University of Groningen

Controlling the order pool in make-to-order production systems

Germs, Remco

IMPORTANT NOTE: You are advised to consult the publisher's version (publisher's PDF) if you wish to cite from it. Please check the document version below.

Document Version
Publisher's PDF, also known as Version of record

Publication date:
2012

Link to publication in University of Groningen/UMCG research database

Citation for published version (APA):
Germs, R. (2012). *Controlling the order pool in make-to-order production systems*. University of Groningen, SOM research school.

Copyright
Other than for strictly personal use, it is not permitted to download or to forward/distribute the text or part of it without the consent of the author(s) and/or copyright holder(s), unless the work is under an open content license (like Creative Commons).

The publication may also be distributed here under the terms of Article 25fa of the Dutch Copyright Act, indicated by the “Taverne” license. More information can be found on the University of Groningen website: https://www.rug.nl/library/open-access/self-archiving-pure/taverne-amendment.

Take-down policy
If you believe that this document breaches copyright please contact us providing details, and we will remove access to the work immediately and investigate your claim.

Downloaded from the University of Groningen/UMCG research database (Pure): http://www.rug.nl/research/portal. For technical reasons the number of authors shown on this cover page is limited to 10 maximum.

Download date: 11-12-2022

Controlling the Order Pool in Make-To-Order Production Systems

Remco Germs

Publisher:	University of Groningen Groningen, The Netherlands
Printer	Ipskamp Drukkers B.V. Enschede, The Netherlands
ISBN:	978-90-367-5262-6 (book) 978-90-367-5261-9 (e-book)

© 2012, Remco Germs

All rights reserved. No part of this publication may be reproduced, stored in a retrieval system of any nature, or transmitted in any form or by any means, electronic, mechanical, now known or hereafter invented, including photocopying or recording, without prior written permission of the author.

This document was prepared using LaTeX.

rijksuniversiteit
groningen

Controlling the Order Pool in Make-To-Order Production Systems

Proefschrift

ter verkrijging van het doctoraat in de
Economie en Bedrijfskunde
aan de Rijksuniversiteit Groningen
op gezag van de
Rector Magnificus, dr. E. Sterken,
in het openbaar te verdedigen op
donderdag 19 januari 2012
om 14.30 uur

door

Remco Germs

geboren op 23 mei 1981
te Stadskanaal

Promotor:	Prof. dr. ir. J. Slomp
Copromotores:	Dr. J. Riezebos
	Dr. N. D. van Foreest
Beoordelingscommissie:	Prof. dr. N. Vandaele
	Prof. dr. I. F. A. Vis
	Prof. dr. W. H. M. Zijm

Opgedragen aan mijn vriendin,
Lutiena Bouwers.

Contents

Chapter 1

Introduction

This chapter discusses the importance of short and reliable delivery times for make-to-order companies. It distinguishes three production control decisions that affect the delivery performance of such companies. Two of them, order acceptance and order release, are the focus of the research in this thesis. This chapter provides an overview of the order acceptance and order release problems that are explored and how they relate to the overall problem of achieving short and reliable delivery times.

1.1 Make-To-Order Production

In this thesis we focus on manufacturing firms that operate in a make-to-order (MTO) environment. In such an environment, all operations necessary to manufacture each specific product start after the receipt of a customer order. This manufacturing strategy allows firms to produce a high diversity of products in small quantities.

The MTO strategy is gaining more importance. The trend of outsourcing standard products that can be produced to forecast to low-wage countries has encouraged manufacturers in Western economies to shift their production from make-to-stock to make-to-order. Technological developments have given manufacturing companies the ability to provide their customers a large number of options in their products without incurring high additional costs for such customization. Along with this has come an increased demand for customized products which has led to a growth in the diversity of products that are produced to order (Stevenson et al., 2005).

In addition, customers today expect much faster and reliable delivery of their products than was acceptable in the past (Suri, 2010). Since MTO production is driven by customer orders, the length and predictability of time to complete an order (throughput time) directly determines the length and reliability of the delivery time of an order. As a result, for MTO firms it is becoming more and more of strategic importance that the throughput time of an order is short and predictable.

However, throughput times in MTO firms are in general long and unpredictable. Due to the high diversity of end-products, there is a high level of variability with respect to the routings (sequence of workstations visited by an order) and the processing times of orders. As a consequence, different orders with distinct processing requirements compete on the shop floor for the capacity of the same set of resources (i.e. workstations and workers). The limited capacities of the manufacturing resources cause orders to wait for the availability of these resources, resulting in queues of orders. In fact, the actual process time of an order (including setups, downtime, etc.) typically represents only a small fraction (5 to 10 percent) of the total throughput time of an order, while the majority of the extra time is spent on waiting in queues (Bradt, 1983, Hopp and Spearman, 2008). In addition, due to setup times and the high variability of arrivals, routings and processing times, the throughput times of orders in MTO environments are unpredictable.

This thesis is motivated by the aforementioned observations and mainly concentrates on policies for controlling the queues in MTO production so that delivery dates can be met, whilst good use is made of the available capacity. We continue in Section 1.2 with a discussion on how production control can influence the length of queues in MTO production. Section 1.3 introduces the main research problems addressed in this thesis.

1.2 Controlling the Order Pool

Production control can influence the length of queues by means of input or output control decisions. Input control regulates the amount of work (workload) that is allowed to flow into the production system, onto the shop floor or into a particular queue. Output control is the control of orders out of the production system, shop floor or queue and is achieved by adjusting capacity. In this thesis we focus on input control decisions and consider production capacity to be a given constraint, limited in the short run by the output control decision. In many MTO environments in which machines are the primary constraint the assumption that capacity is fixed in the short run is realistic; controlling short term fluctuations in workload by adjusting capacity is simply too costly in machine-capacitated production environments.

There are three decision moments in the flow of an order at which input control can regulate the length of queues in an MTO production system (Land and Gaalman, 1996):

(i) Dispatching – day to day shop floor control;

(ii) Release – short term production planning;

(iii) Acceptance – medium term production planning.

Traditionally a lot of attention has been given to the dispatching decision which is concerned with the choice which order to select next for processing after the operation of another order has been completed. The dispatching decision is made locally and on-line at each workstation and is generally based on some priority rule. Literally hundreds of different dispatching rules have been proposed by researchers as well as practitioners (see Blackstone Jr. et al., 1982, Haupt, 1989, Ramasesh, 1990, for extensive surveys) which makes this topic a bit overemphasized considering the fact that dispatching is a relative weak mechanism to control the length of queues, especially, if used alone (Hendry et al., 1998).

In this thesis we therefore focus on stronger instruments to control the length of queues: order release and order acceptance. Figure 1.1 illustrates these two decision moments in the flow of an order through an MTO production system. In what follows we provide a short description of the order release and order acceptance decisions, discuss their importance and concentrate on the most important ingredients of MTO production which complicate the control of queues at these decision moments. We refer to Land (2004) and the references therein for a more detailed summary on research regarding input control.

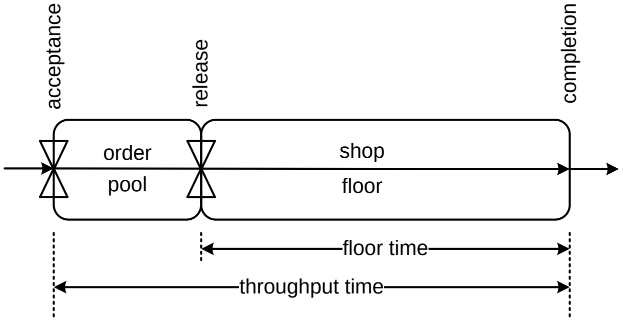

Figure 1.1: Order acceptance and order release decisions in the flow of an order.

1.2.1 Order Release

Order release decides when an order is authorized to enter the shop floor and to start its first operation. Until the release of an order to the floor, the order is just an item in the order pool (see Figure 1.1). In this thesis we focus our attention on an order release method that has become one of the cornerstones of modern manufacturing practice (Hopp and Spearman, 2004): pull production. A pull production system controls the throughput time of orders by constraining the workload on the shop floor.

The simplest way to constrain the workload on the shop floor is by controlling the number of orders on the shop floor. Alternatively, the workload can be constrained by controlling the work content (processing time) of orders. Pull systems that control the number of orders on the shop floor are referred to as *unit-based* and pull systems that constrain the workload based on the work content of orders as *load-based*. The simplest unit-based pull system that is also applicable in MTO production is CONWIP (Hopp and Spearman, 2008). CONWIP is a system that uses cards to maintain a constant maximum amount of orders on the shop floor. The most comprehensive example of a load-based pull system is Workload Control (WLC, see Thürer, 2011, for a recent literature review). WLC methods regulate the release of jobs by considering the current load (e.g., at each workstation), workload constraints and order characteristics (e.g., processing time).

The main reason for constraining the amount of work on the shop floor is to control the shop floor time of orders. By Little's law (Little, 1961), for a given arrival rate of orders a low level of average total workload on the shop floor implies a short average floor time. An additional advantage of controlled release is that it creates a transparent shop floor situation and therefore makes problems in the production process more noticeable (Hopp and Spearman, 2004). Our empirical findings (Slomp et al., 2009) also show that reducing the workload on the shop floor leads to improved quality of work and improved productivity of workers.

In spite of these advantages, many authors (e.g., Kanet, 1988, Melnyk and Ragatz, 1988) have long-since dismissed the concept of pull production in MTO environments, arguing that while controlled release reduces the shop floor time of orders, the total throughput time of orders increases. This is because the workload constraint blocks the release of orders whenever the workload limit is reached. When this happens, arriving orders have to wait in the order pool until the workload on the shop floor drops below the limit. Constraining the release of orders to the shop floor therefore increases the order pool time. As a result, they argue that the reduction in the floor time will be offset by the increase in the order pool time.

Recent literature (e.g., Land, 2004, Vandaele et al., 2008, Thürer, 2011) shows that some load-based pull system are able to reduce the total throughput time, if the system improves the balance of the workload on the shop floor, such that the variability of the workload at each workstation decreases. These pull methods are able to balance the workload on the shop floor by restricting the release of orders to workstations that are busy and by releasing orders to workstations that are waiting for work. The resulting more balanced arrival pattern of orders at each workstation leads to less blocking of the release of orders and thereby to a shorter order pool time for a given floor time. A disadvantage of load-based pull systems is their complexity. One of the problems with the implementation of advanced control systems is that the workforce does not understand the underlying logic of the system (Hendry et al., 2008). Hence, with respect to implementation, an important advantage of unit-based pull systems is their simplicity. However, whether unit-based pull systems can reduce shop floor and total throughput time simultaneously is still an open question.

1.2.2 Order Acceptance

As can be seen from Figure 1.1, the first decision moment in the flow of an order at which the amount of work in the production system can be controlled is order acceptance. Order acceptance largely determines the throughput time of orders in an MTO environment. Accepting too many orders leads to an over-loaded production system, in which throughput times increase and orders are increasingly delivered late (Ebben et al., 2005). Order acceptance thereby determines the constraints for the subsequent order release decision: once too many orders have been accepted, orders remain in the order pool for too long thereby missing their promised delivery dates. Hence the order release decision can itself only be fully effective if the queue of orders in the order pool is also controlled.

The decision which orders to accept and which to reject depends on the strategic direction of the firm, the current status of capacity already allocated, and the profitability of the order in question (Slotnick, 2011). In particular, there is a trade-off between the revenue brought in by a particular order, and all of its associated costs of processing, which includes the opportunity costs of having to reject later arriving orders.

There are a number of ways that firms can respond to the trade-off between cost of capacity and per-order revenue. The first option is to try to negotiate or renegotiate on delivery dates and prices. In the past, much research has been done in related areas of lead time estimation (Slotnick and Sobel, 2005, Öztürk et al., 2006), revenue management (Çelik and Maglaras, 2008), and due-date setting (Baker and Bertrand, 1981, Baker, 1984, Cheng and Gupta, 1989). In this thesis, we approach the trade-off by limiting ourselves to the decision which orders to accept and reject. In particular, the decision-maker is faced with a stream of randomly arriving orders and has the option of rejecting some of those orders if the available capacity is insufficient to meet promised delivery dates.

Due to the high variety of products, the large number of customers, and uncertainty in demand in MTO environments, the order acceptance problem is in general very complex. In order to reduce the complexity of the order acceptance problem typically some form of order classification is applied. That is, orders are grouped into families based on similar production characteristics, order lead times and revenues.

Order acceptance policies differ in the amount of information they use from the state of the production system (e.g., the workload and progress of previously accepted but not yet completed orders). At one extreme we have static admission policies that specify a priori whether each order family is admitted, based on the revenue that orders of this family generate and their expected capacity requirements. This specification is made independently of information about the state of the production system and is equivalent to determining whether one order family is preferred over the other. As an example, irrespective of the available capacity, a wholesaler generally does not accept small orders as the processing of small orders is not profitable at the low prices they quote.

On the other hand, dynamic admission policies make the decision of admitting an arriving order contingent on the current level of congestion upon arrival in the production system, in addition to the order family. Thereby dynamic strategies offer more flexibility in the allocation of resource capacity among different order families. Besides the use of state information, the performance of the production system can be improved significantly if order acceptance and scheduling decisions are considered jointly. This is especially important when a company faces large setup times from manufacturing one order family to another. Most literature on order acceptance however considers the two issues (order acceptance and scheduling) separately, undermining the company's profit and degrading the company's service level (Carr and Duenyas, 2000). This joint acceptance and scheduling problem recently received a lot of attention (see, e.g., Huang et al., 2011, Slotnick, 2011).

1.3 Research Objectives and Themes

Characteristics of MTO environments (such as the high variety of products, setup times, due-dates, random arrival of customer orders) make the control of queues in these environments a complex problem. A first approach to find good ways to control the length of queues in such a complicated situation is to analyze simple models that include only the most important ingredients of the complex real problem (Wijngaard, 2007). Second, the general insights obtained from these simple models can then be evaluated in more complex (possibly simulation) models, to check how robust these findings are and whether they can be used in real situations. Our focus in this thesis will be on the first part of this approach.

The main research objectives of this thesis are as follows:

1. To better understand the underlying mechanisms of good order acceptance and order release policies for MTO production environments;
2. To use these insights to develop simple order acceptance and order release policies to control queues in MTO production so that delivery dates can be met, whilst good use is made of the available capacity.

The reason to look explicitly for simple policies is that in MTO environments control decisions are often made by people. For a successful implementation of a control policy it is crucial that the workers understand the underlying logic of the policy (Fransoo et al., 2011). This makes it possible to use their insights and experience which further contributes to the performance of the control policy.

The thesis consists of two research themes corresponding to the two input control decisions we consider. In the first theme we consider the order release decision while the second theme focuses on the order acceptance decision. The following two paragraphs outline the content of these two research themes.

Theme 1 Order Release A popular way of controlling the workload on the shop floor is by means of unit-based pull systems. The popularity of these unit-based systems is for a large extent due to their ease of control (Hopp and Spearman, 2004) as this can be done using no more then a set of cards. Figure 1.2 illustrates how the CONWIP unit-based pull system uses cards to control the release of orders onto the shop floor.

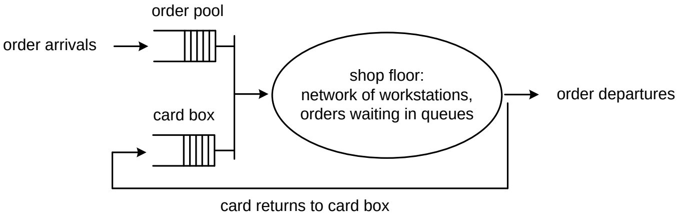

Figure 1.2: Representation of the CONWIP unit-based order release system.

In the figure, the shop floor is modeled as a network of workstations, each with a number of orders waiting to be processed. We see accepted orders entering the order pool. Signals are used to authorize the release of orders to the shop floor. Although these signals can be electronic we refer to them as cards.

The cards control the release of orders as follows. An order may only enter the shop floor if a free card can be attached to it. Upon completion, the order leaves the shop floor to fulfill the customers demand while the detached card is placed in a card box, where it waits until it is attached to a new order in the order book. In this system, each time an order leaves the shop floor the order book receives a signal to authorize the release of a new order. Because no order can enter the shop floor without a card attached to it, the number of orders on the shop floor is constrained by the number of cards circulating on the shop floor and in the card box.

Literature on unit-based pull systems that are applicable in MTO environments is scarce (Stevenson et al., 2005, Framinan et al., 2003). The unit-based pull systems that do seem suitable for MTO companies – POLCA, CONWIP and m-CONWIP, according to Stevenson et al. (2005) and Hopp and Spearman (2008) – receive only limited attention in performance comparisons. Moreover, all research into the performance of unit-based pull systems focuses on shop floor throughput time and workload levels and not on the performance indicator that is most relevant for MTO companies: total throughput time.

The first research theme of this thesis aims at making a start to fill this gap in the literature by analyzing throughput time performance and applicability of unit-based pull systems in an MTO environment. The following three papers contributing to the first theme are published or accepted for publication in international academic journals:

1. Slomp, J., J. A. C. Bokhorst and R. Germs. 2009. A lean production control system for high-variety/low-volume environments: A case study implementation. *Production Planning & Control* **20**(7) 586–595.

2. Germs, R. and J. Riezebos. 2010. Workload balancing capability of pull systems in MTO production. *International Journal of Production Research* **48**(8) 2345–2360.

3. Ziengs, N., J. Riezebos and R. Germs. 2011. Placement of effective work-in-progress limits in route-specific unit-based pull systems. Accepted for publication in the *International Journal of Production Research.*

Slomp et al. (2009) investigate by means of a case study/action research method how a unit-based pull system can be successfully implemented in a high-variety/-low-volume MTO environment. Germs and Riezebos (2010) and Ziengs et al. (2011) use a simulation model to address the question whether unit-based pull systems are able to reduce the total throughput time by improving the workload balance in an MTO system.

From these three papers we have selected the second paper to be included in this thesis as Chapter 2. The reason being that the author of this thesis is the first author of that paper.

Theme 2 Order Acceptance The second and major theme of this thesis concerns the problem of accepting and processing a stream of randomly arriving orders.

We start with considering a firm that supplies one type of product on order. Customer orders arrive randomly; each order concerns a batch of one product type. The firm has a single production process with finite capacity that produces the products in a batch-wise manner. To control the throughput times of orders, the firm has the possibility to reject customer orders. The problem of the firm is to determine a good acceptance strategy such that a given throughput time performance can be realized; and to find a batch service policy that determines, once a service batch is finished, when to start a new batch service.

This production situation has many applications. Good examples of batch-wise production systems are the ovens that can be found in the aircraft and manufacturing industry. A typical batch-wise process in the aircraft industry concerns the hardening of synthetic parts (Hodes et al., 1992, Van der Zee et al., 2001). Batch sizes are limited by the physical size of the oven and by a process constraint, which determines a maximum fill rate for the oven. The trade-off involved in the decision when to start a new batch service includes the balancing of rejection costs and logistical costs (e.g., stock keeping, machine utilization) against customer service requirements (e.g., short and reliable throughput times). Additional applications that possess more or less similar characteristics are in the serving of passengers by elevators, shuttle busses and ferries, and congestion control mechanisms to regulate transmission rates in packet-switched communication networks.

The described systems are known in the literature as batch arrival and batch service queues (usually called *bulk queues*) with restricted accessibility. A queueing system has restricted accessibility if not every customer (order) is admitted to the system. For such a system the admittance of an order will in general depend on the state of the queueing system at the moment of it's arrival. A system with a limited number of places in queue is a typical example of restricted accessibility

(Cohen, 1969): an arriving order that does not find sufficient room in the queue is not admitted.

