_new_state=zeros(1,n); % initialize
new_time=zeros(1,n); % initialize
output_from_anneal=[]; % initialize
current_input_list=input1; % initialize
out_slabs_count=0; % initialize
%
if isempty(input1); % no input from buffer
   for i=1:n % for each machine
       if index_old_state(i) ~= 0 % machine busy
          %
          if i==1
             old_state_i=old_state(1:index_old_state(i));
          else
             old_state_i=old_state(sum(index_old_state(1:i-1))+1:sum(index_old_state(1:i-1))+index_old_state(i));
          end % here old_state_i is known
          %old_state_i=determine_slabs_annealed(old_state,index_old_state,i);
          %
          top_slab_i=old_state_i(1,:);
          total_time_for_anneal_i=obtain_anneal_time(top_slab_i);
   %
          if old_time(i)+1 < total_time_for_anneal_i % annealing not finished at end of period
             new_state=[new_state ; old_state_i];
             new_time(i)=old_time(i)+1; % increment clock
             index_new_state(i)=index_old_state(i);
          else % process ends at end of time-interval
             if out_slabs_count + index_old_state(i) + no_slabs_out_anneal_buff <= out_anneal_buff_capacity % no overflow
                new_time(i)=-1;
                index_new_state(i)=0;
                output_from_anneal=[output_from_anneal ; old_state_i]; % stamp before returning
                out_slabs_count=out_slabs_count+index_old_state(i); % update count of out slabs
             else % ouput will cause overflow to out_anneal_buff
                new_state=[new_state ; old_state_i];
                index_new_state(i)=index_old_state(i);
                new_time(i)=old_time(i); % freeze time - try to output next time round
             end
          end
       else % machine is empty
          new_time(i)=-1;
       end % if
    end %for
else % some input waiting to be loaded
    for i=1:n % for each machine
        if index_old_state(i) ~= 0 % machine busy
           %
           if i==1
              old_state_i=old_state(1:index_old_state(1));
           else
              old_state_i=old_state(sum(index_old_state(1:i-1))+1:sum(index_old_state(1:i-1))+index_old_state(i));
           end % here old_state_i is known
           %old_state_i=determine_slabs_annealed(old_state,index_old_state,i);
           %
           % No input-> state transition at end of interval, possible state->output transition
           top_slab_i=old_state_i(1,:);
           total_time_for_anneal_i=obtain_anneal_time(top_slab_i);
           %
           if old_time(i)+1 < total_time_for_anneal_i % annealing not finished at end of period
              new_state=[new_state ; old_state_i];
              index_new_state(i)=index_old_state(i);
              new_time(i)=old_time(i)+1; % increment clock
           else % process ends at end of time-interval
              if out_slabs_count+index_old_state(i)+no_slabs_out_anneal_buff <= out_anneal_buff_capacity % no overflow
                 new_time(i)=-1;
                 index_new_state(i)=0;
                 output_from_anneal=[output_from_anneal ; old_state_i]; % stamp before returning
                 out_slabs_count=out_slabs_count+index_old_state(i); % update count of out slabs
            else % ouput will cause overflow to out_anneal_buff
              new_state=[new_state ; old_state_i];
              index_new_state(i)=index_old_state(i);
              new_time(i)=old_time(i); % freeze time - try to output annealed slabs next time round
            end
        end
        %
else % i-th machine is empty
   %
```

```
            if i==1
               old_state_i=old_state(1:index_old_state(1));
            else
               old_state_i=old_state(sum(index_old_state(1:i-1))+1:sum(index_old_state(1:i-1))+index_old_state(i));
            end % here old_state_i is known
            %old_state_i=determine_slabs_annealed(old_state,index_old_state,i);
            %
            % No state->output transition possible, check for input->state transition
            %
            [in_slabs,index_slabs_in]=determine_slabs_for_anneal(current_input_list); % check for compatibility
            current_input_list(index_slabs_in)=[]; % eliminate imported slabs to i-th machine
            current_input_list=check_empty(current_input_list);
            new_state=[new_state ; in_slabs]; % update new global state
            index_new_state(i)=size(in_slabs,1); % update with number of slabs
            if isempty(in_slabs)
               new_time(i)=-1; % keep time at -1
            else
               new_time(i)=1; % start clock
            end
         end %if
      end %for
   end %if
%
   output_from_anneal=anneal_stamp(output_from_anneal); % update anneal stamp
%
%---------------------------------------- end update_anneal1_n.m ----------------------------------------
```

▶ Function update_buff:

```
function
new_state=update_buff(old_state,input_from_hotline,output_to_highbay)
% function new_state=update_buff(old_state,input_from_hotline,output_to_highbay)
%
% update state of buff, located between Hotline and HighBay.
% Function should be called EITHER with input_from_hotline=[] OR output_to_highbay=[]
% (or both []). NOTE: This buffer does not define its output, which is "pulled in" by
% the HighBay. It is the responsibility of the HighBay (through function update_highbay)
% to check that requested row-size of "output_to_highbay" does not exceed buffer's
% state-dimension. Similarly, it is the responsibility of the hotline (via update_hotline)
% to ensure it does not load buff when later is at full capacity.
%
% INPUTS:
%
%       old_state:          List of slabs at beginning of current time-step
%       input_from_hotline: Empty matrix (no output from anneal_n) or list of slabs
%       output_to_highbay:  Empty matrix (no input requested flom HB) or list of slabs
%
% OUTPUTS:
%
%       new_state:          List of slabs at end of current time-step

  input_from_hotline=check_empty(input_from_hotline);
  output_to_highbay=check_empty(output_to_highbay);
  old_state=check_empty(old_state);
%
  if ~isempty(input_from_hotline) & ~isempty(output_to_highbay)
      new_state=[];
      disp('error in update_buff ...')
      return
  end
%
  if isempty(input_from_hotline) & isempty(output_to_highbay) % no input HB or from hotline
     new_state=old_state;
  elseif isempty(output_to_highbay) % Load from Hotline
     new_state=[old_state ; input_from_hotline];
  elseif isempty(input_from_hotline) % called from inside HB
     new_state=old_state;
     no_slabs_requested=size(output_to_highbay,1);
     new_state(1:no_slabs_requested,:)=[];
     new_state=check_empty(new_state);
  end
%
%---------------------------------- end update_buff.m ----------------------------------
```

▶ Function update_buff1:

```
  function [new_state,output_to_anneal]=update_buff1(old_state,input1,input2)
% function [new_state,output_to_anneal]=update_buff1(old_state,input1,input2)
% OLD FILE NOT IN USE!!!!!!
% Buff1: located between HighBay and Annealing machine
% INPUTS:
%       old_state: list of slab_types at current time_step
%       input1: Empty matrix (no output from HB) or list of slabs
%       input2: 1=Anneal machine empty; 0 Anneal machine busy
% OUTPUTS:
%       new_state: list of slabs after transition
%       output_to_anneal: empty if no transfer to anneal machine, else list of slabs to be transfered
%
  if isempty(input1) % no output from HB
     new_state=old_state;
  else
     new_state=[old_state ; input1];
  end
%
  if input2==0 % anneal machine busy
     output_to_anneal=[];
```

```
    return;
  else
      [output_to_anneal,index_out]=determine_slabs_for_anneal(new_state); % load 4 compatible slabs
      if size(index_out,2) ~= size(new_state,1)
         new_state(index_out)=[]; % if new_state=[], newstate([])=[] gives [];
      else
         new_state=[];
      end
  end
%
%------------------------ end of update_buff1.m --------------------------
```

▶ Function update_buff1_n:

```
function
[new_state,output_to_anneal]=update_buff1_n(old_state,input1,input2)
% function [new_state,output_to_anneal]=update_buff1_n(old_state,input1,input2)
% Buff1: located between HighBay and Annealing machine
% INPUTS:
%        old_state: list of slab_types at current time_step
%        input1: Empty matrix (no output from HB) or list of slabs
%        input2: n-dim row vector where n=no of annealing machines; input2(i)=1 means i-th annneal machine empty;
%                input2(i)=0 means i-th anneal machine busy
% OUTPUTS:
%        new_state: list of slabs after transition
%        output_to_anneal: empty if no transfer to anneal machine, else list of slabs to be transfered
%
  n=size(input2,2); % no of parallel annealing machines
%
  if isempty(input1) % no output from HB
     new_state=old_state;
  else
     new_state=[old_state ; input1];
  end
%
  output_to_anneal=[];
 %
  for i=1:n
      if input2(i) == 1 % i-th anneal machine empty
         [output_to_anneal_i,index_out_i]=determine_slabs_for_anneal(new_state); % load 4 compatible slabs
         if size(output_to_anneal_i,1) ~= 0; % slabs match found
            new_state(index_out_i)=[];
            output_to_anneal=[output_to_anneal ; output_to_anneal_i];
         end
      end
   end
%
if size(new_state,1)==0 | size(new_state,2)==0
   new_state=[];
end
%
%------------------------ end of update_buff1_n.m --------------------------
```

▶ Function update_bwg:

```
function
[new_state,new_time,output_from_bwg]=update_bwg(old_state,old_time,input1);
% function [new_state,new_time,output_from_bwg]=update_bwg(old_state,old_time,input1);
%
% INPUTS:
%        old_state: coils being in bwg or empty matrix
%        old_time: empty matrix if bwg free; time-step since
%                  beginning of bwg process if bwg is busy
%        input1:   list of coils from buffer output (could be empty)
% OUTPUTS:
%        new_state: coil inside bwg at end of time-interval
%        new_time: If busy->busy: new_time=old_time+1
%                     busy->just finished new_time=[];
%                     just loaded from empty:  new_time reset 1
%        output_from_bwg: empty if bwg process still in progress, or
%                         bwg not busy, else coil that passed bwg stage
%
%
  if isempty(old_state) % bwg not processing a coil
     if isempty(input1); % no input from buffer
        new_state=[];
        new_time=[];
        output_from_bwg=[];
        return
     else
        new_state=input1;
        new_time=1; % initialize clock
        output_from_bwg=[];
     end
  else % bwg in progress
     coil=old_state;
     bwg_time=obtain_bwg_time(coil);

     if old_time+1 < bwg_time % bwg process not finished at end of period
        new_state=old_state;
        new_time=old_time+1; % increment clock
        output_from_bwg=[];
     else % process ends at end of time-interval
        new_state=[];
        new_time=[];
        output_from_bwg=old_state;
```