To determine optimal system configuration, good admission control policies, and optimal batch sizing policies for these bulk queueing systems, it is helpful for managers and operators to be able to compute relevant performance measures, such as the average time of orders in the system (i.e., throughput time), moments of the number of accepted orders and rejection probabilities for arriving orders. In this thesis we therefore develop a simple, numerically stable, and efficient algorithmic method that allows the performance evaluation of various alternative system configurations and policies for bulk queueing systems with restricted accessibility.

The following two papers that consider the bulk queueing system with restricted accessibility are published or are under review for journal publication at the time of finishing this thesis:

1. Germs, R. and N. D. van Foreest. 2010. Loss probabilities for the $M^X/G^Y/1/K$ bulk queue. *Probability in the Engineering and Informational Sciences* **24** 457–471.

2. Germs, R. and N. D. van Foreest. 2011b. Analysis of finite-buffer state-dependent bulk queues. Under revision.

We included Germs and Van Foreest (2011b) in this thesis as Chapter 3 as this paper generalizes the model considered in Germs and Van Foreest (2010).

We continue by considering a production situation in which we generalize the above situation by including two important ingredients which are present in many MTO environments: setup times and due-dates. That is, we consider a production situation in which different items are produced on one machine. Customer orders drive the production and belong to product families, and have family dependent due-date, size, and profit margin. When production changes from one family to another a setup time is incurred. Orders are to be delivered on-date to customers and orders may be rejected if these orders cause late deliveries. For this production situation, it is critical to selectively accept and schedule customer orders, so that neither manufacturing capacity gets wasted on setups nor high profit earning orders are turned down because low profit earning orders have been previously accepted. This requires that order acceptance and scheduling decisions are considered jointly instead of separately, as mentioned before in Section 1.2.2.

Following Markowitz, Reiman, and Wein (2000), Markovitz and Wein (2001) and Winands et al. (2011) we will refer to this order acceptance and scheduling problem as the *Customized Stochastic Lot Scheduling Problem (CSLSP) with strict due-dates.*

There are many papers illustrating that the CSLSP is a common problem in practice; the Ph.D. thesis of Ten Kate (1995) contains an industrial motivation for this problem and describes specific applications, e.g., in a cardbox production plant and in the production of granulated plastics. Some other applications described in the open literature are in the production of steel tubes of various kinds and lengths to order (see Bertrand et al., 1990), production of baseball bats from aluminium tubes of different diameters (see Schmidt et al., 2001), scheduling of packaging pharmaceuticals on order (see Strijbosch et al., 2002) and forming effective batch sizes for an NMR scanner (see Vandaele et al., 2003).

The *off-line* counterpart of the CSLSP (that is, a scheduling problem in which all orders are known in advance rather than become known in the course of time) is a special case of the (NP-hard) Vehicle Routing Problem with selective pickups and time windows, which has many applications, e.g., in reverse logistics operations (Gutiérrez-Jarpa et al., 2010). Efficient (heuristic) algorithms for solving the off-line CSLSP have recently been proposed by Oğuz et al. (2010) and Huang et al. (2011).

Literature on how to approach the CSLSP is however very limited and mostly simulation based, such as Wester et al. (1992), Ten Kate (1995), Van Foreest et al. (2010). The detailed state description which is necessary to make rescheduling decisions upon order acceptance makes an analytical approach for finding an optimal policy for the CSLSP impossible for realistic problem instances. In this thesis we aim to make a start at filling this gap in the literature by studying a method to formulate the CSLSP as a Markov decision process (MDP) to gain insight into the optimal control of the production system.

The following two papers related to the CSLSP with strict due-dates are published or submitted for journal publication at the time of finishing this thesis:

1. Germs, R. and N. D. van Foreest. 2011a. Admission policies for the customized stochastic lot scheduling problem with strict due-dates. *European Journal of Operational Research* **213**(2) 375–383.

2. Germs, R. and N. D. van Foreest. 2011c. Order acceptance and sequencing policies for a make-to-order environment with setup times and family-

dependent lead times. Submitted.

Germs and Van Foreest (2011a) provide an MDP formulation for CSLSP with strict due-dates and use the MDP to benchmark the performance of a simple heuristic acceptance/scheduling policy. Germs and Van Foreest (2011c) extend the model by considering family-dependent lead times. The papers are complementary to each other so we included Germs and Van Foreest (2011a) and Germs and Van Foreest (2011c) in this thesis as Chapter 4 and Chapter 5. We tried to remove overlap in these chapters by skipping the literature overview section and the formal specification of MDP model in Chapter 5 of the thesis.

Chapter 2

Order Release: Unit-Based Pull Systems

In this chapter we study the throughput time performance of three order release policies that control the number of orders on the shop floor by just using a set of cards: CONWIP, m-CONWIP and POLCA. Due to their ease of control, these so-called *unit-based* pull systems are widely implemented in practice. However, all research into the performance of these pull systems focuses on shop floor throughput time and workload levels and not on the performance indicator that is most relevant for make-to-order companies: total throughput time. Their effectiveness in terms of reducing total throughput time is questioned. Theory states that an improvement in the average total throughput time will be due to the workload balancing capability of an order release system, but that many order release systems lack this capability. This chapter shows that this workload balancing capability exists for POLCA and m-CONWIP, but not for CONWIP. The magnitude of the effect differs strongly and depends on the configuration of the system, the order arrival pattern and the variability of the processing time of the orders.

2.1 Introduction

Nowadays, many make-to-order (MTO) companies focus on realising short throughput times as a competitive edge. Material control is an important part of the chain of tools used in realising short throughput times. A *material control system* regulates the flow of goods on the shop floor. This includes the authorisation to start an order, the release of new material on the shop floor, setting priorities for orders that are waiting to be processed, and initiating the start of succeeding activities, such as transport, quality control, etc.

Pull systems are a special type of material control system. They aim to control the throughput times of orders by constraining the amount of work (workload) on the shop floor (Hopp and Spearman, 2004). The simplest way to constrain the workload on the shop floor is by controlling the *number of orders* on the shop floor. Alternatively, the workload can be constrained based on the *work content* (processing time) of orders. We refer in this chapter to pull systems that control the number of orders on the shop floor as *unit-based* pull systems and to pull systems that constrain the workload based on the work content of orders as *load-based* pull systems. The Kanban material control system (Sugimori et al., 1977) is a well-known unit-based pull system, while WLC (Work Load Control, see Gaalman and Perona (2002) for a discussion) is the most sophisticated example of a load-based pull system. We restrict our attention to unit-based pull systems that control the throughput time of orders in an MTO environment. However, unit-based pull systems that are applicable in an MTO environment are scarce. This chapter discusses the throughput time performance of three unit-based pull systems that, according to Stevenson et al. (2005), appear to be suitable in an MTO environment: POLCA, CONWIP and m-CONWIP.

The throughput time performance of a pull system in an MTO environment depends on its capability to create a balanced distribution of the workload among the workstations on the shop floor. This capability of a pull system is known in the literature as the *workload balancing capability* (Land and Gaalman, 1998). The workload balancing capability of a pull system results in better control of the time of arrival of orders at the workstations on the shop floor. As a consequence, the average queue length required in front of these workstations to achieve a given utilisation level becomes smaller. This reduces the time between the release and completion of an order and might reduce the time between the arrival and completion of an order. We refer to the former as the *shop floor throughput time* and to the latter as the *total throughput time* of orders (see Section 2.2

for a detailed description of these two throughput time measures). We call the workload balancing capability of a pull system *effective* when the constraint on the workload results in both a reduction of the average shop floor throughput time and average total throughput time compared with the unconstrained system.

There are few published reports that are able to demonstrate the effective workload balancing capability of pull systems. The literature on workload control (e.g., Kanet, 1988, Melnyk and Ragatz, 1988) even suggests the existence of a paradox related to the absence of this workload balancing capability in unit-based pull systems. While practical implementations show significant reductions in the total throughput time of orders, simulation studies show that constraining the workload on the shop floor leads to both shorter shop floor throughput times and longer total throughput times. There are some studies, such as those of Land and Gaalman (1998), Breithaupt et al. (2002), and Land (2004), that show the existence of an effective workload balancing capability in load-based pull systems. However, we are not aware of any study that shows the existence of an effective workload balancing capability in unit-based pull systems.

The central question posed in this chapter is whether unit-based pull systems can have an effective workload balancing capability in an MTO environment. We introduce an MTO production system that perfectly suits pull systems that are able to balance the workload. By means of a simulation study, we analyse the workload balancing capability of POLCA, CONWIP and m-CONWIP in this specific production system. In the simulation study several experimental factors are varied, such as the processing time of orders and the order arrival pattern, to determine their influence on the magnitude of the workload balancing effect.

The structure of this chapter is as follows. Section 2.2 focuses on pull systems in MTO environments and describes the characteristics of the three unit-based pull systems considered in this chapter. Section 2.3 presents the research questions and Section 2.4 the design of the simulation study. Section 2.5 discusses the results and Section 2.6 concludes.

2.2 Pull Systems

As briefly mentioned in the Introduction, the literature on unit-based pull systems that are applicable for an MTO environment is scarce. Well-known unit-based pull systems such as Kanban are designed for make-to-stock (MTS) situations, as they use small intermediate stocks. In these pull systems, cards or containers (bins) are directly related to a specific product type. For example, an empty bin signals that it should be filled with exactly the same product type as before. For MTO companies, such a direct relation between signal and product type is not practically viable. MTO companies face a much greater product variety, which would lead to an unreasonable large number of different bins. In turn, the repetition of identical orders is not that frequent, which would lead to long waiting times of the intermediate stock in a bin. The combination of both effects would result in large work-in-process inventories.

There are some unit-based pull systems that are applicable in MTO companies (Stevenson et al., 2005). However, the unit-based pull systems that seem suitable for MTO companies (POLCA, CONWIP and m-CONWIP according to Stevenson et al. (2005)) receive only limited attention in performance comparisons. Framinan et al. (2003) provide an overview of 15 comparison studies of CONWIP with other material control systems, but only two of these studies consider the applicability of these systems in an MTO environment. The POLCA system is not included in one of these comparison studies. Fernandes and do Carmo-Silva (2006) do include the performance of POLCA, but again only for an MTS system. Studies that analyse the throughput time performance of unit-based pull systems in an MTO environment are therefore still largely lacking. This chapter aims at making a start to fill this gap in the literature by considering the throughput time performance of POLCA, CONWIP and m-CONWIP in an MTO environment.

To determine the throughput time performance of these unit-based pull systems, we distinguish in this chapter three (complementary) measures of the throughput time of orders that can be influenced by controlling the workload in an MTO production system. Figure 2.1 illustrates these measures graphically by the flow of an order through an MTO production system. Orders arrive at the production system and the material control system determines when an order is released to the shop floor. Until release to the shop floor the order waits in the order pool. As Figure 2.1 illustrates, we refer to the average time an order spends waiting in the order pool as the average order pool time (OPT) and to the average time between the release and completion of an order as the average shop floor throughput time (STT). The average total throughput time (TTT) is defined as the average time between the arrival and completion of an order and is therefore the sum of OPT and STT.

Pull systems can influence the STT by constraining the workload on the shop floor. If the total workload on the shop floor is low, according to Little's law

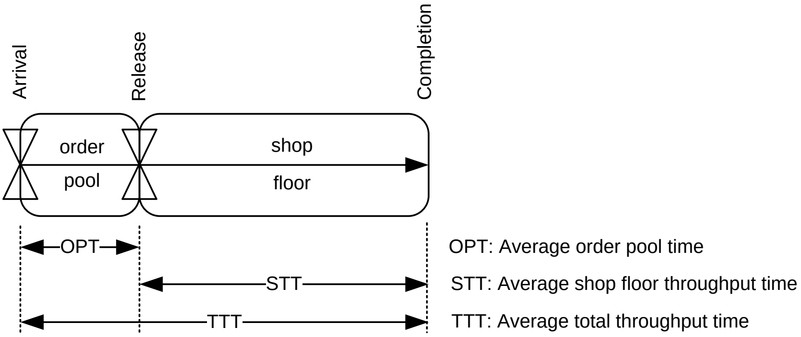
Arrival
Release
Completion
order
pool
shop
floor
OPT
STT
TTT
OPT: Average order pool time
STT: Average shop floor throughput time
TTT: Average total throughput time

Figure 2.1: Throughput time measures in an MTO production system.

(Little, 1961), the STT will be short. However, a short STT does not necessarily mean a short TTT. The workload constraint blocks the release of orders whenever the workload limit is reached. When this happens, arriving orders have to wait in the order pool until the workload on the shop floor drops below the limit. Constraining the release of orders to the shop floor therefore increases the OPT. As a result, the reduction in the STT can be offset by the increase in the OPT.

To reduce the TTT, the pull system must improve the balance of the workload on the shop floor, such that the variability of the workload at each workstation decreases. Pull systems can balance the workload on the shop floor by restricting the release of orders to workstations that are busy and by releasing orders to workstations that are waiting for work. The resulting more balanced arrival pattern of orders at each workstation leads to less blocking of the release of orders and thereby to a shorter OPT for a given STT. In case the constraint on the workload imposed by the pull system results in a reduction in STT that is higher than the increase in OPT, we call the workload balancing capability of the pull system effective for this workload constraint.

Figures 2.2(a) and (b) illustrate workload balance in a shop floor consisting of four workstations (A, B, C and D).

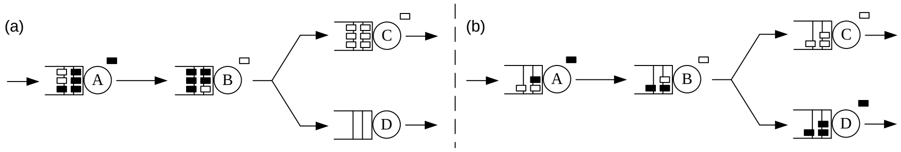

Figure 2.2: Shop floor without (a) and with (b) workload balancing.

Two types of orders (represented by black and white rectangles in the figure) are produced that differ with respect to the routing they follow on the shop floor. White orders follow route A→B→C, whereas black orders follow route A→B→D.

In Figure 2.2(a), the workload on the shop floor is not balanced. That is, workstation C is very busy, whereas workstation D is waiting for orders. Since a considerable amount of work directed for workstation D is waiting to be processed at workstations A and B, the opposite situation will occur when workstation C enters an idle period. This means that the variability of the workload at workstations C and D is also large.

Figure 2.2(b) shows the same shop floor, but this time the workload is balanced among the workstations. The variability of the workload at the workstations in this system is lower and throughput times will be shorter than in the unbalanced shop floor of Figure 2.2(a).

The workload balancing capability of a pull system is mainly determined by its pull structure, i.e. the specific pattern of control loops that regulates the workload on the shop floor (Gaury, 2000). In the next subsections we describe the pull structure of CONWIP, m-CONWIP and POLCA in detail.

2.2.1 CONWIP and m-CONWIP

The simplest unit-based pull system that is also applicable in MTO production is CONWIP (Hopp and Spearman, 2008). The CONWIP system can be explained by considering the production system in Figure 2.3.

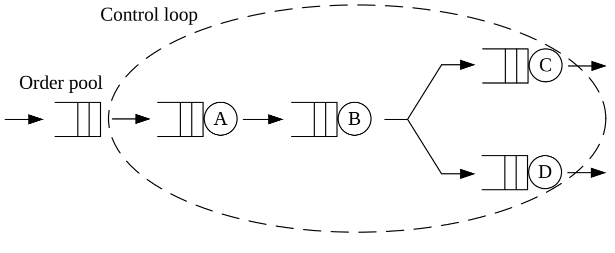

Figure 2.3: CONWIP controlled MTO production system.

The production system consists of an order pool and four workstations (A, B, C and D). The flow of orders between workstations A, B, C and D in the system is depicted by the thick arrows. After release, orders can follow two different routings on the shop floor. The dashed loop shows the part of the production system where the workload is controlled by cards. This loop is called the *control loop.* The CONWIP system has one control loop that constrains the total workload on the shop floor. This works as follows. An order may only enter the shop floor if a free card can be attached to it. Upon completion, the order leaves the shop floor to fulfil the customer's demand while the attached card is removed and then returns to the entrance of the shop floor, where it waits until it is attached to another order in the order pool. In the CONWIP system, each time an order leaves the shop floor the order pool receives a signal to authorise the release of a new order. Because no order can enter the shop floor without a card attached to it, the number of orders on the shop floor is constrained by the number of cards circulating on the shop floor. CONWIP uses a single control loop covering all workstations on the shop floor. This loop constrains the workload on the shop floor, but does not balance the work across the workstations. Therefore, the CONWIP system has no workload balancing capability.

Instead of using just one control loop for the whole shop floor, we can introduce a CONWIP loop for every possible routing on the shop floor. We denote such a system an 'm-CONWIP', where m stands for multiple CONWIP loops. Figure 2.4 gives an illustration of an m-CONWIP system. There are two routings on the shop floor and, therefore, the m-CONWIP system consists of two CONWIP loops. The two loops in this system balance the work among routings A→B→C and A→B→D by constraining the amount of orders that are allowed in these routings separately.

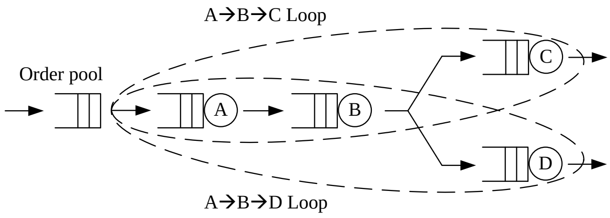

Figure 2.4: m-CONWIP controlled MTO production system.

In the CONWIP and m-CONWIP system that we consider in this chapter, a free card signals the release opportunity of a new order for the loop to which the card belongs. This means that in the order release decision, the processing time of the new order is not taken into account. Hence, CONWIP and m-CONWIP constrain the release based on the numbers of orders on the shop floor and are therefore unit-based pull systems.

2.2.2 POLCA

POLCA (Suri, 1998, Riezebos, 2010) is a pull system according to the definition of Hopp and Spearman (2004) because of its triggering authorisation mechanism. The triggering mechanism is a card system, which can be implemented either physically or electronically (Vandaele et al., 2008). Figure 2.5 displays a POLCA controlled MTO production system. In the POLCA system, each control loop covers two workstations. An order is allowed to start production on a given workstation when a card becomes available for the loop the order is trying to enter. Similar to the CONWIP system, the number of cards in a loop constrains the number of orders that are allowed in that loop.

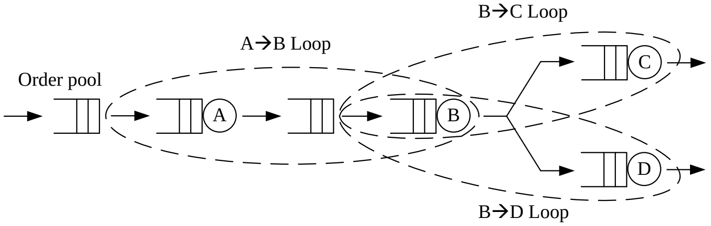

Figure 2.5: POLCA controlled MTO production system.

POLCA uses *overlapping* loops for orders that need to visit more than two workstations, as shown in Figure 2.5. The overlapping loops ensure that workstations A and B will only process orders for which, in the near future, capacity becomes available in workstations C and/or D downstream. For example, if no card B→C is available in workstation B, this means that workstation C is backlogged with work. Working on an order destined for workstation C would only increase the inventory on the shop floor, since workstation C has a lack of capacity to work on this order. It is better to process another order, for example one that needs further processing in workstation D. In this way the POLCA system balances work between workstations C and D (Suri and Krishnamurthy, 2003).

The basic POLCA system is indifferent with respect to the amount of work (in processing time units) represented by a POLCA card and is thereby a unit-based pull system. However, the original POLCA system can be transformed into a load-based version such that the number of cards in each loop is replaced by an allowable workload (in processing time units) for that loop. This system provides a more adequate and robust representation of available capacity in settings in which the processing times of production orders vary significantly and product mix changes occur frequently (Vandaele et al., 2008). In this chapter we consider the basic, unit-based POLCA system.

2.3 Research Questions

In the previous section we discussed how the pull structures of POLCA, CONWIP and m-CONWIP control the workload in an MTO production system and that the TTT is a good indicator for the workload balancing capability of pull systems. Since CONWIP has no workload balancing capability, we expect that constraining the workload in a CONWIP controlled MTO production system increases the TTT of orders. In Sections 2.1 and 2.2 we explained that the m-CONWIP and POLCA systems balance the workload in the system in different ways. m-CONWIP uses multiple CONWIP loops, one for every routing in the production system, to balance the workload among the different routings in the production system. POLCA balances the workload by releasing an order based on the available capacity in the next workstation in the order's routing. Which of these two pull structures has the best performance with respect to workload balancing is one of the research questions we would like to address in this chapter.

Besides the pull structure we expect that the *configuration* of the system has a large influence on the workload balancing capability. The configuration of a pull system is defined as "the set of card numbers to be placed in the control loops" of the pull system (Gaury, 2000, p. 12). If the number of cards in each control loop is large, the workload on the shop floor is hardly constrained by any of the control loops of the pull system. This configuration can be used to represent a *push system* with immediate release, since in a push system there is no explicit contrain on the workload that can be on the shop floor (Hopp and Spearman, 2004).

Figure 2.6 shows an illustrative example of the STT and TTT performance of a pull system that has effective workload balancing capability. The points in the figure represent the STT and TTT performance of different configurations of the pull system. In the figure, the point at the right end of the curve shows the throughput time performance of a push system. Note that, at this point, the TTT and STT are equal since the OPT is zero in the push system.

When we move to the left of the curve, by reducing the number of cards in the control loops, the configurations become more constrained and the STT decreases while the OPT increases. In this example, when the configurations become more

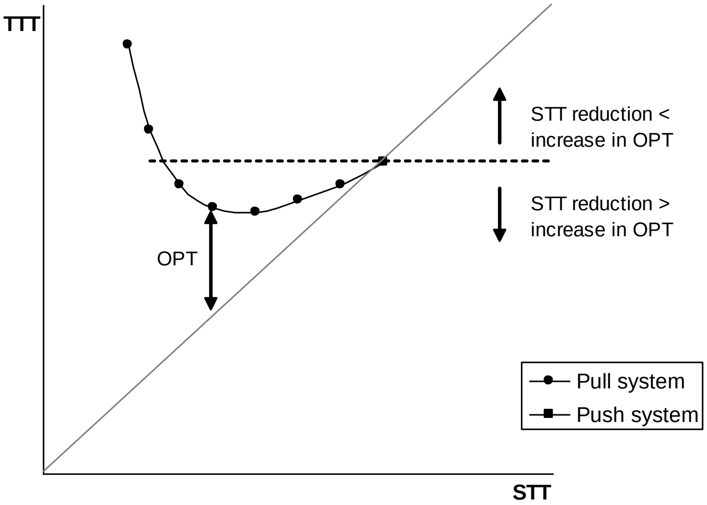

Figure 2.6: Illustrative example of effective workload balancing capability.

constrained the TTT first decreases and after a certain point the TTT increases rapidly. For the configurations below the dashed line in the figure, the reduction in STT, compared with the push system, is greater than the increase in OPT. This means that these configurations show the effective workload balancing capability of the pull system. For the configurations above the dashed line the increase in the OPT offsets the decrease in the STT. The lowest point of the curve shows the configuration for which the pull system obtains its optimal TTT performance. The difference between the TTT of this point and the TTT of the push system measures the maximum TTT reduction that can be obtained by the pull system.

We expect that an effective workload balancing capability will not only depend on the structure and configuration of the pull system. From the literature on queueing theory (see, e.g., Buzacott and Shanthikumar, 1993) we know that factors such as the order arrival pattern and the variability of the processing time of orders also have to be taken into account. Variability in the order arrival pattern increases the average queue length in front of the workstations on the shop floor and, thereby, the choice of orders in front of the control loops. We expect that this increase in choice improves the balancing capability of the control loops and reduces the blocking of the release of orders.

Variability in the processing time of orders increases the variability of the workload (in terms of processing time units) released into a control loop. This is because the pull systems that we consider in this chapter control the workload on the shop floor based on the *numbers of orders* on the shop floor and not on the *processing* times of orders. Variability of the workload in a control loop increases the variability of the workload at each workstation and, thereby, decreases the workload balance. Therefore, we expect that processing time variability has a negative impact on the workload balancing capability of the unit-based pull systems. Note that processing time variability also increases the average queue length in front of the workstations in the system and, thereby, the choice of orders in front of the control loops. We expect, however, that this positive effect on the workload balancing capability will be offset by the negative effect caused by increased variability of the workload in the control loops.

The central question of this chapter is whether m-CONWIP and POLCA can have effective workload balancing capability. We mentioned in this section the factors that we expect to influence the effective workload balancing capability of pull systems. Together with our central question, these expectations lead to the following four research questions that we address in this chapter.

(1) Does the TTT of orders increase when the workload in an MTO production system is controlled by a CONWIP system?

(2) How do POLCA and m-CONWIP perform with respect to workload balancing?

(3) What influence has the configuration of POLCA and m-CONWIP on the workload balancing capability of these pull systems?

(4) How sensitive is the workload balancing capability of POLCA and m-CONWIP to factors such as the order arrival pattern and the variability of the processing time of orders?

2.4 Experimental Design

In the previous section we formulated our research questions. We use a simulation model to analyze the these questions. In the next two subsections, we discuss the simulated production system and the performance measurements in detail.

2.4.1 Simulation Model

Figure 2.7 shows the topology of the simulated MTO production system. This type of MTO system can be denoted as a divergent segmented MTO system, where orders generally visit more than one operation (Hyer and Wemmerlöv, 2002). It perfectly suits pull systems that are able to balance workload. This specific topology of a production system enables us to identify whether m-CONWIP and POLCA can have effective workload balancing capability and whether experimental factors, such as the order arrival pattern and the variability of the processing time of orders, have a significant impact on the workload balancing capability of these pull systems.

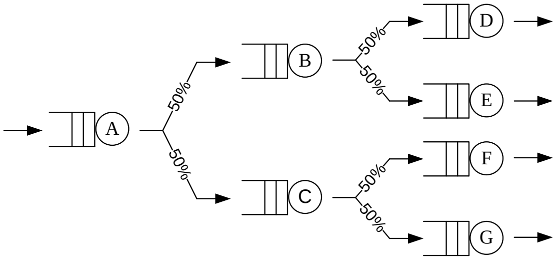

Figure 2.7: Topology of simulated production system.

The production system consists of seven workstations (A to G) and an order pool. Each workstation can handle one order at a time. The capacity of the workstations is assumed to be constant during the experiments. Each operation requires one specific workstation. Customer orders are handled in an MTO strategy. The routing of an order becomes known at the moment the order arrives and the arriving orders are uniformly distributed over the four different routings in the production system.

Workstation processing time is either deterministic or Erlang-2 distributed. To ensure that all workstations have the same utilization level we set the (mean) processing time of workstation A to one time unit, of workstations B and C to two time units and of workstations D, E, F and G to four time units. The arrival rate is such that the workstations have an average utilisation level of 80%, 85% or 90%. The inter-arrival time is either deterministic or exponentially distributed. The number of orders arriving simultaneously (batch size) can be either 1 or 10. Orders are processed on a First Come First Served (FCFS) basis at each workstation.

2.4.2 Performance Measurement

Table 2.1 summarises the experimental factors and their experimental levels that we consider in our simulation study. The distribution of the inter-arrival time, utilisation level (ρ) and the batch size (B) of orders are used as intermediate variables to measure the influence of the order arrival pattern on the workload balancing capability of the pull systems. The distribution of the processing time of orders is used as a variable to measure the influence of processing time variability on the workload balancing capability.

Table 2.1: Experimental factors.

Factor	Experimental levels
Order arrival pattern	
Inter-arrival time	Deterministic, exponential
Utilisation (ρ)	80%, 85%, 90%
Batch size (B)	1, 10
Processing time variability	
Processing time	Deterministic, Erlang-2

To generate scenarios for the simulation study we consider a full factorial design for the combinations of inter-arrival time, utilisation, batch size and processing time. For a given scenerio, we simulate the POLCA, CONWIP and m-CONWIP system and vary the number of number of cards in the control loops. We start with a large number of cards, such that the release of orders is not constrained by any of the control loops. Then we decrease the number of cards in the control loops gradually, such that the configurations of the pull systems become more constrained. Note that an identical number of cards in the CONWIP, POLCA and m-CONWIP systems does not mean that the STT in these systems is the same. For example, due to the overlapping loops, the number of cards in a POLCA configuration will generally be larger than the number of cards in an m-CONWIP configuration for a given STT level.

In each simulation experiment we determine the STT and TTT performance of the pull system. If we plot the STT performance against the TTT performance for different configurations of a pull system, we obtain a performance curve similar to that shown in Figure 6. By comparing the curves for CONWIP, m-CONWIP and POLCA for a given scenario, we can determine the difference of the throughput time performance of these pull systems.

Naturally, we are interested in the *optimal* throughput time performance of the pull systems for a given scenario. Therefore, we determine for each pull system the configuration for which the pull system obtains the shortest TTT. Determining the optimal configuration of a pull system can be a difficult task, especially when a pull system consists of multiple control loops. Gaury (2000) gives a short review of the techniques that can be used for determining the optimal configuration of a pull system. We simply use an exhaustive search to determine the optimal configuration of the pull system.

The simulation model is constructed in DESIMP, a discrete event simulation library within Delphi. DESIMP is very fast, flexible and suitable for this type of research. We use common random numbers to reduce the variance across experiments. Each experiment consists of 100 independent experiments with a run length of 100,000 time units. All experiments include a warm-up period of 25,000 time units in order to eliminate the initial transient. If we state that there is a performance difference between two experiments in the following section, the significance can be shown by a paired *t*-test at the 95% confidence level.

2.5 Results

This section presents the results of the simulation experiments. It gives an in-depth analysis of the workload balancing capabilities of CONWIP, POLCA, and m-CONWIP.

In Figure 2.8 we give a graphical representation of the throughput time performance of the pull systems for three different scenarios (a, b and c). In all three scenarios the utilisation level is 85% and the batch size is 1. Hence, the scenarios (a), (b) and (c) only differ with respect to the distribution of the inter-arrival time and processing time of orders. This allows us to understand the influence of variability in the inter-arrival time and processing time on the throughput time performance of the pull systems. In Tables 2.2 and 2.3 on page 31 and 33 we show the throughput time performance of POLCA and m-CONWIP for a broader range of experimental factors.

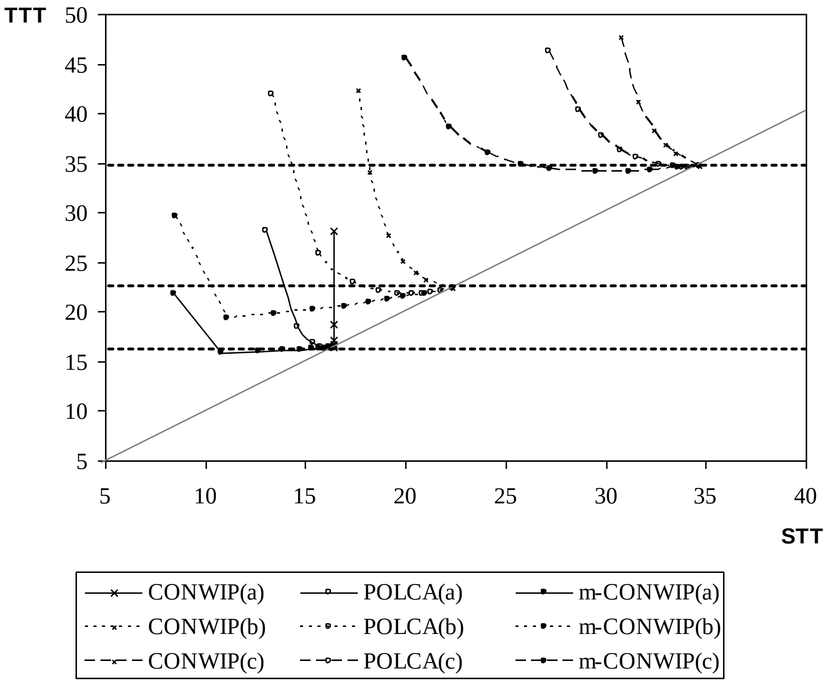

Figure 2.8: TTT and STT performance of CONWIP, POLCA and m-CONWIP. Scenario (a) deterministic inter-arrival and processing time, (b) exponential inter-arrival time and deterministic processing time, (c) exponential inter-arrival and Erlang-2 processing time.

We first consider the simulation experiments of scenario (a) in Figure 2.8. In this scenario, both the inter-arrival time and processing time of orders are deterministic. Hence, the curves CONWIP(a), POLCA(a) and m-CONWIP(a) show the throughput time performance of CONWIP, POLCA and m-CONWIP for the case where there is no randomness in the inter-arrival and processing time of orders. Figure 2.8 shows that when the configuration of CONWIP(a) becomes more constrained (i.e. when we move from right to left along the curve), the TTT increases immediately. This result confirms our expectation concerning the throughput time performance of CONWIP (see Section 2.3): because the CONWIP system has no workload balancing capability, any reduction in the STT obtained by constraining the workload on the shop floor is offset by an increase in the OPT. When the configurations of m-CONWIP(a) and POLCA(a) become more constrained, the TTT first decreases (only slightly for POLCA(a), see also Table 2.2) and after a certain point the TTT increases rapidly. POLCA(a) reaches this point much earlier than m-CONWIP(a) and this means that m-CONWIP obtains a better performance in terms of STT and TTT than POLCA for scenario (a). Note that because both pull systems are able to reduce both the TTT and STT, the workload balancing capability of POLCA and m-CONWIP is effective.

In scenario (b), the inter-arrival time of orders is exponential, while the processing time of orders is deterministic. The introduction of randomness in the inter-arrival time of orders in scenario (b) naturally increases the STT and TTT for all pull systems compared with scenario (a). However, the relative performance of the pull systems does not alter. If we compare the curves m-CONWIP(a) and POLCA(a) with the curves m-CONWIP(b) and POLCA(b), we see that an increase in the variability of the inter-arrival times results in larger TTT reductions that can be realised by POLCA and m-CONWIP. This confirms our expectation that an increase in the average queue length in front of the workstations in the production system improves the workload balancing capability of the pull systems.

In scenario (c), the inter-arrival time of orders is exponential and the processing time is Erlang-2. Introducing variability in the processing time of orders has a strong negative effect on the workload balancing capability of m-CONWIP and POLCA, as can be seen from a comparison of scenarios (b) and (c) in Figure 2.8. For scenario (c), the POLCA system has no effective workload balancing capability, while the optimal throughput time performance of m-CONWIP is only slightly below that of the push system (see also Table 2.2). This result confirms our expectation that increased processing time variability has a negative impact on the workload balancing capability of m-CONWIP and POLCA.

In Tables 2.2 and 2.3 we show the optimal throughput time performance of POLCA and m-CONWIP in terms of the percentage TTT and STT reduction these two pull systems achieve in their optimal configuration, relative to, respectively, the TTT and STT of the push system. Recall from the previous section that the optimal configuration of a pull system is defined as the configuration for which the pull system obtains the shortest TTT. Note that we have omitted the performance of the CONWIP system in these tables since the CONWIP system has no effective workload balancing capability and, therefore, the optimal configuration of the CONWIP system is equal to the push system.

Table 2.2 contains the optimal throughput time performance of m-CONWIP and

POLCA for different scenarios, given the restriction that the same number of cards is used in every control loop. The curves in Figure 2.8 already indicate a relationship between the variability in the inter-arrival and processing time of orders and the effective workload balancing capability of the pull systems. The results in Table 2.2 confirm that increased variability in the inter-arrival time of orders improves the workload balancing capability of POLCA and m-CONWIP. Table 2.2 also confirms the negative influence of increased processing time variability on the workload balancing capability of POLCA and m-CONWIP.

Table 2.2: Throughput time performance of m-CONWIP and POLCA. The percentages show the TTT and STT reduction obtained by m-CONWIP and POLCA in their optimal configuration relative to the TTT and STT of the push system. The configurations of m-CONWIP and POLCA are optimal given the restriction that the same number of cards is used in each control loop.

		Deterministic inter-arrival time				Exponential inter-arrival time			
		POLCA		m-CONWIP		POLCA		m-CONWIP	
B	ρ	%TTT	%STT	%TTT	%STT	%TTT	%STT	%TTT	%STT
Deterministic processing time									
1	80%	0.47	1.14	1.56	23.09	2.23	8.56	10.13	39.34
	85%	0.71	1.43	3.23	34.30	2.73	9.13	13.40	50.40
	90%	1.09	2.49	6.56	50.12	3.31	11.61	17.31	63.71
10	80%	1.15	9.17	6.37	35.69	5.22	30.31	19.65	75.63
	85%	1.05	5.10	8.21	44.30	5.47	38.42	21.66	81.35
	90%	1.21	4.81	10.68	56.49	5.67	43.25	23.54	87.09
Erlang-2 processing time									
1	80%	0.00	0.00	0.00	0.00	0.00	0.00	1.44	11.56
	85%	0.00	0.00	0.00	0.00	0.00	0.00	1.71	14.80
	90%	0.00	0.00	0.00	0.00	0.00	0.00	2.38	18.79
10	80%	0.00	0.00	0.00	0.00	0.00	0.00	4.39	47.33
	85%	0.00	0.00	0.00	0.00	0.00	0.00	4.94	48.46
	90%	0.00	0.00	0.00	0.00	0.00	0.00	5.25	52.94

Table 2.2 further shows that the other intermediate variables of the order arrival pattern, the utilisation level and batch size, influence the TTT reduction that can be realised by m-CONWIP and POLCA. For instance, given a deterministic inter-arrival and processing time and a batch size of 1, the TTT reduction for m-CONWIP increases from 1.56% to 6.56% if the utilisation increases from 80% to 90%. In general, we see from Table 2.2 that the percentage of TTT reduction obtained by m-CONWIP increases with increasing utilisation level. This means

that, for m-CONWIP, workload balancing has more of an effect for higher levels of utilisation. Land (2004) shows that the same relationship between utilisation and workload balancing exists for load-based pull systems. For the POLCA system we see that this relationship does not hold for the scenarios with deterministic inter-arrival and processing time and a batch size of 10. The exception to the rule is caused by the restriction we put on the number of cards that can be used in each control loop. This restriction reduces the set of allowable configurations for POLCA. As can be seen from Table 3 the relationship between utilisation and workload balancing holds for POLCA if we consider all configurations of POLCA.

Table 2.2 also shows that the batch size has a large effect on the TTT performance of m-CONWIP and POLCA. For instance, given a deterministic inter-arrival and processing time and a utilisation level of 80%, the TTT reduction for m-CONWIP increases from 1.56% to 6.37% as a result of the increased batch size. Utilisation and batch size both increase the average queue length in front of the workstations in the system, and thereby the choice of orders in front of the control loops. The results in Table 2.2 again confirm our intuition that an increase of choice in orders improves the balancing capability of the pull systems and reduces the blocking of the release of orders.