```
    end
end % if
%
%------------------------------------ end update_bwg ------------------------------------
```

▶ Function update_cold_roll:

```
function
[new_state,new_time,output_from_cold_roll]=update_cold_roll(old_state,old_time,input1,state_out_cold_roll_buff,param);
% function [new_state,new_time,output_from_cold_roll]=update_cold_roll(old_state,old_time,input1,state_out_cold_roll_buff,param);
%
% INPUTS:
%
%        old_state:                 Coil being cold-rolled or empty matrix.
%        old_time:                  Specifies time since last start of cold-rolling
%        input1:                    Coil from buffer in_cold_roll_buff or empty matrix
%        state_out_cold_roll_buff   Current state of out_cold_roll_buffer (list of slabs or empty matric). Used to
%                                   calculate output so that no overflow occurs.
%
% OUTPUTS:
%
%        new_state:                 Current coil being cold-rolled at end of time-interval or empty matrix.
%        new_time:                  Time since last rolling begun at end of current time-step
%                                      If busy->busy: new_time=old_time+1
%                                      If cold-rolling completed but cold-rolled slab not loaded to out buffer (so that
%                                         no overflow occurs): new_time=old_time (time frozen)
%                                      If loaded from empty:  new_time reset to 1
%                                      If machine empty at end of time interval: new_time(i)=-1
%        output_from_cold_roll      Cold-rolled slab loaded to out_cold_roll_buffer or empty matrix.

%--------------------------------------------------------------------------
% Decode param structure
%
  out_cold_roll_buff_capacity=param.out_cold_roll_buff_capacity;
%-------------------------------------------------------------------------------------------------
  input1=check_empty(input1);
  old_state=check_empty(old_state);
  if ~isempty(old_state)
      old_state=old_state(1,:); % only one coil treated in CR!!!
  end
  state_out_cold_roll_buff=check_empty(state_out_cold_roll_buff);
  no_slabs_out_cold_roll_buff=size(state_out_cold_roll_buff,1);
%
  if isempty(input1); % no input from buffer
     if ~isempty(old_state)% machine busy
         %gauge_coil=old_state.gauge;
         total_time_for_rolling=determine_cold_roll_time(old_state);
         %
         if old_time+1 < total_time_for_rolling % cold-rolling not finished at end of period
               new_state=old_state;
               new_time=old_time+1; % increment clock
               output_from_cold_roll=[];
         else % process ends at end of time-interval
               if no_slabs_out_cold_roll_buff+1 <= out_cold_roll_buff_capacity % no overflow out_cold_roll_buff
                  new_time=-1;
                  new_state=[];
                  output_from_cold_roll=old_state;
               else % ouput will cause overflow to out_cold_roll_buff
                  new_state=old_state;
                  output_from_cold_roll=[];
                  new_time=old_time; % freeze time - try to output next time round
               end
         end
     else % machine is empty
           new_state=[];
           output_from_cold_roll=[];
           new_time=-1;
     end % if
     %
else % some input waiting to be loaded
     %
     if ~isempty(old_state) % machine busy
          %gauge_coil=old_state.gauge;
          total_time_for_rolling=determine_cold_roll_time(old_state); %????????
          %
          if old_time+1 < total_time_for_rolling % cold-rolling not finished at end of period
             new_state=old_state;
             output_from_cold_roll=[];
             new_time=old_time+1; % increment clock
          else % process ends at end of time-interval
             if no_slabs_out_cold_roll_buff+1 <= out_cold_roll_buff_capacity % no overflow out_cold_roll_buff
                new_time=-1;
                new_state=[];
                output_from_cold_roll=old_state;
              else % ouput will cause overflow to out_anneal_buff
                new_state=old_state;
                output_from_cold_roll=[];
                new_time=old_time; % freeze time - try to output next time round
             end
       end
       %
else % cold-rolling machine is empty
       %
       input1=input1(1,:); % get top slab
       new_state=input1; % new_state
       output_from_cold_roll=[];
       new_time=1; % start clock
end %if
```

```
 end % if
%
output_from_cold_roll=cold_roll_stamp(output_from_cold_roll); % stamp before exiting
%
%---------------------------------------- end update_cold_roll.m ----------------------------------------
```

▶ Function update_highbay_new:

```
function
[new_state,out_to_buff1_n,out_to_in_cold_roll_buff,out_to_in_BWG_buff,in_from_buff,in_from_anneal_buff,...
          in_from_out_cold_roll_buff,state_out_anneal_buff,state_out_cold_roll_buff,state_buff,index_list_coils_cr, ...
          real_crane_moves]= ...
          update_highbay_new(old_state,time_step,state_buff1_n,state_in_cold_roll_buff,state_in_BWG_buff, ...
          state_out_anneal_buff,state_out_cold_roll_buff,state_buff,index_list_coils_cr,param);
%
% function [new_state,out_to_buff1_n,out_to_in_cold_roll_buff,out_to_in_BWG_buff,in_from_buff,in_from_anneal_buff,...
%          in_from_out_cold_roll_buff,state_out_anneal_buff,state_out_cold_roll_buff,state_buff,index_list_coils_cr,...
%          real_crane_moves]= ...
%          update_highbay_new(old_state,time_step,state_buff1_n,state_in_cold_roll_buff,state_in_BWG_buff, ...
%          state_out_anneal_buff,state_out_cold_roll_buff,state_buff,index_list_coils_cr,param);
%
% Function updates HighBay state and outputs.
%
% INPUTS:
%
% old_state                   list of coils being stored in HighBay or empty matrix; these are listed vertically
% time_step                   current iteration index (integer)
% state_buff1_n               state of buff1_n (list of coils) at beginning of time step
% state_in_cold_roll_buff     state of in_cold_roll_buff (list of coils) at beginning of time step
% state_in_BWG_buff
% state_out_anneal_buff       state of out_anneal_buff (list of coils) at beginning of time step. Note: this state
%                             is updated internally.
% state_out_cold_roll_buff    state of out_cold_roll_buff (list of coils) at beginning of time step. Note: this state
%                             is updated internally.
% state_buff                  state of buff (list of coils) at beginning of time step. Note: this state is updated
%                             internally.
% index_list_coils_cr         index of coils in identified matched list in old_state
% param                       parameter structure
%
% OUTPUTS:
%
% new_state                   list of coils being stored inside HB at end of time-interval, listed vertically
% out_to_buff1_n              list of matching coil quartets loaded to buff1_n (input side of anneal_n)
% out_to_in_cold_roll_buff    list of coils loaded to input cold-roll buffer
% out_to_in_BWG_buff          list of coils loaded to input BWG buffer
% in_from_buff                list of coils loaded into HighBay from buffer (out-side of HotLine)
% in_from_anneal_buff         list of coils loaded from anneal buffer (out-side of annealing machine)
% in_from_out_cold_roll_buff  list of coils loaded to HB from cold-roll output buffer
% state_out_anneal_buff       state of out_anneal_buff (list of coils) at end of time step. Note: this state
%                             internally.
% state_out_cold_roll_buff    state of out_cold_roll_buff (list of coils) at endof time step. Note: this state
%                             is updated internally.
% state_buff                  state of buff (list of coils) at end of time step. Note: this state is updated
%                             internally.
% index_list_coils_cr         index of coils in identified matched list in new_state
% real_crane_moves            real no of crane moves
%
% RULES SEQUENCE:
%
% (1) Load out all COOL coils to out_buff.
% (2) Load out compatible quartets of COOL coils to buff1_n, provided total no of coils in buff1_n does not exceed
%     buff1_n's capacity, starting from cooler coil.
% (3) Load out COOL coils to in_cold_roll_buff, provided buffer's capacity not exceeded, starting from cooler coil.
% (4) Load in coils from out-anneal buffer, provided HB capacity is not exceeded
% (5) Load in coils from out-cold-roll buffer, provided HB capacity is not exceeded
% (6) Load in coils from buffer (out-HotLine side) provided HB capacity is not exceeded
%
% ADDITIONAL FEATURES:
%
% Each coil ENTERING HB is stabled with current clock-time (used to calculate cool-off time)
% Each coil LEAVING HB has clock-stamp cleared
% Each coil ENTERING HB from ANNEAL output buffer has its anneal flag set to 1
% Each coil ENTERING HB from COLD ROLL out buffer has its cold-roll flag incremented by 1
%
%-------------------------------------------------------------------------------------------------
% Decode param structure
%
  simulation_time=param.simulation_time;
  simulation_step=param.simulation_step;
  no_anneal_machines=param.no_anneal_machines;
  high_bay_capacity=param.high_bay_capacity;
  buff_capacity=param.buff_capacity;
  buff1_n_capacity=param.buff1_n_capacity;
  in_BWG_buff_capacity=param.in_BWG_buff_capacity;
  capacity_BWG_buff=param.capacity_BWG_buff;
  out_anneal_buff_capacity=param.out_anneal_buff_capacity;
  in_cold_roll_buff_capacity=param.in_cold_roll_buff_capacity;
  out_cold_roll_buff_capacity=param.out_cold_roll_buff_capacity;
  cool_time_after_anneal=param.cool_time_after_anneal;
  cool_time_after_coldroll=param.cool_time_after_coldroll;
  crane_moves=param.crane_moves;
  priorities=param.priorities;
  input_file=param.input_file;
  %
  max_no_circulating_coils=4*no_anneal_machines+buff1_n_capacity+out_anneal_buff_capacity+ ...
                           in_cold_roll_buff_capacity+out_cold_roll_buff_capacity+1;