Table 2.3 contains the optimal throughput time performance of m-CONWIP and POLCA, without any restriction on the number of cards used in the control loops. We have excluded from this table the experiments with Erlang-2 processing times, because for these experiments the POLCA system has no effective workload balancing capability. Note that the optimal configuration for the m-CONWIP system does not change after relaxing the restriction on the number of cards. For the POLCA system, however, the optimal configuration has changed. In the optimal configuration of POLCA, the number of cards in the loops A→B and A→C is infinite, which means in fact that the workload released to the shop floor is not constrained. Note that the optimal POLCA configuration is therefore not consistent with the definition of a pull system. The infinite number of cards in the loops A→B and A→C results in a POLCA configuration that does not use overlapping loops (see Section 2.2.2) to balance the workload on the shop floor. This remarkable result implies that the control loops B→D, B→E, C→F and C→G are completely responsible for the effective workload balancing capability of POLCA.

Table 2.3: Throughput time performance of m-CONWIP and POLCA. The percentages show the TTT and STT reduction obtained by m-CONWIP and POLCA in their optimal configuration relative to the TTT and STT of the push system. With respect to the optimal configuration, there is no restriction on the number of cards used in a control.

		Deterministic inter-arrival time				Exponential inter-arrival time			
		POLCA		m-CONWIP		POLCA		m-CONWIP	
B	ρ	%TTT	%STT	%TTT	%STT	%TTT	%STT	%TTT	%STT
Deterministic processing time									
1	80%	3.72	3.72	1.56	23.09	6.12	6.12	10.13	39.34
	85%	5.57	5.57	3.23	34.30	7.76	7.76	13.40	50.30
	90%	8.30	8.30	6.56	50.12	9.69	9.69	17.31	63.71
10	80%	3.36	3.36	6.37	35.69	6.31	6.31	19.65	75.63
	85%	4.56	4.56	8.21	44.30	6.84	6.84	21.66	81.35
	90%	6.77	6.77	10.68	56.49	7.34	7.34	23.54	87.09

2.6 Conclusions

For MTO companies, short average total throughput time (TTT) is of strategic importance for winning orders. Pull systems aim to reduce the throughput times by controlling the workload on the shop floor. Constraining the workload on the shop floor reduces the average time orders spend on the shop floor (STT) compared with the unconstrained production system. However, the restricted release of orders onto the shop floor increases the average time orders spend waiting before being released (i.e., the average order pool time or OPT) due to the blocking of the release of orders. As a result, the reduction in STT might be offset by the increase in the OPT. The literature on workload control shows that a reduction in the TTT can only be obtained if the release mechanism not only reduces the workload, but also improves the balance of the workload on the shop floor. This literature, however, shows the existence of an effective workload balancing capability only in *load-based* pull systems. The problem we addressed in this chapter is whether *unit-based* pull systems, that are easier to understand and thereby easy to implement in practice than load-based systems, can also improve the workload balance on the shop floor such that both the STT and TTT are reduced.

To obtain insight into this problem we used simulation to analyse the throughput time performance of three unit-based pull systems that are considered applicable in MTO environments: POLCA, CONWIP and m-CONWIP. The results of our simulations show that unit-based pull systems can reduce both the TTT and STT, and that the magnitude of the reduction is dependent on the pull structure, on the configuration of the pull system and on the order arrival pattern and processing time variability of the orders.

The pull structure of CONWIP has no workload balancing capability and our simulation results show that constraining the workload in a CONWIP controlled MTO production system increases the TTT of orders. The overlapping loops in the POLCA system bring forward some workload balancing capability compared with CONWIP, but these loops do not perfectly detect and signal an imbalance in workload. As a result, POLCA faces a longer TTT for a given STT than m-CONWIP, the system with the best workload balancing capability.

Our results further show that the configuration of the pull system has a large influence on the workload balancing capability. When the configurations of a pull system become more constrained, the STT decreases, while the OPT increases. If the pull system has effective workload balancing capability, constraining the amount of work on the shop floor will first result in a STT reduction (compared with the unconstrained system) that outweighs the increase in OPT. After a certain point the configurations become too constrained and the OPT increases rapidly and finally overcompensates the decrease in STT.

Our simulation studies show that an increase in the choice of orders in front of the control loops of POLCA and m-CONWIP improves the workload balancing capability of these pull systems. Such an increase in the choice of orders can occur, for example, due to an increase in the variability of the inter-arrival times of orders. Variability of the workload in a control loop increases the variability of the workload at each workstation and, thereby, decreases the workload balance. This is because the pull systems that we consider in this chapter control the workload based on the *number of orders* on the shop floor and not on the *processing time* of orders, i.e. are unit-based instead of load-based. Our simulation results show that processing time variability has a large negative impact on the workload balancing capability of the unit-based pull systems considered in this chapter.

Although this chapter shows that unit-based pull systems can reduce both the TTT and STT, our simulation studies also show that as soon as the manufacturing conditions become more realistic (i.e. when variability in the processing times of orders is introduced), the increase in the OPT off-sets the decrease in the STT. An important issue that requires additional study is whether the throughput time performance of POLCA and m-CONWIP can be improved if the release of orders is load-based instead of unit-based. A remarkable result of this chapter shows that, in the optimal configuration of POLCA, the final control loops are fully responsible for the effective workload balancing capability of POLCA. An interesting issue for future research is whether this result can also be found in production systems with different topologies than that considered here.

Chapter 3

Order Acceptance: State-Dependent Bulk Queues

We showed in Chapter 2 that unit-based release policies can reduce the total throughput time in a make-tor-order production system by balancing the workload, but the magnitude of the effect is rather small. In the second part of the thesis we therefore focus on a stronger instrument to control delivery times while making good use of the available capacity: order acceptance.

In this chapter we propose a queueing approach for studying the performance of simple order acceptance policies for a make-to-order production system in which orders arrive and are served in batches by a single machine. To determine optimal system configuration and good order acceptance policies, we model the production system as a batch arrival batch service (bulk) queue with restricted accessibility. Such queueing systems have rich applications in manufacturing, service operations, computer and telecommunication systems.

Our principle result in this chapter is the development of a unifying method to study the performance of a general class of queueing systems that covers many bulk queueing systems with restricted accessibility as special cases. We use semi-regenerative analysis to develop a numerically stable method for calculating the limiting probability distribution of the queue length process. Based on the limiting probabilities, we present various performance measures for evaluating admission control and batch service policies, such as the loss probability for an arriving group of customers and for individual customers within a group. We demonstrate our method by means of numerical examples.

3.1 Introduction

Group arrival and batch service queues (usually called *bulk queues*) have many applications in manufacturing, service operations, computer and telecommunication systems. Since most of these systems have finite buffer capacity, it is of interest to study queueing systems with finite queue size. For example, in manufacturing systems, there is limited waiting room before workstations in assembly lines, material handling systems, or cellular manufacturing cells. In service systems such as facilities, there are limited circulation systems (elevators, stairways, and corridors) and finite storage areas (MacGregor Smith and Cruz, 2005). Finally, in computer and telecommunication systems, routers and switches that regulate the transmission of information packages have finite buffer capacity.

In many of these applications the arrival and service rate depend on the state of the queue. For example, a long queue can "discourage" arriving customers (Dshalalow, 1997) leading to queue-length dependent balking. Another example consists of systems where the server is a human being and the perception of the workload may directly influence the server's productivity (Bekker, 2004, Bekker et al., 2004). Besides the arrival and service rate, the size of arriving group and service batches may be queue length dependent. For instance, when the queue length hits the maximum buffer capacity, a situation can occur that a newly arriving group of customers does not find enough room in the queue and that a part of the group has to be refused from entering the system. Furthermore, service batch sizes are typically determined by the capacity of the server (i.e. the maximum number of customers that can be served simultaneously) and the number of customers waiting in queue, e.g., in the serving of people by elevators, shuttle buses, and ferries. Finally, the batch service time can also depend on the batch size; typically larger batches require more service time. In all these applications, it is helpful to be able to compute relevant performance measures, such as average time in system, moments of the number of customers in queue and loss probabilities for arriving groups of customers, or individual customers within a group. This allows operators to determine optimal system configuration, good admission control policies, or optimal batch sizing policies.

In this chapter we develop a simple, numerically stable, and efficient algorithmic method that allows the performance evaluation of a general queueing system that contains all of the above examples as special cases. The queueing system that we study for our purpose is the finite-buffer state-dependent bulk queue: $M(n)^{X(n)}/G(n)^{Y(n)}/1/K+B$. Here $M(n)$ and $G(n)$ correspond to the state-dependent arrival and batch service processes, the exponents $X(n)$ and $Y(n)$ represent the (random) state-dependent sizes of the arriving groups and service batches, the capacity of the queue is limited by K, and, finally, the maximal service capacity is B. The formal analysis of this queueing system is considered an open problem in the queueing literature (Dshalalow, 1997) and thereby our research makes a start to fill a gap in literature. To do so, we use a semi-regenerative analysis to obtain the limiting probabilities of the queueing process, which in turn allows the computation of many performance measures relevant for selecting the best system configuration.

The chapter is organized as follows. In Section 3.2 we provide applications of the finite-buffer state-dependent bulk queue and review literature related to the analysis of the model. After introducing the $M(n)^{X(n)}/G(n)^{Y(n)}/1/K{+}B$ model in Section 3.3 we illustrate in Section 3.4 how various special cases and applications are covered by the model. In Section 3.5 we present the semi-regenerative analysis of the model and obtain the limiting probabilities in terms of recurrence relations. Section 3.6 presents the algorithmic aspects of our solution method and Section 3.7 defines various performance measures of the model. In Section 3.8 we use numerical examples to demonstrate our method.

3.2 Applications and Literature Review

Before reviewing related literature, we sketch two practical scenarios leading to bulk queue models with finite buffers and state-dependent arrival or service processes.

A typical batch-wise process in the aircraft industry concerns the hardening of synthetic parts (Hodes et al., 1992, Van der Zee et al., 2001). These parts arrive in groups from preceding manufacturing steps and are hardened in an oven in a batch-wise manner. Upon arrival the parts enter a buffer where they wait until they are loaded into the oven. The maximum time parts can stay in the buffer is limited due to strict quality constraints. In particular, if parts stay more than T time units in the buffer, the products become worthless for any further use. The time limit is operationalized by constraining the capacity of the buffer to K parts. Furthermore, service batch sizes are limited by the physical size of the oven, and processing times (including preparation times) are independent of the number of parts in a batch. Once processing has started, no interruption is allowed, i.e. no addition or extraction of parts is possible during the production process. Given these characteristics, a control policy is required that determines, once a service batch is finished, when to start a new batch service in such a way that logistical costs and product loss are minimized and a given service level is reached. This process can be modeled as a finite-buffer state-dependent bulk queue. The arrival group sizes correspond to the synthetic parts which are state-dependent due to the finite capacity of the buffer. A service batch corresponds to the parts that are hardened in the oven in a batch-wise manner and the service batch size is also dependent on the number of parts waiting in the buffer. Other production systems that possess more or less similar characteristics are ovens that are used for the diffusion/oxidation process in the manufacture of semiconductor wafers (Fowler et al., 1992, Uzsoy et al., 1994) and the burn-in operation of a manufacture of medical diagnostic units (Hopp and Spearman, 2008).

Bulk queueing systems are also often found in transportation since mass transit vehicles are natural batch servers to which passengers arrive in groups of varying size. Furthermore, arriving passengers may decide to take another mode of transport when the queue length becomes excessive which makes state-dependent arrival rates a realistic assumption. The single server system is generally found in the form of a shuttle between two or more campuses of an institution, see e.g. (Deb, 1978, Weiss, 1979). The travel time does not depend on the number of passengers aboard and the fixed travel cost is only minimally affected by the number of passengers carried. Given these characteristics, an operating policy is required that determines when to dispatch the shuttle such that service cost and passenger waiting time are minimized. The state-dependent finite-buffer bulk queue is a reasonable model to evaluate dispatching rules for the shuttle bus problem.

Besides for practical examples, the formal analysis of the $M(n)^{X(n)}/G(n)^{Y(n)}/1/K{+}B$ model is also theoretically a challenging problem. There is a long tradition in the development of algorithmic methods for computing the limiting probabilities of generalizations of the $M/G/1/K$ queueing process, e.g. cf. Neuts (1977) and Takagi (1993). The $M/G/1/K$ queue with state-dependent arrival and service rates was first analyzed by Courtois and Georges (1971) using the embedded Markov chain approach. However, as Gupta and Rao (1998) pointed out, the method presented by Courtois and Georges (1971) is numerically unstable. A stable recursive algorithm for computing the limiting probabilities of the $M/G/1/K$ queue with state-dependent arrival rates has been given by Tijms and Van Hoorn (1981). Schellhaas (1983) and Gupta and Rao (1998) generalized the model of Tijms and Van Hoorn (1981) by allowing state-dependent service times, using respectively a semi-regenerative approach and the supplementary variable method.

Comparatively less work has been done to introduce state dependencies into finite-buffer $M/G/1$ bulk queues. In the survey on queueing systems with state-dependent parameters, Dshalalow (1997) mentions that it is still an open problem to generalize the state-dependent $M/G/1/K$ model for group arrivals and batch services. In recent years, however, significant contributions have been made to the development of algorithmic methods for computing the limiting probability of $M^X/G^Y/1/K+B$ bulk queues under various rejection policies, cf. Nobel (1989), Dudin et al. (2005), Chang et al. (2004) and Germs and Van Foreest (2010). Also the literature on queueing models with different types of batch service policies has grown over the years. In Medhi (2003) and Chaudry and Templeton (1983) a comprehensive treatment of bulk queues with batch service can be found. However, in all of the aforementioned research on bulk queues, none of the input or service parameters of the queueing models are state-dependent.

3.3 Model

We consider a single server queue at which groups of customers arrive according to a state-dependent Poisson process with finite rate λ_i when the queue contains i customers. We note that λ_i denotes the rate at which groups of customers *arrive*; note that, due to the finite capacity K of the queue, the arrival rate can be different from the rate at which groups of customers are *accepted.* The sizes of the *arriving* groups form a sequence of independent integer random variables, distributed as the generic random variable X_i with probability mass function $\mathrm{P}\{X_i = k\} = x_i(k)$, $k \geqslant 1$. Here and in the sequel, the subscript i will always refer to the dependence on the queue length (number of customers waiting for service) at the moment of customer arrival or service completion (the context will always clarify which of the two cases apply).

Due to the limited capacity of the queue, it can occur that a newly arriving group does not find enough room in the queue. As a consequence, a decision has to be made which part of the group is to be refused from entering the system. Hence, dependent on the *rejection policy* in use and the queue length, the distribution of the size of an *accepted* group may differ from the distribution of the size of an *arriving* group. Let the sizes of the accepted groups be distributed as the generic random variable $\hat{X}_i$ with $\mathrm{P}\{\hat{X}_i = k\} = \hat{x}_i(k)$, $k \geqslant 0$. We refer to Section 3.4 for examples that illustrate how to define the $\hat{x}_i(k)$ for various rejection policies.

Customers are served in FCFS order in service batches. Service batch sizes are independent integer random variables, distributed as the generic random variable Y_i with distribution $\mathrm{P}\{Y_i = k\} = y_i(k)$, for $k = 0, \ldots, B$, where B is the maximal server capacity. Here $y_i(0)$ denotes the probability that no customers are taken into service and that, as a consequence, the server enters an idle period. A situation in which it is reasonable to keep the server idle while there are customer waiting in queue is when the aim is to minimize average waiting time of customers in the system. In fact, (Aalto, 2000, Deb and Serfozo, 1973) prove that it is optimal to start serving customers only when the number of customers in queue exceeds some threshold a. Note that $y_i(k) = 0$ if $k > i$, since it is impossible to take k customers into service when there are only $i < k$ customers in queue. We assume that any arrival during a service joins the queue, if accepted. (Thus, if a group arrives to find $k > 0$ customers in service, the group cannot join the batch already undergoing service.) Batch service times $S_{i,k}$ are assumed to be independent of the arrival process, but may depend on the service batch size k and on the queue length i, and form a set of independent random variables distributed as $G_{i,k}(s) = \mathrm{P}\{S_{i,k} \leqslant s\}$. We assume $\mathrm{E}\left(S_{i,k}\right) < \infty$ for all i, k.

3.4 Special Cases

In this section we illustrate that $M(n)^{X(n)}/G(n)^{Y(n)}/1/K+B$ model covers a large class of well-known finite-buffer single server queueing models. The models are loosely ranked in order of complexity. As later models are in most cases extensions of previous models, we only specify the parameter settings in which these models differ from the previous models.

We extend the finite-buffer single server model mainly in two directions: different service batching policies, and rejection (blocking) policies. We choose to implement the service (rejection) policies by means of specific choices for $y_i(k)$ ($\hat{x}_i(k)$).

3.4.1 $M/G/1/K+1$ Queue

This queue is the base model for the other models and can be derived by taking $\lambda_i = \lambda$, $x_i(1) = 1$ and $G_{i,k}(\cdot) = G(\cdot)$, for $i, k \geqslant 0$, in the $M(n)^{X(n)}/G(n)^{Y(n)}/1/K+B$ model. Since customers are blocked when K customers are in queue it follows that $\hat{x}_i(1) = 1$ if $i < K$ and $\hat{x}_i(0) = 1$ if $i \geqslant K$. Observe that the server does not idle if the queue is not empty and serves the customers one by one. Therefore, $y_0(0) = 1$ and $y_i(1) = 1$ for all $i > 0$.

3.4.2 $M/G^Y/1/K{+}B$ Queue with Random Batch Service

In this model the server has a random capacity Y. The actual number of customers accepted in a given service period equals the whole queue, or the current batch capacity, whichever is less (see Bagchi and Templeton (1973)). To implement the policy, we set

$$y_i(k) = \begin{cases} \mathrm{P}\{Y = k\}, & \text{if } k < i, \\ \sum_{k=i}^{\infty} \mathrm{P}\{Y = k\}, & \text{if } k = i, \\ 0, & \text{otherwise.} \end{cases}$$

A practical example of the random batch-service policy can be found in the semiconductor industry, where it is frequently observed that circuit boards are processed in random batches (Hochbaum and Landy, 1997).

3.4.3 $M/G^{[a,b]}/1/K{+}b$ Queue with Minimal Batch Service

With the *minimal batch service* policy the server only serves batches of size at least a and not larger than b, that is, $\mathrm{P}\{Y_i = \min\{i, b\}\} = 1$ only when $i \geqslant a$. To implement the policy, we set $y_i(0) = 1$ if $i < a$ and $y_i(k) = 1\{k = \min\{i, b\}\}$ if $i \geqslant a$.

Deb and Serfozo (1973) show that the minimal batch service policy is optimal for a batch service queue where costs are incurred for serving the customers and for holding them in the system. Aalto (2000) generalizes the result to queueing systems with compound Poisson arrivals. Note that if the cost of serving is set to zero, minimizing the expected averaged cost is equivalent to minimizing the average waiting time. Applications of this batch service policy are abundant and can be found in the serving of people by elevators, ferries, and shuttle buses; the transhipment of mail, and military supplies; the processing of computer programs, job applications and library books; and the production, inventory control and shipment of commercial products (Deb and Serfozo, 1973).

3.4.4 $M/G^{[b,b]}/1/K{+}b$ Queue with Full Batch Service

The *full batch service policy* is contained in the previous model by setting $a = b$.