%
%   param= struct('simulation_time',simulation_time,...
```

```
%         'simulation_step',simulation_step, ...
%         'no_anneal_machines',no_anneal_machines, ...
%         'high_bay_capacity',high_bay_capacity,...
%         'buff_capacity',buff_capacity, ...
%         'buff1_n_capacity',buff1_n_capacity,...
%         'in_BWG_buff_capacity',in_BWG_buff_capacity,...
%         'capacity_BWG_buff',capacity_BWG_buff,...
%         'out_anneal_buff_capacity',out_anneal_buff_capacity, ...
%         'in_cold_roll_buff_capacity',in_cold_roll_buff_capacity,...
%         'out_cold_roll_buff_capacity',out_cold_roll_buff_capacity, ...
%         'cool_time_after_anneal',cool_time_after_anneal,...
%         'cool_time_after_coldroll',cool_time_after_coldroll, ...
%         'crane_moves',crane_moves,...
%         'input_file',input_file, ...
%         'priorities',priorities);
%
%-------------------------------------------------------------------------
% new_state=[];
% out_to_buff1_n=[];
% out_to_in_cold_roll_buff=[];
% out_to_in_BWG_buff=[];
% in_from_buff=[];
% in_from_anneal_buff=[];
% in_from_out_cold_roll_buff=[];
% state_out_anneal_buff=[];
% state_out_cold_roll_buff=[];
% real_crane_moves=[];

%-------------------------------------------------------------------------------------------------
  old_state=check_empty(old_state);
  new_state=old_state;
  real_crane_moves=zeros(1,5);
%
% get no of coils stored in HB
  no_slabs=size(old_state,1); % no of coils inside HB at beginning of this time-interval
%
% Determine if a list of coils (initially 20 or 40) has been identified in
% an earlier step
%
  index_list_coils_cr=check_empty(index_list_coils_cr);
  if isempty(index_list_coils_cr)
     flag_early_list=0; % no list for cold-rolling exists
  else
     flag_early_list=1; % list exists
  end
%
% Determine which of the cold-rolled coils have cooled (so they can be moved to output_buff);
% also, which of the coils which have been annealed have cooled (so they can move to in_cold_roll_buff),
% and which coils need to be loaded to anneal machine through buff1_n
%
  index_slabs_to_in_BWG_buff=[]; % initialize
  index_slabs_to_new_list=[]; % initialize
  index_slabs_to_buff1_n=[]; % initialize
  %
  for i=1:no_slabs
      %
      current_slab=old_state(i,:);
      time_of_entry=current_slab.timer;
      current_slab_flags=current_slab.flags;
      current_slab_anneal_flag=current_slab_flags(2);
      current_slab_cold_roll_flag=current_slab_flags(3);
      current_slab_gauge=current_slab.gauge;
      current_slab_width=current_slab.width;
      %
      if (current_slab_anneal_flag == 0)  % check if coil has not been annealed
         index_slabs_to_buff1_n=[index_slabs_to_buff1_n i]; % potentially sent for annealing
      elseif (current_slab_anneal_flag == 1) % slab has been annealed
          time_in_highbay=time_step-time_of_entry;
          [flag1,flag_ready]=determine_cold_status(current_slab,time_in_highbay,param);
          if (flag1==1 & flag_ready==1)
             index_slabs_to_new_list=[index_slabs_to_new_list i];
          elseif (flag1==0 & flag_ready==1)
             index_slabs_to_in_BWG_buff=[index_slabs_to_in_BWG_buff i];
          end
      end % if
%
  end % for
%
% Restrict slabs that can be moved to input-BWG buffer according to buffer
% capacity
%
  state_in_BWG_buff=check_empty(state_in_BWG_buff);
  no_coils_in_BWG_buff=size(state_in_BWG_buff,1);
  available_places_BWG_buff=capacity_BWG_buff-no_coils_in_BWG_buff;
  no_slabs1=size(index_slabs_to_in_BWG_buff);
%
  if available_places_BWG_buff <= no_slabs1
     index_slabs_to_in_BWG_buff=index_slabs_to_in_BWG_buff(1:available_places_BWG_buff);
  end
%
  out_to_in_BWG_buff=check_empty(old_state(index_slabs_to_in_BWG_buff,:)); % slabs potentially movable to in_BWG_buff
% with indices index_slabs_to_in_BWG_buff referred to old state
  no_out_to_in_BWG_buff=size(out_to_in_BWG_buff,1); % and their number
%
% Determine coil (if any) to be moved to cold-roll_buffer (in-side)
%
% if time_step==51
%    keyboard
```