3.4.5 $M^X/G/1/K{+}1$ Queue with Partial Acceptance

Since the queue length is bounded, and group sizes may be larger than 1 we need to decide how to handle arriving groups whose size exceeds the free capacity. In case of *partial acceptance*, whenever the size of the arriving group and the queue length i at an arrival epoch exceed K, only the part of the batch that fits into the buffer is accepted (i.e. $K - i$ customers). Hence, for $i < K$

$$\hat{x}_i(k) = \begin{cases} x_i(k), & \text{if } i + k < K, \\ \sum_{l \geqslant k} x_i(l), & \text{if } i + k = K, \end{cases}$$

and $\hat{x}_i(0) = 1$ for $i = K$.

This policy has many application in manufacturing, service, computer and telecommunication system as the partial batch acceptance policy utilizes the buffer space in an optimal manner so that the loss probability of customers is rather low.

3.4.6 $M^X/G/1/K{+}1$ Queue with Complete Rejection

In a make-to-order situation where a group of customers represents a batch of products belonging to one order, it is often not possible to allow partial acceptance of individual products. The same holds for telecommunication systems where a group of customers is interpreted as a set of packages belonging to one information unit (Dudin et al., 2005). For these situations it is more realistic to select the *complete rejection* or the *complete acceptance* admission policy.

Under the complete rejection policy the complete arriving group is rejected if its size exceeds the available buffer space. It is not difficult to see that the distribution of $\hat{X}_i$ for the complete rejection model is given by $\hat{x}_i(k) = x_i(k)1\{i + k \leqslant K\}$, for $k \geqslant 1$. Observe that under the complete rejection policy, $\hat{x}_i(0)$ is the probability that at an arrival epoch all customers in the group are rejected. Hence, $\hat{x}_i(0) = \sum_{k>K-i} x_i(k)$.

3.4.7 $M^X/G/1/K{+}1$ Queue with Complete Acceptance

In situations where customers arrive in large groups the complete rejection policy has a rather high loss probability. The complete acceptance policy may provide in these cases a much better performance. Under this policy, a group is completely accepted whenever part of it can be accepted and therefore $\hat{x}_i(k) = x_i(k)$ if $i < K$ and $\hat{x}_i(0) = 1$ if $i \geqslant K$.

The complete acceptance discipline suggest a presence of some additional place for admitting a whole group which can not be completely placed into the buffer. This is however not a problem in many real life systems. For instance, if we model a computer system we can consider RAM (Random Access Memory) as a finite buffer. In case of buffer overflow, the information that does not fit into the RAM can be placed into extended or expanded memory (Dudin et al., 2005).

3.4.8 $M(n)/G(n)/1/K$ Queue

In this model, the arrival and service process are dependent on the number of customers in the system (i.e. the number of customers in the queue plus the one in service in case the server is busy). The model, and some special cases of it (e.g. the machine repairman problem), has been discussed extensively by Schellhaas (1983) and Gupta and Rao (1998). As we will discuss in Remark 3.5.3, we can let the arrival rate depend on the status of the server by replacing the λ_i in the model by $\lambda_{i,l}$, where $l = 1$ if the server is busy and $l = 0$ otherwise. Now, let the index m denote the number of customers in the system, then we can cover the $M(n)/G(n)/1/K$ queue by defining $\lambda_{0,0} = \lambda_0$, if $m = 0$, and $\lambda_{m-1,1} = \lambda_m$, if $m > 0$.

3.5 Semi-Regenerative Analysis of the Model

We start with characterizing the state of the $M(n)^{X(n)}/G(n)^{Y(n)}/1/K{+}B$ queue and defining the limiting probabilities of the queue length process. Next, we derive a procedure to compute these limiting probabilities.

3.5.1 Preliminaries

To characterize the state of the $M(n)^{X(n)}/G(n)^{Y(n)}/1/K{+}B$ system at an arbitrary point in time t, we need to specify both the queue length and the server state at t. To see this, note that in the present model the service policy may idle the server even when customers are present in queue. Therefore, knowing the number of customers in queue at time t is not sufficient to determine whether the server is idle or busy at t. Let the queue length process $\{Q(t), t \geqslant 0\}$ take values in the finite set $E \subset \mathbb{N}$, while the busy process $\{B(t), t \geqslant 0\}$ takes values in $\{0, 1\}$, so that $B(t) = 1$ when the server is busy at time t and $B(t) = 0$ otherwise. The system is now characterized by the right continuous, bi-variate process $\{Q(t), B(t)\}$, which is assumed to have left limits in t.

The server observes the queue length at service completion epochs and at arrival epochs of customers when the server is idle. Let $0 = T_0 < T_1 < T_2 < \cdots$ be the ordered sequence of these epochs, and let $\{Q_n, n \geqslant 0\}$ denote the (embedded) queue length process as observed by the server at these times, that is, we define

$$Q_n = \begin{cases} Q(T_n-), & \text{if } T_n \text{ is a service completion epoch,} \\ Q(T_n-) + \hat{X}_{Q(T_n-)}, & \text{if } T_n \text{ is an arrival epoch and the server is idle.} \end{cases}$$

Thus Q_n is *either* the queue length just before service completion *or* the queue length just after the acceptance of (part of) the group of customers. Then it is clear (although it requires some technical arguments, see e.g. Çinlar (1975) or Asmussen (2003)) that $\{Q_n, T_n\}$ is a Markov renewal process embedded in $\{Q(t), B(t)\}$, so that $\{Q(t), B(t)\}$ is a semi-regenerative process. This means that for any n the conditional distribution of $\{Q(t + T_0 + \cdots + T_n), B(t + T_0 + \cdots + T_n)\}_{t \geqslant 0}$ given $T_0, \ldots, T_n$, $Q_0, \ldots, Q_n = i$ is the same as the conditional distribution of $\{Q(t), B(t)\}$ given $T_0 = 0$ and $Q_0 = i$. Hence, to characterize the conditional distribution of $\{Q(t), B(t)\}$ it suffices to specify the behavior $Q(t)$ and $B(t)$ on the interval $[T_0, T_1)$. Let $T_0 = 0$ and $Q_0 = i$, then $\{Q(t), B(t)\}$ must satisfy for $t \in [0, T_1)$,

$$B(t) = 1\{Y_i > 0\}, \tag{3.1a}$$

where $1\{A\}$ is the indicator function of the set $\{A\}$, and

$$Q(t) = \begin{cases} i, & \text{if } Y_i = 0, \\ i - Y_i + Z_{i-Y_i}(t), & \text{if } Y_i > 0, \end{cases} \tag{3.1b}$$

since if $Y_i = 0$ the server remains idle during $[0, T_1)$ and if $Y_i > 0$ it takes a batch of size Y_i into service while the random variable $Z_i(s)$ represents the number of *accepted* arrivals during $[0, s]$ given that at the start of the interval the queue length is i and the server is busy. Note that $Q(t) \geqslant 0$ for all $t \geqslant 0$, since $y_i(k) = 0$ if $k > i$, and $\{Z_i(s)\}$ is a pure birth process.

We assume that the embedded Markov chain $\{Q_n\}$ with state space E is irreducible and aperiodic and that the Markov renewal process $\{Q_n, T_n\}$ is aperiodic. Since E is also finite, it follows that $\{Q_n\}$ is positive recurrent.

Assuming that all these conditions are satisfied, the limiting distributions π of the embedded Markov chain $\{Q_n\}$ and p of the semi-regenerative process $\{Q(t), B(t)\}$ exist. That is, for $j \in E$,

$$\begin{aligned}
\pi_j &= \lim_{n\to\infty} \mathrm{P}\{\text{at time } T_n,\ j \text{ customers wait in queue}\} \\
&= \lim_{n\to\infty} \mathrm{P}\{Q_n = j\} \\
p_{j,0} &= \lim_{t\to\infty} \mathrm{P}\{\text{at time } t,\ j \text{ customers wait in queue and the server is idle}\} \\
&= \lim_{t\to\infty} \mathrm{P}\{Q(t) = j, B(t) = 0\} \\
p_{j,1} &= \lim_{t\to\infty} \mathrm{P}\{\text{at time } t,\ j \text{ customers wait in queue and the server is busy}\} \\
&= \lim_{t\to\infty} \mathrm{P}\{Q(t) = j, B(t) = 1\}.
\end{aligned} \tag{3.2}$$

3.5.2 Analysis

We next derive a method to compute the limiting distributions of the embedded Markov chain $\{Q_n\}$ and the semi-regenerative process $\{Q(t), B(t)\}$. We start with deriving a numerically stable procedure to compute the semi-Markov kernel $H = \{H_i(j,t); i, j \in E, t \geqslant 0\}$ corresponding to the Markov renewal process $\{Q_n, T_n\}$. We recall from Çinlar (1975) or Asmussen (2003) that the elements $H_i(j,t)$ of H over E are defined as

$$\begin{aligned}
H_i(j,t) &= \mathrm{P}\{Q_{n+1} = j, T_{n+1} - T_n \leqslant t \mid Q_n = i\} \\
&= \mathrm{P}\{Q_1 = j, T_1 \leqslant t \mid Q_0 = i\}.
\end{aligned} \tag{3.3}$$

To start the computation of H we expand the definition of $H_i(j,t)$ by conditioning on Y_i;

$$H_i(j,t) = \sum_{k=0}^{i} \mathrm{P}\{Y_i = k\}\, \mathrm{P}\{Q_1 = j, T_1 \leqslant t \mid Q_0 = i, Y_i = k\}.$$

Observe that when $Y_i = 0$, T_1 corresponds to an idle period that starts with i customers in queue and ends within t time units with the arrival of a group of customers from which $j - i$ are accepted. Hence,

$$\mathrm{P}\{Q_1 = j, T_1 \leqslant t \mid Q_0 = i, Y_i = 0\} = (1 - e^{-\lambda_i t})\, \hat{x}_i(j - i).$$

Otherwise, when $Y_i = k > 0$, T_1 corresponds to a batch service of size k that starts with i customers in queue and ends within t time units during which $j-i+k$ customers are accepted. Writing $R_i(m, s) = \mathrm{P}\{Z_i(s) = m\}$ for the probability to accept m customers during a service interval of duration s that starts with i customers in queue, we have that for $k \geqslant 1$

$$\mathrm{P}\{Q_1 = j, T_1 \leqslant t \,|\, Q_0 = i, Y_i = k\} = \int_0^t R_{i-k}(j-i+k, s)\, dG_{i,k}(s).$$

Now we can expand the expression for $H_i(j, t)$ as

$$H_i(j,t) = y_i(0)(1-e^{-\lambda_i t})\,\hat{x}_i(j-i) + \sum_{k=1}^{i} y_i(k) \int_0^t R_{i-k}(j-i+k, s)\, dG_{i,k}(s). \quad (3.4)$$

From (3.4), it is obvious that it remains to find a suitable expression to compute $R_i(m, s)$. In the following lemma we present an efficient recursion for this purpose.

Lemma 3.5.1. *The probability that in a service period of duration s, m customers are accepted, given that just after the start of the service i customers are in queue, can be written as*

$$R_i(m,s) = \sum_{n=0}^{\infty} U_i(m,n) e^{-\lambda s} \frac{(\lambda s)^n}{n!}, \quad (3.5)$$

for some (finite) $\lambda \geqslant \max_{i \in E} \lambda_i$, and where $U_i(m, n)$ satisfies the following recursion for $i \in E$ and $n, m \geqslant 0$,

$$\begin{aligned} U_i(m, n+1) = U_i(m,n) &+ \frac{\lambda_{i+m}}{\lambda}\,[\hat{x}_{i+m}(0) - 1]\, U_i(m,n) \\ &+ \sum_{l=0}^{m-1} \frac{\lambda_{i+l}}{\lambda}\,\hat{x}_{i+l}(m-l) U_i(l,n), \end{aligned} \quad (3.6)$$

with initial conditions

$$U_i(m,0) = \begin{cases} 1, & \text{if } m = 0, \\ 0, & \text{if } m > 0. \end{cases}$$

Proof. Since the group inter-arrival times are exponentially distributed it follows for sufficiently small $h > 0$ that

$$\begin{aligned} R_i(m, s+h) = {} & [1 - \lambda_{i+m} h(1 - \hat{x}_{i+m}(0))] R_i(m,s) \\ & + h \sum_{l=0}^{m-1} \lambda_{i+l}\, \hat{x}_{i+l}(m-l) R_i(l,s) + o(h). \end{aligned}$$

Subtracting $R_i(m,s)$ at both sides, dividing by h, and taking the limit $h \downarrow 0$ we arrive at the Kolmogorov forward equation

$$\frac{d}{ds}R_i(m,s) = \lambda_{i+m}(\hat{x}_{i+m}(0)-1)R_i(m,s) + \sum_{l=0}^{m-1} \lambda_{i+l}\,\hat{x}_{i+l}(m-l)R_i(l,s). \quad (3.7)$$

By the finiteness of the λ_i, there exist a finite λ such that $\lambda \geqslant \max_{i\in E}\lambda_i$. Therefore, we can use the uniformization method and substitute the form

$$R_i(m,s) = \sum_{n=0}^{\infty} U_i(m,n)e^{-\lambda s}\frac{(\lambda s)^n}{n!} \quad (3.8)$$

in (3.7) for any such λ. After simplifying the result, we obtain (3.6).

The initial conditions follow from observing in (3.8) that $R_i(m,0) = U_i(m,0)$, and that $R_i(m,0) = 1\{m=0\}$. □

Remark 3.5.1. Observe that $U_i(m,n)$ can be interpreted as the probability to accept m customers given that n groups of customers arrived since the start of the service epoch and given that the number of customers in queue just after the start of the service epoch was i.

Now we have all the tools to compute the semi-Markov kernel H and obtain the transition matrix $P = \{P(i,j); i,j \in E\}$ of the embedded Markov chain $\{Q_n\}$ by taking the limit of H as $t \to \infty$. From the assumption that $\{Q_n\}$ is an ergodic Markov chain, it follows that the limiting distribution π exists and is the unique solution (up to normalization) of

$$\begin{aligned}\pi_j &= \sum_{i\in E}\pi_i H_i(j,\infty)\\ &= \sum_{i\in E}\pi_i\left[y_i(0)\hat{x}_i(j-i) + V(i,j)\right],\end{aligned} \quad (3.9)$$

where

$$V(i,j) = \sum_{k=1}^{i} y_i(k)\sum_{n=0}^{\infty} U_{i-k}(j-i+k,n)a_{i,k}(n)$$

which we obtain after substituting (3.5) for $R_i(k,s)$ in (3.4) and reorganizing so that the integrations reduce to the mixed Poisson probabilities

$$a_{i,k}(n) = \int_0^\infty e^{-\lambda s}\frac{(\lambda s)^n}{n!}dG_{i,k}(s). \quad (3.10)$$

To proceed from π to p we use semi-regeneration in the following theorem. First, let C denote the length of the interval between two successive embedded Markov points T_n and T_{n+1}. Supposing that $Q_n = i$, observe that C is a service interval of length $S_{i,k}$ when $Y_i = k \geqslant 1$, and an inter-arrival time when $Y_i = 0$. Therefore, the expected cycle time C_i is

$$\mathrm{E}\,(S_i) := \sum_{k=1}^{\infty} y_i(k) \mathrm{E}\,(S_{i,k}),$$

if a service starts with a queue length $Q_n = i$ while it is $y_i(0)/\lambda_i$ when the server idles. Hence,

$$\mathrm{E}\,(C) = \mathrm{E}\,(\mathrm{E}\,(C|Q)) = \sum_{i \in E} \pi_i \left[\frac{y_i(0)}{\lambda_i} + \mathrm{E}\,(S_i) \right].$$

Note that it may occur that $\lambda_i = 0$ for some $i \in E$. We require in such states that $y_i(0) = 1$ to prevent that i is an absorbing state. In such cases set $y_i(0)/\lambda_i \equiv 0$.

Theorem 3.5.2. *The limiting distribution p satisfies*

$$p_{j,0} = \frac{y_j(0)}{\lambda_j} \frac{\pi_j}{\mathrm{E}\,(C)}, \tag{3.11a}$$

$$p_{j,1} = \sum_{i \in E} \frac{\pi_i}{\mathrm{E}\,(C)} V_e(i,j) \tag{3.11b}$$

where

$$V_e(i,j) = \sum_{k=1}^{i} y_i(k) \sum_{n=0}^{\infty} U_{i-k}(j-i+k, n)\, a^e_{i,k}(n), \tag{3.11c}$$

$$a^e_{i,k}(n) = \int_0^\infty e^{-\lambda s} \frac{(\lambda s)^n}{n!} [1 - G_{i,k}(s)] ds. \tag{3.11d}$$

Proof. To prove (3.11) we use Çinlar (1975, Theorem 6.6.12) which states that for $j \in E$ and $l \in \{0, 1\}$

$$p_{j,l} = \frac{1}{\mathrm{E}\,(C)} \sum_{i \in E} \pi_i \int_0^\infty \psi_i(t, j, l) dt, \tag{3.12}$$

provided that $\{Q_n, T_n\}$ is an ergodic process, $\mathrm{E}\,(C) < \infty$, and the function $t \to \psi_i(t, j, l) = \mathrm{P}\{Q(t) = j, B(t) = l, T_1 > t \,|\, Q_0 = i\}$ is directly Riemann integrable for every $i, j \in E$ and $l \in \{0, 1\}$.

To check these conditions, note that the first two conditions are true by the assumptions made in Section 3.3. From (3.1), (3.4), and the fact that T_1 equals a service time $S_{i,k}$ when $Y_i = k \geqslant 1$, and an inter-arrival time when $Y_i = 0$, it follows that

$$\psi_i(t, j, l) = \begin{cases} y_i(0)\, e^{-\lambda_i t}, & \text{if } l = 0, j = i, \\ \sum_{k=1}^{i} y_i(k) R_{i-k}(j-i+k, t)[1 - G_{i,k}(t)], & \text{if } l = 1, \\ 0, & \text{else.} \end{cases} \tag{3.13}$$

It is clear that $\psi_i(t,i,0) = y_i(0)\,e^{-\lambda_i t}$ is directly Riemann integrable for any $i \in E$. From (3.13) we have that $\psi_i(t,j,1) \leqslant \max_{k \leqslant i}\{1 - G_{i,k}(t)\}$ for $i,j \in E$. Note that $1 - G_{i,k}(t)$ is directly Riemann integrable since it is non-increasing and $\int_0^\infty [1 - G_{i,k}(t)]dt = \mathrm{E}\,(S_{i,k}) < \infty$. Hence, $t \to \psi_i(t,j,l)$ is directly Riemann integrable for every $i,j \in E$ and $l \in \{0,1\}$.

For $l = 0$, (3.11a) follows directly from (3.12) and (3.13). Let $l = 1$ and $j \in E$. Then by (3.12), (3.13), (3.5),

$$\begin{aligned} p_{j,1} &= \frac{1}{\mathrm{E}\,(C)} \sum_{i \in E} \pi_i \sum_{k=1}^{i} y_i(k) \int_0^\infty R_{i-k}(j-i+k,t)[1 - G_{i,k}(t)]dt \\ &= \sum_{i \in E} \frac{\pi_i}{\mathrm{E}\,(C)} \sum_{k=1}^{i} y_i(k) \sum_{n=0}^{\infty} U_{i-k}(j-i+k,n) \int_0^\infty e^{-\lambda t} \frac{(\lambda t)^n}{n!} [1 - G_{i,k}(t)]dt. \end{aligned}$$

This proves (3.11b). □

Remark 3.5.3. So far, the arrival process may only depend on the number of customers in queue and not on the status of the server (i.e. whether the server is idle or busy) at the moment of customer arrival. Dependence of the arrival process on the status of the server can easily be included in our model at the expense of an additional index l, where $l = 1$ if the server is busy and $l = 0$ otherwise. Now we define $\lambda_{i,1}$ ($\lambda_{i,0}$) to be the rate at which customers arrive when there are i customers in queue and the server is busy (idle). In a similar way we extend the definitions of the probabilities $x_{i,l}(k)$ and $\hat{x}_{i,l}(k)$, for $l = 0,1$.

The equations needed for the computation of the limiting probabilities π and p, which we derived in this section, can now be adapted to cover the described extension. First observe that the λ_i and $\hat{x}_i(k)$ in Equation (3.6) all correspond to the arrival rates and group sizes during a busy period and therefore can be replaced by $\lambda_{i,1}$ and $\hat{x}_{i,1}(k)$, respectively. Furthermore, the $y_i(0)\,\hat{x}_i(j-i)$ part in (3.9) and λ_i in (3.11a) (and in $\mathrm{E}\,(C)$) correspond to the group size and arrival rate of customers to an idle server and can be replaced by $y_i(0)\,\hat{x}_{i,0}(j-i)$ and $\lambda_{i,0}$, respectively. With these small modifications we can generalize our model to include the dependence of the arrival process on the status of the server. In Section 3.4 we showed that this generalization enables us to study the $M(n)/G(n)/1/K$ where the number of customers in the *system* (instead of in the *queue*) is limited by K.

3.6 Algorithmic Aspects

Now we summarize the approach for computing the limiting probabilities at embedded, i.e., π_j, and arbitrary epochs, i.e., p_j, and we show how the precision of our numerical method can be specified in advance.

The numerical method that we have developed in the previous section leads to the following algorithm

Step 1 Compute (by numerical integration or if possible explicitly) the mixed Poisson probabilities $a_{i,k}(n)$ and $a^e_{i,k}(n)$ from relations (3.10) and (3.11d).

Step 2 Compute $U_i(k,n)$ for $i \in E$ and $k, n \geqslant 0$, by means of the recursion (3.6).

Step 3 Use standard numerical procedures to compute π from (3.9).

Step 4 Compute p using the relations in Theorem 3.5.2.

To compute π_j to a given precision $\epsilon > 0$ it suffices to compute $U_i(k,n)$ and the probabilities $a_{i,k}(n)$ up to some finite N_i, where

$$N_i = \min\left\{m; \sum_{n=0}^{m}\sum_{k=1}^{i} y_i(k) a_{i,k}(n) \geqslant 1-\epsilon\right\}.$$

This follows, since, c.f. (3.9),

$$V(i,j) = V_{N_i}(i,j) + e_{N_i},$$

where

$$\begin{aligned} V(i,j) &= \sum_{k=1}^{i} y_i(k) \sum_{n=0}^{\infty} U_{i-k}(j-i+k,n)\, a_{i,k}(n), \\ V_{N_i}(i,j) &= \sum_{k=1}^{i} y_i(k) \sum_{n=0}^{N_i} U_{i-k}(j-i+k,n)\, a_{i,k}(n), \end{aligned}$$

and e_{N_i} satisfies

$$e_{N_i} = \sum_{k=1}^{i} y_i(k) \sum_{n \geqslant N_i+1} U_{i-k}(j-i+k,n)\, a_{i,k}(n) \leqslant \sum_{n \geqslant N_i+1} \sum_{k=1}^{i} y_i(k) a_{i,k}(n),$$

since the probabilities $U_{i-k}(j-i+k,n) \leqslant 1$. Therefore,

$$e_{N_i} \leqslant 1 - \sum_{n=0}^{N_i}\sum_{k=1}^{i} y_i(k) a_{i,k}(n) \leqslant \epsilon.$$

Similar reasoning applies to the computation of $p_{j,1}$.

Finally, note that the computations in our approach only involve additions and multiplications of positive and bounded numbers, thereby preventing a loss a significant digits. Observe also that explicit expressions for the $a_{i,k}(n)$ and $a^e_{i,k}(n)$ can be given for the cases of deterministic and phase-type services.

3.7 Performance Measures

In this section we derive a set of relevant performance measures such as the average number of customers in queue (L_q), the average waiting time in queue (W_q), the server utilization (ρ) and the loss probability of a group of customers and of an arbitrary customer within a group. All these performance measures can be obtained from the limiting probabilities $p_{j,k}$, c.f. the definition in Equation (3.2). Since $p_{j,k}$ is the probability that the queue contains j customers and the server is in state $k \in \{0, 1\}$, $p_j = p_{j,0} + p_{j,1}$ is the limiting probability that at an arbitrary point in time j customers are waiting in queue. Now it easily follows that

$$\rho = 1 - \sum_{j\in E} p_{j,0} = \sum_{j\in E} p_{j,1} \qquad \lambda' = \sum_{j\geqslant 0} \lambda_j \, p_j \, \mathrm{E}\,(\hat{X}_j) \tag{3.14a}$$

$$L_q = \sum_{j\in E} j\, p_j \qquad W_q = \frac{L_q}{\lambda'} \tag{3.14b}$$

where λ' is the *acceptance* rate of customers.

The loss probability of a group of customers and of an arbitrary customer within a group clearly depend on the rejection policy. Common rejection policies are the ones we discussed in Section 3.4, i.e., partial acceptance, complete rejection and complete acceptance. In what follows, we discuss the computation of the loss probabilities for these three rejection policies.

3.7.1 Complete Acceptance

In the sequel, let Γ and γ, respectively, correspond to the event that a group is lost and that an arbitrary customer is lost. Then it is not difficult to check that for the complete acceptance policy

$$\mathrm{P}\{\Gamma\} = \mathrm{P}\{\gamma\} = 1 - \sum_{j=0}^{K-1} p_j .$$

3.7.2 Complete Rejection

Recall from Section 3.4 that under the complete rejection policy the probability that customers in an arriving group are rejected is $\hat{x}_i(0) = \sum_{k \geqslant K-i+1} x_i(k)$. Therefore,

$$\mathrm{P}\{\Gamma\} = \sum_{i=0}^{K} p_i \, \hat{x}_i(0). \tag{3.15}$$

To calculate the rejection probability for an arbitrary customer, we use the following renewal-theoretic result (Burke, 1975):

$$q_k := \mathrm{P}\{\text{an arbitrary customer belongs to a group of size } k\} = \frac{k\,x(k)}{\mathrm{E}\,(X)}, \tag{3.16}$$

with $x(k) = \sum_{i \in E} p_i \, x_i(k)$ and $\mathrm{E}\,(X) = \sum_{k \geqslant 1} k\,x(k)$. Define $\bar{q}_k = \sum_{m=k}^{\infty} q_m$ to be the probability that an arbitrary customer belongs to a group of size greater or equal to k. Then

$$\begin{aligned} \mathrm{P}\{\gamma\} &= \sum_{i=0}^{K} \mathrm{P}\left\{\gamma \;\middle|\; \begin{array}{l} \text{customer sees } i \text{ customers} \\ \text{in queue upon arrival} \end{array}\right\} p_i \\ &= \sum_{i=0}^{K} \mathrm{P}\left\{ \begin{array}{l} \text{customer belongs to a group} \\ \text{of size larger than } K-i \end{array}\right\} p_i \\ &= \sum_{i=0}^{K} p_i \, \bar{q}_{K-i+1}. \end{aligned} \tag{3.17}$$

3.7.3 Partial Acceptance

Under the partial acceptance policy it is preferable to interpret Γ as the event that a group of customers *overflows*, i.e. when an arriving group does not fit completely into the queue (see Nobel (1989)). Then it is easy to check that

$$\mathrm{P}\{\Gamma\} = \sum_{i=0}^{K} p_i \sum_{k \geqslant K+1-i} x_i(k).$$

To calculate the rejection probability for an arbitrary customer, we define for $k \geqslant 1$

$$\begin{aligned} \eta_k &:= \mathrm{P}\{\text{an arbitrary customer occupies the } k\text{th position in the group}\} \\ &= \sum_{m \geqslant k} \frac{x(m)}{\mathrm{E}\,(X)}, \end{aligned}$$

where the last equation follows by conditioning on the event that "an arbitrary customer belongs to a group of size k" and then using (3.16). Let $\bar{\eta}_k = \sum_{m=k}^{\infty} \eta_m$ denote the probability that an arbitrary customers occupies a position greater or equal to k in his group. Then analogous to the derivation of (3.17) we obtain

$$\mathrm{P}\{\gamma\} = \sum_{i=0}^{K} p_i \, \bar{\eta}_{K-i+1}.$$

3.8 Numerical Examples

In this section we apply the model to the numerical analysis of three examples, a batch queueing process with queue length dependent balking, a queuing process subject to holding cost and loss, and a batch arrival/service process subject to queue length dependent batch arrival sizes and batch size dependent service rates. Our code is available on request.

3.8.1 Bulk Queues with State Dependent Balking and Service Rates

Consider a single server shop. Customers require varying amounts of service. With little amount of work in the system, all customers are prepared to enter the system, but when there is a large amount of work, the 'large' customers still enter while most of the 'small' customers balk. As is commonly the case (see, e.g., (Bekker, 2004, Bekker et al., 2004)), the server increases the service rate when the queue becomes longer. We assume complete acceptance, and K to be so large that the probability of overflow is negligible.

As a concrete example, suppose that the service requirement of a large customer is 10 times that of a small customer. We model this by letting the service requirement of a small (large) customer correspond to a batch size of $k = 1$ ($k = 10$) unit. Large customers arrive at the system at rate $\lambda_l = 5$ per hour. We implement the balking behavior of the small customers by taking $\lambda_{s,i} = \max\{0, 10-i\}$. Then, take $\lambda_i = \lambda_{s,i} + \lambda_l$, and set

$$x_i(k) = \begin{cases} \lambda_{s,i}/\lambda_i, & \text{for } k = 1, \\ \lambda_l/\lambda_i, & \text{for } k = 10, \\ 0, & \text{else.} \end{cases} \tag{3.18}$$

Since we assume complete acceptance: $\hat{x}_i(k) = x_i(k)$ if $i < K$ and $\hat{x}_i(0) = 1$ if

$i \geqslant K$. Let service take place in single units, thus, $y_i(1) = 1$ for all $i > 0$ and $y_0(0) = 1$. Note that the queue length corresponds now to the workload in queue. For simplicity we assume deterministic service times. When the queue length is long, however, the employee feels more stress, and therefore works at a higher rate. This is implemented by taking $S_{i,k} \equiv (90 + i/5)^{-1}$ for all i, k.

For the case with $K = 50$ we find that the acceptance rate, see Equation 3.14a, $\lambda' = 56.1$ per hour, $\rho = 0.6119$, $L_q = 5.678$, and $\gamma = \Gamma = 0.0009$. As a simple reference we compare this system to an $M/D/1$ queue with load $\rho = (5 \cdot 10 + 10)/90 = 2/3$, which leads to $L_q(M/D/1) = 2/3$. Clearly, this value is much lower than 5.678, leading us to conclude that simpler queueing models are not accurate models for general batch queueing processes.

3.8.2 Minimal Batch Service Queues with Holding and Setup Costs

Consider a batch service system subject to setup and service costs, holding costs and rejecting costs. A natural batch service policy for this system is to start service only when the queue length exceeds some threshold a and then serve as many customers as possible. In this section we compute for this $M^X/G^{[a,b]}/1/K{+}b$ queueing process, see Section 3.4.3, the costs as a function of the threshold parameter a. We consider also three loss policies: complete rejection, complete acceptance and partial acceptance. As in Aalto (2000) we assume that the holding cost is c_h per customer in queue per unit time, a service cost $c_k + c_s\, j$ is incurred at each service epoch when j is the batch service size, and a rejection cost of c_r per unit.

Let π and p denote the limiting distribution of the queue length process at embedded and arbitrary epochs, respectively. It is easy to check that the average rejection costs per time unit under the complete acceptance (RC_{ca}), complete rejection (RC_{cr}) and partial acceptance (RC_{pa}) policy can be expressed as follows

$$\begin{aligned}
RC_{ca} &= c_r \sum_{i \geqslant K} p_i \sum_{j=1}^{\infty} x_i(j)\, j = c_r \sum_{i \geqslant K} p_i \,\mathrm{E}\,(X_i) \\
RC_{cr} &= c_r \sum_{i=0}^{K} p_i \sum_{j=K+1-i}^{\infty} x_i(j)\, j \\
RC_{pa} &= c_r \sum_{i=0}^{K} p_i \sum_{j=K+1-i}^{\infty} x_i(j)\,(j - K + i).
\end{aligned}$$

Now the average cost per time unit under the minimal batch service policy and rejection policy $\xi \in \{ca, cr, pa\}$ can be expressed as $AC_\xi(a) = HC(a) + SC(a) + RC_\xi$, where the average holding cost

$$HC(a) = c_h \sum_{i \geqslant 0} p_i \, i,$$

and average service cost per time unit

$$SC(a) = \sum_{i=a}^{B} \pi_i \, (c_k + c_s \, i) + \sum_{i=B+1}^{\infty} \pi_i \, (c_k + c_s \, B).$$

As a concrete example, suppose for all $i \in E$, $\lambda_i = 0.2$, $x_i(1) = 0.25$, $x_i(3) = 0.5$, $x_i(5) = 0.25$, $\mathrm{E}(X_i) = 3$, service is deterministic, i.e., $S_{i,k} \equiv 10$ for all k, i, $B = 10$, and $K = 10$. The parameters $\hat{x}_i(k)$ and $y_i(k)$ are defined as in Section 3.4 for the three rejection policies and the minimal batch service policy. Furthermore, the cost parameters are given by $c_h = 5$, $c_k = 10$, $c_s = 5$ and $c_r = 50$. In Table 3.1 we present the costs per unit time for different values of the threshold a.

Table 3.1: Long run average holding cost (HC), service cost (SC), rejection cost (RC) and total cost (AC) for the rejection policies Partial rejection (pr), Complete rejection (cr) and Complete admission (ca) as functions of the minimal batch service threshold a.

a	1	2	3	4	5	6	7	8	9	10
HC_{ca}	14.90	14.47	14.37	14.42	14.78	15.44	16.94	18.49	21.15	24.52
SC_{ca}	35.37	33.75	33.02	30.61	28.99	27.60	26.05	25.06	24.22	23.52
RC_{ca}	10.75	10.28	10.05	9.26	8.66	8.11	7.61	7.50	7.91	9.21
AC_{ca}	61.02	58.49	57.45	54.30	52.43	51.14	50.60	51.06	53.28	57.25
HC_{pr}	13.24	12.86	12.81	13.00	13.45	14.19	15.62	16.84	18.61	20.70
SC_{pr}	34.14	32.57	31.99	29.53	28.02	26.74	25.01	23.89	22.30	20.46
RC_{pr}	15.32	14.63	14.36	13.11	12.26	11.47	11.57	12.41	16.23	22.87
AC_{pr}	62.70	60.06	59.15	55.64	53.74	52.41	52.20	53.15	57.13	64.03
HC_{cr}	12.83	12.47	12.43	12.65	13.13	13.89	15.44	16.73	18.84	21.27
SC_{cr}	32.89	31.51	30.96	28.65	27.24	26.03	24.30	23.19	21.40	19.38
RC_{cr}	21.46	20.48	20.10	18.36	17.17	16.07	20.53	23.31	33.63	43.31
AC_{cr}	67.27	64.47	63.49	59.66	57.54	55.99	60.27	63.24	73.87	83.97

It is easy to find that the optimal minimal batch service threshold value $a^\star$ is 6 for the complete rejection policy, 7 for the partial acceptance and complete acceptance policies. Thus, the threshold value increases when the acceptance policy is less 'strict'. This is as expected, since the policy makes a trade-off between set-up costs, i.e., a cost c_k is incurred for each service interval, and the rejection costs. Setting the threshold a to a lower value increases the long run average setup costs but lowers the rejection costs.

3.8.3 Queueing at Thrill Rides at Fairs

Consider now the queueing process at a thrill ride such as the 'Freak Out' (see Wikipedia[1] for a description). Customers arrive in groups, and are served in batches. Larger groups tend to balk less quickly, as the customers in one group also take pleasure (hopefully) in each other's company. The service time of a batch depends on the batch size, since each customer in the rider requires a safety check before the ride can take off. The problem is to determine the minimal batch size, i.e., the a parameter of the previous model, that maximizes the number of persons entering, i.e., paying.

As a simple numerical illustration, suppose couples, i.e., two customers, arrive at rate $\lambda_{s,i} = \max\{0, 1 - i/14\}$ per minute, while groups of 4 persons arrive as $\lambda_{l,i} = 0.25\, 1\{i \leqslant 20\}$, where $1\{\cdot\}$ is the indicator function. Set $\lambda_i = \lambda_{s,i} + \lambda_{l,i}$, take

$$x_i(k) = \begin{cases} \lambda_{s,i}/\lambda_i, & \text{for } k = 2, \\ \lambda_{l,i}/\lambda_i, & \text{for } k = 4, \\ 0, & \text{else.} \end{cases} \tag{3.19}$$

and assume complete acceptance. The service time of a batch consists of the time of the actual ride, 2 (very long) minutes, 1 minute of loading and unloading, and 5 seconds per safety check. Assuming that the service time does not depend on the queue length, the service distribution then becomes $S_{i,k} = 3 + k/12$ minutes, where k is the batch size. Finally, the Freak Out has 16 seats, so the maximal batch size is 16.

Clearly, the revenue rate equals the rate λ_e at which customers enter the system, which is given by

$$\lambda_e = \sum_{i \in E} p_i (2\lambda_{s,i} + 4\lambda_{l,i})$$

Note that the maximal entering rate occurs when $Q(t) = 0$, which in this case becomes $2\lambda_{s,0} + 4\lambda_{l,0} = 3$ per minute.

Table 3.2 shows the dependency of λ_e on the minimal threshold parameter a.

[1] `http://en.wikipedia.org/wiki/Freak_Out_(ride)`

Interestingly, greedy service leads for this model to higher revenues than full batch service.

Table 3.2: Revenue rate λ_e as a function of the minimal batch service threshold a.

a	2	4	6	8	10	12	14	16
λ_e	2.3204	2.3239	2.3230	2.2988	2.2319	2.1148	1.9495	1.6844

Chapter 4

Order Acceptance: Setups & Strict Lead Times

A limitation of the stylized production situation we considered in Chapter 3 is that two important characteristics of many make-to-order production systems are not included: setup times and due-dates. In this chapter we therefore extend the production situation of Chapter 3 as follows. We consider the admission control and scheduling of customer orders in a production system that produces different items on a single machine. Customer orders drive the production and belong to product families, and are fully characterized by their lead time, processing time, and reward. When production changes from one family to another a setup time is incurred. Moreover, if an order cannot be accepted, it is considered lost upon arrival. The problem is to find a policy that accepts/rejects and schedules orders such that long run profit is maximized. This problem finds its motivation in batch industries in which suppliers have to realize high machine utilization while delivery times should be short and reliable and the production environment is subject to long setup times.

We model the joint admission control/scheduling problem as a Markov decision process (MDP) to gain insight into the optimal control of the production system and use the MDP to benchmark the performance of simple heuristic acceptance/scheduling policies. Numerical results show that the heuristic performs very well compared with the optimal policy of the MDP for a wide range of parameter settings, including product family asymmetries in arrival rate, order size, and order reward.

4.1 Introduction

In this chapter we consider a production-to-order environment in which a single bottleneck machine produces one product family at a time and is subject to significant setup times when the product family changes. The sequence in which families can be produced is arbitrary, rather than, for instance, cyclic. Orders for all families arrive with geometrically distributed inter-arrival times, require deterministic service time and have strict lead time. If an arriving order cannot be produced within its lead time, it is rejected upon arrival. The aim of the production system is to maximize the long run acceptance probability, or in case the rewards per order may differ among the families, the long run expected reward per arriving order.