```
% end

  state_in_cold_roll_buff=check_empty(state_in_cold_roll_buff);
  no_coils_in_cold_roll_buff=size(state_in_cold_roll_buff,1);
  if flag_early_list == 1 & (no_coils_in_cold_roll_buff < in_cold_roll_buff_capacity)
     slab_cr_index=index_list_coils_cr(1); % slab index potentially going for CR
     slab_cr=old_state(index_list_coils_cr(1),:); % top coil
  elseif flag_early_list == 1 & (no_coils_in_cold_roll_buff >= in_cold_roll_buff_capacity)
     slab_cr_index=[];
     slab_cr=[];
  elseif flag_early_list == 0 % no list exists
     slab_cr_index=[];
     slab_cr=[];
     % try to make new matching list
     index_slabs_to_new_list=check_empty(index_slabs_to_new_list);
     if ~isempty(index_slabs_to_new_list) % some coils found that are ready for cold-rolling
         slabs_to_new_list=old_state(index_slabs_to_new_list,:);
         [slabs_to_new_list,index_out]=determine_slabs_for_rolling1(slabs_to_new_list);
         slabs_to_new_list=check_empty(slabs_to_new_list);
         if isempty(slabs_to_new_list) % no match found
            slab_cr_index=[]; % no slab to go for CR
            slab_cr=[]; % no slab to go for CR
            index_list_coils_cr=[]; % no matched list next time round
         else % matching list found
            slab_cr_index=[]; % no slab to go for CR
            slab_cr=[]; % no slab to go for CR
            index_list_coils_cr=index_slabs_to_new_list(index_out); % refer indexes to old state!
         end
      else % no coil found ready for cold-rolling
         slab_cr_index=[];
         slab_cr=[];
         index_list_coils_cr=[];
      end
  end
%
% At this stage potential slab to be move to input-cold rolling buffer is slab_cr and its index in
% old_state is slab_cr_index. Also, if flag_early_list=0 (empty matched list was inputed), new list
% of matched coils for cold rolling is created (possibly empty) as array index_list_coils_cr,
% indexed with respect to old_state. If flag_early_list=1 (non-empty matched list was inputed)
% output list either stays the same (if no coil is moved to input cold-roll buffer) or its first row
% will be deleted.
%
% Determine coils to be moved to buff1_n
%
  if ~isempty(index_slabs_to_buff1_n) % some coils can be loaded
      out_to_buff1_n=new_state(index_slabs_to_buff1_n',:);
  else
      out_to_buff1_n=[];
  end
  index2=size(index_slabs_to_buff1_n,2); % coils that could potentially move to buff1_n
  state_buff1_n=check_empty(state_buff1_n);
  no_slabs_in_buff1_n=size(state_buff1_n,1); % no of coils in buff1_n
  available_places_in_buff1_n = buff1_n_capacity-no_slabs_in_buff1_n;
  available_quartets_in_buff1_n = floor(available_places_in_buff1_n/4); % coils loaded in groups of 4
%
  if available_quartets_in_buff1_n == 0 | index2 < 4
     temp_slab_array=[];
     temp_slab_index_array=[];
     %out_to_buff1_n =[];
  else
     temp_slab_array=[];
     temp_slab_index_array=[];
     for i=1:available_quartets_in_buff1_n
           [out_slabs,index_out]=determine_slabs_for_anneal(out_to_buff1_n);
           if isempty(index_out)
              break;break;
           else
              temp_slab_array=[temp_slab_array ; out_slabs];
              temp_slab_index_array=[temp_slab_index_array index_slabs_to_buff1_n(index_out)];
              out_to_buff1_n(index_out')=[];
              index_slabs_to_buff1_n(index_out)=[];
           end
     end
  end
  out_to_buff1_n=temp_slab_array;
  out_to_buff1_n=check_empty(out_to_buff1_n);
  no_out_to_buff1_n=size(out_to_buff1_n,1);
  index_slabs_to_buff1_n=temp_slab_index_array;
  index_slabs_to_buff1_n=check_empty(index_slabs_to_buff1_n);
%
% NOTE: Determine whether there are slabs to be moved to and/or from input
% and output buffers of annealing machine. Check capacities of input
% buffer/highbay. If at least one quartet of 'slabs can be moved, move it
% (after clearing time stamp), update state of output buffer of annealing machine,
% construct new state, new-list and exit. Else perform in/out loading operations
% in sequence of list of priorities (subject to capacity constraints), until all
% permissible crane movements have been exhausted or all tasks have been performed.
%
% Determine available places in HighBay
  available_places=high_bay_capacity-size(new_state,1); % available slots
  available_places=available_places+no_out_to_buff1_n; % vacated slots when buff1_n is loaded
%
  state_out_anneal_buff=check_empty(state_out_anneal_buff); % check for emptry matrix
  no_slabs_out_anneal_buff=size(state_out_anneal_buff,1); % no of coils waiting to be loaded in HB
  no_slabs_in=min(no_slabs_out_anneal_buff,available_places); % no slabs entering HB
  real_crane_moves(2)=no_out_to_buff1_n+no_slabs_in; % no free trips!
  %
  % max(no_slabs_out_anneal_buff,no_slabs_in); % some crane trips are free!
```