This production situation has numerous applications. The Ph.D. thesis of Ten Kate (1995) contains an industrial motivation for this problem. Schmidt et al. (2001) are concerned with efficiently producing baseball bats. Vandaele et al. (2003) focus on forming effective batch sizes for an NMR scanner. Strijbosch et al. (2002) concentrate on the scheduling of packaging pharmaceuticals on order. Following Markowitz et al. (2000), Markovitz and Wein (2001) and the survey paper of Winands et al. (2011) we refer to this production situation as a *customized stochastic lot scheduling problem (CSLSP)* with strict lead times. Here, the term *customized* refers to make-to-order characteristic of the production situation, contrary to e.g. the Stochastic Economic Lot Sizing Problem (SLSP) in which production is make-to-stock hence allows more freedom in deciding the moment, type and quantity of item to produce.

Clearly, maximizing the long run expected reward per arriving order requires a policy that accepts orders and schedules the accepted orders. Designing good policies is, however, not entirely trivial since the acceptance and scheduling decisions are subtlety related: the acceptance depends on a schedule of previously accepted orders, and if the order is accepted, the order should be scheduled such that all already present orders are produced within their lead times. To illustrate, consider the policy that separates each two accepted orders by a setup and applies a FIFO service discipline. This policy is certainly simple, but also unattractive as it leads to a low acceptance rate. Intelligent policies should try to form 'runs' of orders of the same family to reduce the fraction of time spent on setups. In fact, it may sometimes be better to reject orders even when the schedule allows to accept it, the idea being that given the contents of the current schedule the rejection leaves room for later, more attractive, arrivals.

The fact that the CSLSP with strict lead times and rejections is a rather intricate system explains that previous work, such as Bertrand et al. (1990), Ten Kate (1995), Van Foreest et al. (2010), uses simulation to compare the performance of several *heuristic* scheduling and acceptance policies. While this approach is certainly powerful, simulation just allows to compare the performance of one heuristic to another, but provides no means to actually *benchmark* these heuristic policies against an optimal policy when this optimal policy is unknown.

The contribution of this chapter is fourfold. First, we present a Markov decision process (MDP) formulation for the CSLSP. An important element in the characterization of the MDP is the formulation of admissible actions that determine the acceptance/rejection of orders and the formation of job schedules: Upon arrival of a new customer order the planner has the option to accept or reject the arriving order. If the planner chooses to accept the order, the planner next has to decide how to form a new schedule with the accepted but not yet completed orders. Any action that results in a schedule in which orders are produced in time is in principle an admissible action for planner. Note that this set of admissible actions is large and contains many actions that are not interesting for the planner. Moreover, if we want to consider the complete set of admissible actions, the size of the problem becomes too large to solve by means of an MDP approach. As a first attempt to gain insight into good acceptance and scheduling policies using an MDP approach we therefore restrict the set of admissible scheduling actions to two actions that seem to be the most interesting for the planner: one action that combines a new order with a run of the same family in the schedule; and another action that generates a new run for an accepted order.

The second contribution of this chapter is that we use the theory of stochastic dynamic programming to find the optimal policy for this CSLSP. By modeling the system as an MDP, we obtain (numerically) the optimal policy by policy iteration, and analyze its structure and its performance. We show on the one hand that the structure of the optimal policy is difficult (if not impossible) to characterize in closed form in general. On the other hand, we provide support that in case of two families with equal order size and reward the optimal policy has a simple threshold structure. Third, we use the optimal policy to effectively benchmark a heuristic threshold policy developed in the earlier simulation studies. The motivation here is that this heuristic is easy to understand and implement, for instance in a spreadsheet, hence has large practical value. When, by means of benchmarking, it can be shown that this threshold policy performs well compared to the optimal policy, it might suffice in practice to just use this heuristic, rather than the more elaborate optimal policy. Fourth, we provide an improvement of the best heuristic policy of Van Foreest et al. (2010). This modification, while conceptually simple, allows the improved heuristic to cope with product type asymmetries, such as different job sizes, job rewards, arrival rates. We show that that in all these circumstances the modified heuristic performs excellently, and outperforms the previously known heuristic considerably.

This chapter is organized as follows. In Section 4.2 we discuss related work. Section 4.3 presents the model of the production system, the admissible policies, and the optimization problem. Section 4.4 describes the associated Markov decision process and formulates the heuristic policies. Section 4.5 shows the results, and Section 4.6 concludes.

4.2 Theoretical Background

Although variations of the CSLSP with strict lead times have been investigated previously, the analysis by means of MDPs has, to the best of our knowledge, not been addressed before in the literature.

Polling models, see for instance Takagi (1990) or Winands et al. (2011), form an interesting class of relaxations of our production situation, but are not entirely suitable, for at least three reasons. First, many polling systems are concerned with unlimited queues, while the restriction to meet lead times in our situation puts a natural limit on the length of the schedule. Polling models that do consider finite queue sizes, for instance Takagi (1991), Kim and Van Oyen (2000), relate a single queue to each product family. In our case, however, the queueing capacity can be fully shared among all families. Second, numerous polling models that include setups, e.g., Kleinrock and Levy (1988), Righter and Shantikumar (1998), Markovitz and Wein (2001), assume that the server incurs a setup even when it visits an empty queue. This is a reasonable assumption for polling models of telecommunication systems since the locations between the queues are physically separated and the server (token) has to visit a queue to see whether it is empty or not. However, setting up a machine for non-present families appears quite unnatural for the scheduling problems occurring in manufacturing situations; planners usually have a complete view of the order portfolio. Third, many polling models, see e.g., Reiman and Wein (1998), are concerned with cyclic policies. However, our earlier work shows that it is detrimental to cyclically serve the product families for the CSLSP with strict lead times; the heuristic threshold policy performs at least a few percent better. Strategies that use fixed polling tables, such as developed by Boxma et al. (1991), also seem hard to apply here. As the lead times of the orders in the production system need to be respected, the content of the schedule is highly dynamic. Moreover, the schedules we are concerned with may contain *multiple* runs of one product family. This last aspect has a further consequence in that policies such as, e.g., 'serve a queue of orders of the same family to exhaustion', see for instance Liu et al. (1992), are not suitable here. It may be necessary to switch service from one family to another before all orders of the family in service are produced.

Also the approach of Vandaele et al. (2003) is not suitable for our case. They consider fixed group sizes for orders with the same family and optimize over the sizes of the groups. However, in the resulting policy group completion times are not guaranteed, hence orders may not meet their lead times.

In this work we are concerned with the *online* CSLSP, i.e., the set of future orders is unknown at the moment of arrival. Related recent work addresses the *offline* CSLSP, i.e., the orders with strict lead times are given before hand. The offline CSLSP of accepting and scheduling n independent jobs with release dates, lead times, and family setup times on a single machine to maximize total reward is NP-hard in the strong sense, since it is a generalization of the problem 'Sequencing with release times and deadlines' (Garey and Johnson, 1979, p. 236). Oğuz et al. (2010), for example, develop three heuristic algorithms for this problem in which they also consider sequence dependent setup times. Gutiérrez-Jarpa et al. (2010) develop an exact branch-and-price algorithm for the related Vehicle Routing Problem with selective pickups and time windows.

4.3 Production System, Admissible Policies, and Objective Function

We start with describing the production system. Next, in Section 4.3.2, we introduce the decisions that govern the acceptance and scheduling of arriving orders. Section 4.3.3 discusses the objective function.

4.3.1 Production System

The production situation is modeled as follows. A single server receives a stream of orders at arrival epochs $0 = T_0 \leqslant T_1 \leqslant T_2, \ldots$. The inter-arrival times $T_{i+1} - T_i$

are i.i.d., integer valued, and geometrically distributed with success parameter p, that is, according to

$$P(T_{i+1} - T_i = k) = p(1-p)^k, \quad \text{for } k = 0, 1, 2, \ldots .$$

Observe that the inter-arrival time can be zero, i.e., $k = 0$, and thereby multiple orders may arrive at the same time. Since $E(T_{i+1} - T_i) = (1-p)/p$, the arrival rate $\lambda = p/(1-p)$. Arriving orders belong to family f, which is one of N possible families, with probability q_f, independent of anything else. Thus, the arrival rate of family f is $\lambda_f = \lambda q_f$. A job of family f—we use the word 'job' and 'order' interchangeably—requires b_f time units of service, where b_f is deterministic and integer valued. An order can be rejected upon arrival, but if accepted it is scheduled for service such that it can be produced within a lead time of length h, which is the same for all families. (We provide motivation for this choice below.) Whenever two subsequent orders in the schedule belong to different product families a setup of (integer valued) duration s, which is the same for all families, is inserted between these two orders. When an order arrives at an empty system, a setup is not necessary if the last produced order is of the same family as the arriving order. Service of orders and setups is non-preemptive, and the server is assumed to never fail, so that all accepted orders can be produced in time.

With regard to the scaling of the system, observe that without loss of generality it is possible to scale the lead time and the sizes of the jobs and setups such that at least for one family the jobs have unit time length or the setup time is of unit length.

The above model allows, formally, for family-dependent lead times. However, in industry, see, e.g., Ten Kate (1995), it is standard practice to use a uniform lead time, as it is not so clear how to exploit different lead times. Families with long lead times can claim more machine capacity and then families with shorter lead times, and in general, this form of unfairness appears to be quite undesirable and hard to resolve when different lead times are allowed.

4.3.2 Admissible Actions and Policies

We now describe the actions that determine the acceptance/rejection of orders and the formation of job schedules.

The admissible decisions, *Combine*, *Spawn*, and *Reject*, are deterministic (only dependent on the state of the schedule), and illustrated by means of the schedule instance in Table 4.1. Admissible policies are stationary and non-anticipative with respect to the arrival process and work-conserving (non-idling if the schedule contains orders).

Table 4.1: *An instance of a schedule of accepted orders. The symbols 'R', 'G', and 'B', refer to order colors, 's' to a setup. The size of the schedule is 14 positions. Jobs and setups have unit length here.*

1	2	3	4	5	6	7	8	9	10	11	12	13	14
R	R	s	B	B	B	s	G	s	R				

The *Combine* action aims at reducing the fraction of setups by trying to combine a new order with a *run* of the same family, i.e., a set of consecutive orders of the same 'kin'. By simple pairwise interchange arguments, see e.g., Pinedo (2008), it is easy to see that the optimal sequence of jobs within a run is Earliest Due-Date first—the reward cannot increase by inserting arriving orders before orders of the same family—hence, the Combine action adjoins accepted orders only to the *end* of a run. To illustrate, suppose a 'blue' order arrives. The schedule in Table 4.1 contains one blue run, that is, at positions 4, 5, and 6. The Combine action tries to join the order with this run by inserting it at position 7 and shifting the already present orders at positions 8 and 10 back to positions 9 to 11. In case either one of the orders at positions 8 and 10 will be late as a result of the insertion, Combine is not allowed to insert this new order at position 7.

The *Spawn* decision adjoins a new order to the end of the schedule so that it 'spawns' a new generation of its family. This acceptance will necessarily introduce a setup between the last family and the new order. Hence, Spawn schedules the just arrived job at position 12 and inserts a setup at position 11. Thus, to accept such an order the schedule should at least provide room for the setup and the order. If the contents of the schedule is such that both Combine and Spawn are admissible, the former action is always chosen. From a practical point of view this is entirely reasonable: why delay an order more than necessary?

The *Reject* decision simply rejects the arriving order.

For later purposes we define the *slack* of an accepted job as the amount of time that is available to insert other, later arriving, orders in front of the accepted job. For instance, suppose that the 'red' order in position 10 has a lead time of 14. Then the slack of this red job is 4 since 4 time units are available to insert other jobs in front of the job before it will be late.

4.3.3 Objective Function

The last step of the model specification consists of formulating a reward structure. We set the acceptance reward for a job of family f equal to $r_f > 0$ and the earliness cost to zero. The underlying motivation is that we assume that serving orders early is acceptable when this potentially leads to accepting more orders. (In the case of patients this seems to be even an advantage.) Hence, the reward for accepting an order must be higher than the cost of producing early. We also set the setup cost equal to zero. However, inserting setups arbitrarily cannot be optimal, since a setup takes away a position in the schedule thereby preventing potential rewards. There is no tardiness cost, as jobs cannot be late.

The objective is to find the maximal long run average reward of the production system. More formally, given policy π, let $A^\pi(k)$ be the sum of the rewards of accepted orders among the first k arrivals. Then the *long run expected reward per arriving order*, J^π, takes the form

$$J^\pi = \lim_{k\to\infty} \frac{\mathrm{E}\left(A^\pi(k)\right)}{k}. \tag{4.1}$$

The objective is to find

$$J^\star = \max_\pi J^\pi, \tag{4.2}$$

where the maximization is taken over the class of admissible policies. As an aside, (4.1) is often formulated in terms of an limit inferior. However, since the state space is finite, this subtlety is unnecessary as the limit exists by Proposition 8.1.1 of Puterman (1994).

A convenient related performance indicator is the *rate of accepted reward* λJ^π. In case the reward per job is equal to the job length, the reward function J^π represents the long run average fraction of accepted work, and λJ^π corresponds to the *utilization*.

Finally, to study the fairness of a policy it is of interest to define performance indicators per product family. For a given policy π, let $A^\pi_f(k_f)$ be the sum of the rewards of accepted orders of family f among the first k_f arrivals of family f. Then under policy π, the *long run expected reward per arriving order of family* f, J^π_f, is given by

$$J^\pi_f = \lim_{k_f\to\infty} \frac{\mathrm{E}\left(A^\pi_f(k_f)\right)}{k_f}. \tag{4.3}$$

and the *long run expected fraction of accepted orders of family* f by $W^\pi_f = J^\pi_f / r_f$.

4.4 The Markov Decision Process

We now turn to modeling the above production situation as a Markov decision process (MDP). An MDP consists of a set of states, a set of decisions, state transition functions $P(x'|x, a)$ representing the transition probability from state x to a future state x' conditional on choosing decision a, and a function $R(x, a)$ that provides the reward earned when taking decision a in state x. For general background on the definition and analysis of Markov decision processes we refer to Puterman (1994) or Tijms (2003).

As the specification of the system state x is somewhat involved, we characterize the format of a state in Section 4.4.1. The actions have already been introduced in Section 4.3.2; we only need to formalize the actions, which we do in Section 4.4.2. Section 4.4.3 presents the state transition functions. The rewards $R(x, a)$ are already clear from Section 4.3.3: only decisions that lead to the acceptance of an order of family f generate positive reward r_f. Section 4.4.4 explains a convenient method to generate the state space of the Markov chain. In Section 4.4.5 we formulate the heuristic threshold policy in terms of the notation developed and in Section 4.4.6 we evaluate the optimal and threshold policy. Finally, in Section 4.4.7 we discuss a suitable method to aggregate the MDP for instances for which job size, arrival rate and reward are identical for all families. This method allows us to considerably extend the system sizes suitable to study.

4.4.1 Format of a State

Let us characterize a system state such that it contains sufficient information for the actions of the MDP. First, recall that the decision epochs coincide with the arrival epochs, hence we only have to specify the state of the system at these epochs. Next, it is actually not necessary to store information about all individual orders in the schedule. In fact, the Reject action needs no state information at all. The Spawn action requires only the length of the schedule and the size and family of the new job. The Combine action needs only the slack of the 'tightest' orders of the runs, i.e., the orders with the least slack per run, since if the tightest order of a run is in time, all orders in the run will be in time. Thus, a state consists of the sequence of runs, and, per run, the family, length (including a setup), and slack of the 'tightest' job.

To cast the above in suitable notation, write $x = (\sigma; f)$ where σ denotes the contents of the schedule and f indicates the family of the arriving order. The schedule σ is an ordered tuple of runs $(\sigma_1, \ldots, \sigma_n)$. A run σ_i, $i \geqslant 2$, is also an ordered tuple (f_i, δ_i, l_i), where f_i is the run's family, δ_i the slack of the tightest order, and l_i the length (including setup time). The first run σ_1 needs no slack information, since maintaining a run's slack is only relevant to ensure that earlier runs cannot become too long; however there are no runs in front of σ_1, and thus, $\sigma_1 = (f_1, l_1)$. When the schedule is empty, we write $\sigma = (f_1, 0)$, where f_1 is the last produced family.

A simple observation allows us to reduce the information in x still further. Since arriving orders cannot be combined with runs in front of a tight run, nor can new runs be spawned before this tight run, the family and slack of these runs are superfluous, hence the lengths of these runs can be added simply to the length of the tight run, and the tight run must be the first run in the family. More formally, suppose that the kth run in the schedule is tight. Then the schedule $\sigma = (\sigma_1, \ldots, \sigma_n)$ can be aggregated into the schedule $\sigma' = (\sigma'_1, \sigma'_2, \ldots,)$, where $\sigma'_1 = (f_k, \sum_{i=1}^k l_i)$, $\sigma'_2 = \sigma_{k+1}$, ...

The last step is to show how to project back the slack of a newly accepted job to the slack of the tightest job of a run. Suppose that an arriving order of family f with lead time h is to be combined with run σ_k. Then $\sum_{i=1}^k l_i$ is the amount of work in front of the new job, and the slack of the new job is therefore $h - \sum_{i=1}^k l_i - b_f$. Since δ_k was the slack of the tightest job before the acceptance,

$$\delta'_k = \min\{\delta_k, h - \sum_{i=1}^{k} l_i - b_f\}, \tag{4.4}$$

must be new slack of the tightest job in run k. In case the job is spawned, the slack of the tightest job in the new run is simply

$$\delta_{n+1} = h - |\sigma| - b_f - s. \tag{4.5}$$

4.4.2 Actions and Operators

Making a transition from one state to the next involves three steps. The first step is to apply one of the actions Reject, Spawn, and Combine to the arriving order. The second step consists of generating the time to the next arrival epoch, and removing the related amount of workload from the schedule. In third step, a new arriving order is generated. We implement each step by means of operators, to be defined now. For notational convenience, let $|\sigma| = \sum_{i=1}^n l_i$ denote the total amount of work in the schedule including setups.

First we associate the actions Reject, Spawn and Combine to operators R, S, C which map *states* $x = (\sigma; f)$ to *schedules* $\sigma = (\sigma_1, \dots, \sigma_n)$. The operator R simply removes the arriving order:

$$R\,(\sigma; f) = \sigma;$$

S adjoins the order to the end of the schedule,

$$S\,(\sigma; f) = (\sigma_1, \dots, \sigma_n, (f, \delta_{n+1}, s + b_f))\,,$$

where δ_{n+1} is defined in (4.5), s denotes the length of a setup, and b_f the size of the new job; and the operator C combines the arrival with the last run of family f in the schedule:

$$C\,(\sigma; f) = (\sigma_1, \dots, \sigma_{k-1}, (f, \delta'_k, l_k + b_f), \sigma_{k+1}, \dots, \sigma_n), \tag{4.6}$$

where δ'_k is defined in (4.4), and we assume without loss of generality that σ_k is the last run of family f in the schedule.

For each state x, the set $A(x)$ contains the actions that can be applied to x. Specifically, since the Reject action is always possible, $R \in A(x)$ for all x. With regard to the Combine action, $C \in A(x)$ if the insertion of the new order of family f by Combine does not lead to any other job in the schedule becoming late. In more formal terms, suppose σ_k is the last run of family f in σ, as in (4.6). Then the insertion is allowed if $\delta_i \geqslant b_f$ for all $i \geqslant k+1$. Finally, $S \in A(x)$ when $C \notin A(x)$ and $|\sigma| + b_f + s \leqslant h$, i.e., the length of the schedule that results after the operation of S on x is less than or equal to h.