```
%
  if (no_out_to_buff1_n > 0) | (no_slabs_in > 0)
     % coils can be loaded to in or from out annealing-machine buffer
     %
     if (no_out_to_buff1_n > 0) % load out
        %
        out_to_buff1_n=remove_time_stamp(out_to_buff1_n); % remove time-stamp from out-going coils
        new_state(index_slabs_to_buff1_n)=[]; % remove out-going slabs from state-vector
        %
        % Update matching-list of slabs (FUNCTION NOT IMPLEMENTED YET!!!)
        % Function updates first-argument index list by assuming that coils
        % listed in second index list are removed
        %
          index_list_coils_cr=update_list(index_list_coils_cr,index_slabs_to_buff1_n);
        %
     end
     %
     in_from_anneal_buff=[];
     %
     if (no_slabs_in > 0) % load in
        %
        in_from_anneal_buff=state_out_anneal_buff(1:no_slabs_in,:);
        in_from_anneal_buff=check_empty(in_from_anneal_buff);
        in_from_anneal_buff=set_time_stamp(in_from_anneal_buff,time_step); % set time-stamp
        new_state=[new_state ; in_from_anneal_buff]; % new-state update
        % call update_out_anneal_buff to update its state
        state_out_anneal_buff=update_out_anneal_buff(state_out_anneal_buff,[],in_from_anneal_buff);
        available_places=available_places-no_slabs_in; % update available places
     end

     %
     out_to_in_cold_roll_buff=[];
     out_to_in_BWG_buff=[];
     in_from_buff=[];
     in_from_out_cold_roll_buff=[];
     %
     return
     %
  end
%
% If this point is reached, no coils were moved from/to in and out anneal buffers
%
out_to_buff1_n=[]; in_from_anneal_buff=[];
real_crane_moves=zeros(1,5); % reset to zero
%
if priorities(1)==1 & priorities(2)==2 & priorities(3)==3
   priorities123;
elseif priorities(1)==1 & priorities(2)==3 & priorities(3)==2
   priorities132;
elseif priorities(1)==2 & priorities(2)==1 & priorities(3)==3
   priorities213;
elseif priorities(1)==3 & priorities(2)==1 & priorities(3)==2
   priorities312;
elseif priorities(1)==3 & priorities(2)==2 & priorities(3)==1
   priorities321;
elseif priorities(1)==2 & priorities(2)==3 & priorities(3)==1
   priorities231;
end %if
%
% ---------------------------------- end of update_highbay_new.m ---------------------------------
```

▶ Function update_in_BWG_buff:

```
function
[new_state,output_to_BWG]=update_in_BWG_buff(old_state,input1,input2)
% function [new_state,output_to_BWG]=update_in_BWG_buff(old_state,input1,input2)
% Update state and output of in_BWG_buff, located between HighBay and BWG machine
%
% INPUTS:
%
%        old_state:              list of slab_types at current time_step
%        input1:                 Empty matrix (no output from HB) or list of slabs
%        input2:                 1=BWG machine empty; 0 BWG machine busy
%
% OUTPUTS
%
%        new_state:              list of slabs after transition
%        output_to_BWG:          empty if no transfer BWG, else slab to be transfered, taken from
%                                top of the slabs list.
%
  input1=check_empty(input1);
  old_state=check_empty(old_state);
%
  if isempty(input1) % no output from HB
     new_state=old_state;
  else
     new_state=[old_state ; input1];
  end
%
  if input2 == 0 % BWG machine busy
     output_to_BWG=[];
     return;
  else
      if ~isempty(new_state)
          output_to_BWG=new_state(1); % load top coil
          new_state(1)=[];
          new_state=check_empty(new_state);
      else
```

```
          output_to_BWG=[];
       end
  end
%
%---------------------------------- end of update_in_BWG_buff.m ---------------------------------
```

▶ Function update_list:

```
function index_out=update_list(index_in,index_delete)
% function index_out=update_list(index_in,index_delete)
% Updates index list index_in when indexes in index_list are deleted from
% main list. It is assumed that index_in and index_list have no entries in
% common

index_in=sort(index_in); % sort in ascending order
index_delete=sort(index_delete); % sort in ascending order
n1=size(index_in,2); n2=size(index_delete,2);

for i=1:n1
    if any(index_in(i) == index_delete)
       index_out=[];
       return
   end
end
%
index_out=index_in;
%
for i=1:n2
    index_temp=find(index_in > index_delete(i));
    index_temp=check_empty(index_temp);
    if ~isempty(index_temp)
        for j=1:size(index_temp,2)
            index_out(index_temp(j))=index_out(index_temp(j))-1;
        end
    end
end
%
%---------------------------- end of update_list.m -----------------------
```

▶ Function update_out_anneal_buff:

```
function
new_state=update_out_anneal_buff(old_state,input_from_anneal,output_to_highbay)
% function new_state=update_out_anneal_buff(old_state,input_from_anneal,output_to_highbay)
%
% update state of out_anneal_buff, located between anneal machine and HignBay.
% Function should be called EITHER with input_from_anneal=[] OR output_to_highbay=[]
% (or both []). NOTE: This buffer does not define its output, which is "pulled in" by
% the HighBay. It is the responsibility of the HighBay (through function update_highbay)
% to check that requested row-size of "output_to_highbay" does not exceed buffer's
% state-dimension. Similarly, it is the responsibility of the anneal machine (via
% update_anneal_n) to ensure that row-size of "input_from_anneal" pushed into the buffer
% will not cause overflow (i.e. that buffer's state-dimension + input_from_anneal <=
% buffer's capacity)
%
% INPUTS:
%        old_state: list of slab_types at current time_step
%        input_from_anneal: Empty matrix (no output from anneal_n) or list of slabs
%        output_to_highbay: Empty matrix (no input requested flom HB) or list of slabs
%
% OUTPUTS:
%        new_state: list of slabs after transition
%
  if size(input_from_anneal,1) == 0 | size(input_from_anneal,2) == 0
     input_from_anneal=[];
  end
%
   if size(output_to_highbay,1) == 0 | size(output_to_highbay,2) == 0
     output_to_highbay=[];
   end
%
   if size(old_state,1) == 0 | size(old_state,2) == 0
        old_state=[];
   end
%
  if ~isempty(input_from_anneal) & ~isempty(output_to_highbay)
      new_state=[];
      disp('error in update_out_anneal_buff ...')
      return
  end
%
  if isempty(input_from_anneal) & isempty(output_to_highbay) % no input from anneal_n/HB
     new_state=old_state;
  elseif isempty(output_to_highbay) % input from anneal
     new_state=[old_state ; input_from_anneal];
  elseif isempty(input_from_anneal)
     new_state=old_state;
     no_slabs_requested=size(output_to_highbay,1);
     new_state(1:no_slabs_requested)=[];
     if size(new_state,1) == 0 | size(new_state,2) == 0
        new_state=[];
     end
  end
%
%---------------------------------- end update_out_anneal_buff.m ---------------------------------
```

▶ Function update_out_cold_roll_buff:

```
function
new_state=update_out_cold_roll_buff(old_state,input_from_cold_roll,output_to_highbay)
% function new_state=update_out_cold_roll_buff(old_state,input_from_cold_roll,output_to_highbay)
%
% update state of out_cold_roll_buff, located between cold-rolling machine and HignBay.
% Function should be called EITHER with input_from_cold_roll=[] OR output_to_highbay=[]
% (or both []). NOTE: This buffer does not define its output, which is "pulled in" by
% the HighBay. It is the responsibility of the HighBay (through function update_highbay)
% to check that requested row-size of "output_to_highbay" does not exceed buffer's
% state-dimension. Similarly, it is the responsibility of the cold-rolling machine (via
% update_cold_roll) to ensure that row-size of "input_from_cold_roll" pushed into the buffer
% will not cause overflow (i.e. that buffer's state-dimension + input_from_cold_roll <=
% buffer's capacity)
%
% INPUTS:
%
%       old_state:              List of slab_types at start of current time-step
%       input_from_cold_roll:   Empty matrix (no output from anneal_n) or list of slabs
%       output_to_highbay:      Empty matrix (no input requested flom HB) or list of slabs
%
% OUTPUTS:
%
%       new_state:              List of slabs at end of current time-step
if size(input_from_cold_roll,1) == 0 | size(input_from_cold_roll,2)
== 0
    input_from_cold_roll=[];
  end
%
   if size(output_to_highbay,1) == 0 | size(output_to_highbay,2) == 0
     output_to_highbay=[];
   end
%
   if size(old_state,1) == 0 | size(old_state,2) == 0
        old_state=[];
   end
%
  if ~isempty(input_from_cold_roll) & ~isempty(output_to_highbay)
      new_state=[];
      disp('error in update_out_cold_roll_buff ...')
      return
  end
%
  if isempty(input_from_cold_roll) & isempty(output_to_highbay) % no input from in-cold-roll-buff/HB
     new_state=old_state;
  elseif isempty(output_to_highbay) % input from anneal
     new_state=[old_state ; input_from_cold_roll];
  elseif isempty(input_from_cold_roll)
     new_state=old_state;
     no_slabs_requested=size(output_to_highbay,1);
     new_state(1:no_slabs_requested)=[];
     if size(new_state,1) == 0 | size(new_state,2) == 0
        new_state=[];
     end
  end
%
%---------------------------------- end of update_out_cold_roll_buff.m -----------------------------
```

▶ Function update_output_buff:

```
  function new_state=update_output_buff(old_state,input1)
% function new_state=update_output_buff(old_state,input1)
%
% INPUTS:
%       old_state: list of slab_types in buffer
%       input1: slab/slabs
% OUTPUTS:
%       new_state: list of slabs after transition
%
%
  if isempty(input1)
     new_state=old_state;
  else
     new_state=[old_state ; input1];
  end
%
%----------------------- end of update_output_buff.m -----------
```