We next need the *time shift* operator T, which maps a *schedule* to a *schedule*, to implement the effect on the schedule when the time to the next arrival epoch is one time unit. Clearly, if the schedule is not empty and the time to the next arrival epoch is one time unit, (part of) the first run of the schedule is produced, and the slack of the remaining runs is reduced by one time unit. Specifically, while temporarily extending the definition of T to operate on single runs so that we can also write $T\sigma = (T\sigma_1, \dots, T\sigma_n)$, we have

$$\begin{aligned}
T\sigma &= ((f_1, 0)), \quad \text{if } l_1 = 0 \text{ and } n = 1,\\
T\sigma &= ((f_1, l_1 - 1), T\sigma_2, \dots, T\sigma_n), \quad \text{if } l_1 \geqslant 1,\\
T\sigma &= ((f_2, l_2 - 1), T\sigma_3, \dots, T\sigma_n), \quad \text{if } l_1 = 0 \text{ and } n > 1,\\
T\sigma_i &= (f_i, h_i - 1, l_i), \quad \text{if } i \geqslant 2.
\end{aligned} \tag{4.7}$$

Shifting by $m \geqslant 0$ time units is simple, just apply the m times composition T^m. No shift in time is represented by T^0, which is the identity operator.

The last operator F_f maps a *schedule* to a *state* by augmenting σ with an arriving order of family f:

$$F_f(\sigma) = (\sigma; f)\,, \quad f = 1, \dots, N.$$

4.4.3 Transition Matrices

With the above operators we can map a state x to a future state, and hence describe the transition matrices associated to each of the actions Reject, Spawn, and Combine.

If the next order arrives m time units from now and is of family f, then for any $a \in A(x)$ we write $x'_{fm} = (F_f \circ T^m \circ a)(x)$ for the next state. The set $\mathscr{F}(x)$ of future states of x is given by

$$\mathscr{F}(x) = \left\{ x'_{fm} \,\middle|\, x'_{fm} = (F_f \circ T^m \circ a)(x),\ \text{for } a \in A(x), 0 \leqslant m \leqslant |a(x)|, 1 \leqslant f \leqslant N \right\}, \tag{4.8}$$

recall that $|a(x)|$ denotes the length of a schedule after the operation of a on x. The transition matrices now follow easily: the probability to jump from state x to a state $x'_{fm} \in \mathscr{F}(x)$ is

$$P\left(x'_{fm} \,\middle|\, x, a \right) = q_f\, p(1-p)^m, \quad \text{if } m < |a(x)|, \tag{4.9}$$

and

$$P\left(((f_1, 0); f) \,\middle|\, x, a \right) = q_f \sum_{m \geqslant |a(x)|} p(1-p)^m = q_f\,(1-p)^{|a(x)|}, \quad \text{if } m \geqslant |a(x)|, \tag{4.10}$$

where again f_1 is the last produced family.

4.4.4 State Space

The formal characterization of the state space $\mathscr{S}$ requires to enumerate all possible schedules. However, this is a cumbersome task due to the interaction between orders and setups. By far the easiest way to generate $\mathscr{S}$ is by induction. Using the set of future states of x as defined in (4.8), let $S^{(i+1)} = \bigcup_{x \in S^{(i)}} \mathscr{F}(x)$, and take as initial set $S^{(0)} = \{((f,0);g) | f, g = 1, \dots, N\}$. Once there is an i such that $S^{(i+1)} = S^{(i)}$, it must be that $\mathscr{S} = S^{(i)}$. Observe that this iterative procedure stops trivially, due to the finiteness of schedule length and number of different families.

Although there is no simple way to generate the state space which allows us to compute the size of the state space, this size must increase very rapidly as a function of N and h. To obtain a rough indication, suppose that each of the N families has a run in the schedule. The runs are separated by a setup, and each run contains at least one job. Thus, roughly speaking, each run consumes at least two positions of the schedule, one setup and one job. As a consequence, the number of free positions left in the schedule must be $h - 2N$. Clearly, each of these free positions can be occupied by an order, yielding N^{h-2N} possible schedules. In fact, there must be more feasible schedules, since runs can have different due-date slacks, there may also be less than N families present in the schedule, and not all positions in the schedule need be filled.

4.4.5 The Heuristic Threshold Policy

With the machinery developed above it is easy to model the system under the threshold policy as an MDP.

For a given state x, the threshold policy always applies the Combine action if $C \in A(x)$. If $C \notin A(x)$, the Spawn action is chosen, provided $|S(x)| \leqslant ch$, where c is some constant smaller than 1, typically around 0.8. If this also fails, the order will be rejected. Clearly, as this policy is deterministic, and makes decisions based only on the state of the schedule and the newly arriving order, it is stationary, hence admissible.

The intuition behind this policy is as follows. Any time spent on setups does not increase the reward function as defined in (4.2), and potentially decreases it as setup time cannot be used to process orders. Hence, policies that encourage the formation of long runs, without violating the lead time constraints, appear the most interesting. Furthermore, in case the load of one family does not suffice to fill the machine capacity, it is obvious that the capacity should be shared among a few families, hence, runs of orders of families should alternate. However, as soon as multiple families share the capacity, they are in effect ‘competing’ for the capacity. As is generally the case, and shown in Van Foreest et al. (2010), a good policy should regulate this competition. In fact, we show there that if the load is high and the Spawn action is not regulated, typical runs contain just one or two jobs. A good policy should regulate the Spawn action to create *combination* *potential*: runs may only start when the schedule is *not* full so that the first job of a run is *not* tight when the run starts. Since a threshold to enable or disable the Spawn action is about the simplest policy possible, we use this threshold in the heuristic policy.

If the threshold value c is uniform for the families, we expect the threshold policy to work well in symmetric cases, that is, in production situations in which the contending families have roughly similar arrival rates, rewards, and job sizes. However, when there is asymmetry, it may become better to make the threshold values family dependent.

4.4.6 MDP Computation

The objective of the MDP is to maximize the long run expected reward per arriving order $J^\star$ (see Section 4.3.3). To compute the optimal policy we use policy iteration, which comes down to solving for a function $v^\star$, a policy $\pi^\star$, and a constant $g^\star$ such that

$$v^\star(x) = \min_{a\in A(x)} \left\{ R(x,a) - g^\star + \sum_{y\in\mathcal{S}} P(y|x,a)v^*(y) \right\}$$

$$\pi^\star(x) = \arg\min_{a\in A(x)} \left\{ R(x,a) + \sum_{y\in\mathcal{S}} P(y|x,a)v^*(y) \right\}, \quad x\in\mathcal{S}.$$

Then $\pi^\star$ is an optimal control and $g^\star$ is equal to the solution $J^\star$ of (4.2). For further details see (Tijms, 2003, Chapter 6).

To compute the performance of the threshold policy we solve for a function v, and a constant g such that

$$v(x) = R(x,t(x)) - g + \sum_{y\in\mathcal{S}} P(y|x,t(x))v(y),$$

where $t(x)$ implements the action taken by the threshold policy in state x. Then g is the long run average reward under the threshold policy.

4.4.7 Further Aggregation in the Symmetric Case

In the case the families are symmetric, i.e., the arrival rates, job sizes and rewards are equal for all families, it turns out that it is not necessary to store information concerning the family of a run in the schedule. Because of the symmetry in arrival rate, an arriving order is of family f with probability $1/N$. Recall that a new run can only be spawned if it is not possible to combine the arriving order with a run of the same family in the schedule. Since we keep only track of runs in the schedule to which new orders can be combined (i.e. runs preceding a tight runs are aggregated, see Section 4.4.1) it follows that the number of runs in the schedule cannot exceed the number of families and that each family has at most one run in the schedule. Therefore, the probability that an arriving order sees upon arrival a run in the schedule to which it can be combined is n/N, where n denotes the number of runs in the schedule.

The state of the system x can now be aggregated into a tuple $(\sigma; m)$ where $\sigma = (l_1; h_2, l_2; \ldots)$ denotes the content of the schedule and m is an indicator that denotes the run of the schedule to which the arriving order can be combined (in case $m \geqslant 1$) or the case that no run in the schedule is from the same family as the arriving order (in case $m = 0$). For example, $x = ((2; 5, 3; 8; 2); 1)$ depicts the state in which an arriving order can be combined with the first run in the schedule.

It is apparent that the operators C, R, and S of Section 4.4.2 and the state transition probabilities of Section 4.4.3 have to be converted to the aggregated process. As this is relatively easy we refrain from including the details.

4.5 Numerical Study

In this section we are concerned with the performance of the threshold policy implemented as an MDP and compare this to the optimal performance obtained by policy iteration.[1] We investigate the effect of various system parameters. First, in Section 4.5.1, we consider symmetric cases, that is, situations in which the families have identical characteristics. From the results, we will learn that the threshold policy has (near) optimal performance, which makes it interesting to investigate whether the threshold policy resembles the optimal policy. This is done in Section 4.5.2 by means of a statistical analysis of the structure of the optimal policy. In Section 4.5.3 we study the effect of asymmetries in job size, arrival rate per family, and reward and provide in Section 4.5.4 an analysis of the structure of the optimal policy for asymmetric instances.

For notational convenience we use vectors b, q and r, to denote the job size,

[1] We implemented all algorithms in python and numpy, which are freely available on the web. At request the code used for the computation can be obtained from the author.

arrival fraction and reward per family. For instance, $b = (b_1, b_2) = (1, 3)$ specifies the scenario in which the job size of the first (second) family is 1 (3). Throughout we restrict the total load of the system $\rho = \lambda \sum_f q_f b_f$ to the interval $[0.5, 1.2]$. Higher or lower values appear less relevant to study from a practical point of view. The parameters N and h determine the size of the state space and are therefore bounded by computer memory.

The reasoning in Section 4.4.4 shows that cases with, e.g., $N = 8$ and $h = 30$ such as considered in Van Foreest et al. (2010) are completely out of reach for an analysis by means of MDPs. We have to restrict the numerical analysis here to cases with $N \in \{2, 3, 4, 5\}$, and lead time that decrease as a function of N from $h = 30$ to $h = 10$.

4.5.1 Influence of c, h, ρ, s and N

In our first experiment we study the effect of the threshold parameter c, the system load ρ, and the lead time h, for $N = 3$ symmetric families.

Figure 4.1 shows the influence of the threshold parameter c on the long run accepted reward J per arriving order as obtained by the threshold and optimal policy. The graphs for $\rho = 0.5, 0.9$, and 1.2 present the dependence on ρ. Taking $h = 10, 14$, and 18, in the upper left, upper right, and lower panel shows the influence of h.

The figure allows us to make a number of interesting observations. First of all, the expected reward J per arriving order decreases as a function of ρ. This is natural: when for instance $\rho = 1.2$ at least 20% of the offered load has to be rejected. Second, since the graphs of the expected rewards of the threshold and optimal policy touch for some value of c, the threshold policy can have (near) optimal performance. Third, when c is large, in the order of 1, the threshold policy performs quite badly. This is due to the fact that it gives rise to schedules in which short runs of jobs alternate with setups, see Van Foreest et al. (2010) for further detail. On the other hand, when $c \approx 0$, the performance also deteriorates because runs only start when the schedule is nearly empty, hence the server idles quite often. Fourth, the best value of c becomes smaller as ρ becomes larger. This is as expected: the higher the load, the more combination potential by using the Combine action, hence longer runs. Comparing the panels in a clockwise direction, to understand the behavior of h, we see that the average reward J increases. This is not surprising since longer lead times also enable more combination potential.

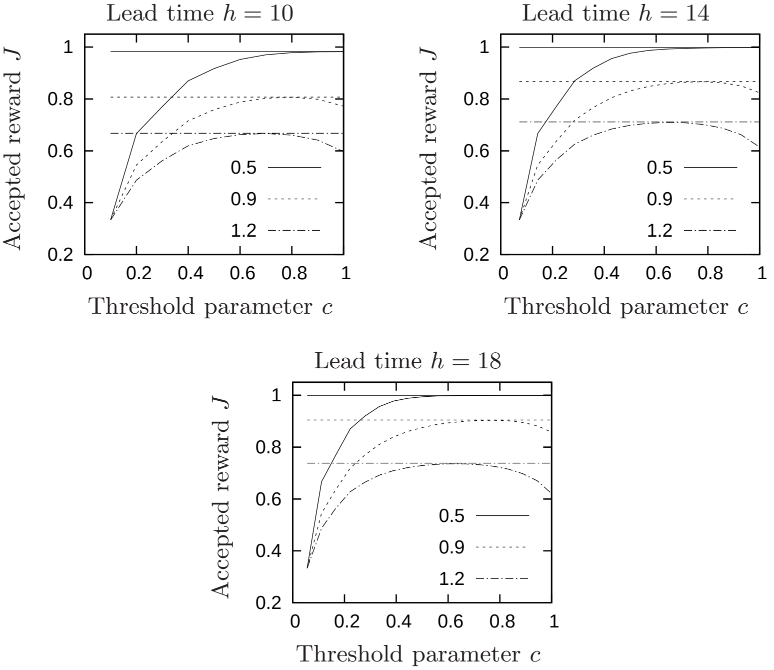

Figure 4.1: The long run accepted reward J per arriving order as a function of the threshold parameter c for the threshold and optimal policy. The straight lines correspond to the performance of the optimal policy which, of course, cannot not depend on c. The highest, middle and lowest graphs correspond to $\rho = 0.5, 0.9$, and 1.2, respectively. The upper left, upper right, and lower panel correspond to $h = 10, 14$, and 18. The other parameters are as follows: $N = 3$, $s = 1$, $b = r = (1, 1, 1)$, and $q = (1/3, 1/3, 1/3)$.

Similar figures for $N = 2, 4$, and 5 and for different setup times s (not included here) show also that the threshold policy has (near) optimal performance when c is set appropriately.

4.5.2 Structure of the Optimal Policy for Symmetric Instances

From the results of the previous sections it appears that a suitable threshold policy might be optimal which in turn would imply that the optimal Spawn–Reject decision has a threshold structure. To investigate this observation more formally we follow an approach, proposed in Haijema et al. (2007), to visualize the structure of the optimal policy by grouping states x into sets

$$\begin{aligned} \mathcal{S}^s_m(a) &= \{x \,|\, S \in A(x), |S(x)| = m, \pi^\star(x) = a\}, \quad \text{if } a \in \{S, R\}, \\ \mathcal{S}^c(a) &= \{x \,|\, C \in A(x), \pi^\star(x) = a\}, \quad \text{if } a \in \{C, R\}, \end{aligned} \tag{4.11}$$

where $\pi^\star(x)$ is the optimal action in state x. For example, $\mathcal{S}^s_6(S)$ is the set of states with $|S(x)| = 6$ and for which it is optimal to spawn a new run rather than reject the arriving order. Let $\eta^\star(\tilde{\mathcal{S}}) = \sum_{x\in\tilde{\mathcal{S}}} \eta^\star(x)$, where $\eta^\star(x)$ is the steady state fraction of time the $\pi^\star$-controlled MDP spends in state x.

Table 4.2 shows the frequencies $\eta^\star(\mathcal{S}^s_m(S))$ (in percentages) and $\eta^\star(\mathcal{S}^s_m(R))$ for $m = 2, \ldots, 10$, and $N = 2$, $b = r = (1, 1)$, $s = 1$, $h = 10$, $\rho = 1.2$ and $q = (1/2, 1/2)$. We see that the optimal Spawn–Reject decision is based only on $|S(x)|$, and that there is a threshold at $|S(x)| = 7$, just as for the best threshold policy. Pertaining to the Combine–Reject decision the table shows that the optimal policy always chooses Combine whenever $C \in A(x)$. Combining these observations we conclude that for this set of parameters the threshold policy with threshold parameter $c = 7/h = 0.7$ is equal to the optimal policy.

Table 4.2: Frequency table of $\eta^\star(\mathcal{S}^s_m(S)) \times 100\%$ and $\eta^\star(\mathcal{S}^s_m(R)) \times 100\%$, $m = 2, \ldots, 10$, for the Spawn–Reject and Combine–Reject decision. The parameters of the case are as follows: $N = 2$, $h = 10$, $\rho = 1.2$, $s = 1$, $b = r = (1, 1)$, and $q = (1/2, 1/2)$.

$\lvert S(x)\rvert$	2	3	4	5	6	7	8	9	10	Total
Spawn	0.80	0.97	1.64	1.66	1.95	2.75	0.00	0.00	0.00	9.77
Reject	0.00	0.00	0.00	0.00	0.00	0.00	4.48	5.19	5.25	14.93

	Total
Combine	62.46
Reject	0.00

To investigate whether a threshold policy is optimal in more general circumstances we next consider for $N = 2$ families a full factorial design of instances with lead time ($h \in \{10, 12, \ldots, 30\}$), setup time ($s \in \{1, 2\}$), job size ($b_f \in \{1, 2\}$), reward ($r_f \in \{1, 2\}$) and load ($\rho \in \{0.5, 0.6, \ldots, 1.2\}$). The results (not included here) show the threshold policy with suitably chosen threshold c to be optimal always.

We next study a case with $N = 3$ families. Now, as is demonstrated by Table 4.3, the optimal Spawn–Reject decision no longer has a clear threshold. Although this is somewhat disappointing (the simple threshold policy can no longer be optimal), not all structure is lost: in 97.55% (= 100 − 2.45) of the states the threshold policy with $c = 0.7$ still takes the same decisions as the optimal policy.

Table 4.3: Frequency table for Spawn–Reject and Combine–Reject decision. The parameters of the test instance are as follows: $N = 3$, $h = 10$, $\rho = 1.2$, $s = 1$, $b = r = (1, 1, 1)$, and $q = (1/3, 1/3, 1/3)$.

$\lvert S(x)\rvert$	2	3	4	5	6	7	8	9	10	Total
Spawn	1.04	1.25	2.33	2.86	4.21	3.60	0.00	0.00	0.00	15.29
Reject	0.00	0.00	0.00	0.00	0.00	2.45	8.30	8.00	6.10	24.85

	Total
Combine	51.47
Reject	0.00

To further investigate the quality of the threshold policy we compute the difference in performance between the optimal policy and the best threshold policy for the symmetric instances with unit job size and reward, setup time $s \in \{1, 2\}$, load $\rho \in \{0.5, 0.6, \ldots, 1.2\}$, lead time $h \in \{10, 12, \ldots, 18\}$ for $N = 3$, $h \in \{10, 11, \ldots, 15\}$ for $N = 4$, and $h \in \{10, 11, 12, 13\}$ for $N = 5$. The histogram in Figure 4.2 shows the results. Clearly, the best threshold policy is never more than 1% off, and in the majority of the cases the performance is within 0.2% of the optimal policy.

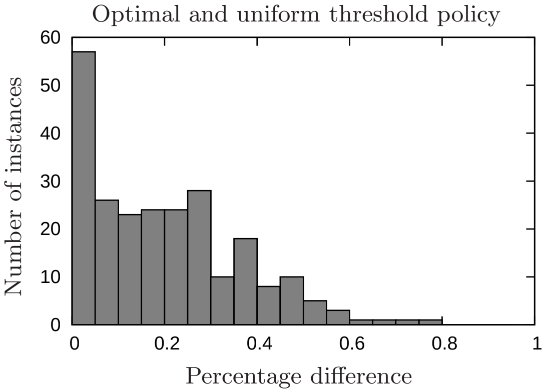

Figure 4.2: Histogram of percentage of performance difference between the optimal policy and the best threshold policy for symmetric instances with $N > 2$ product families.

We infer from a more detailed analysis of states that the optimal policy will, in general, not have a simple structure. For instance, for the case in Table 4.3, the optimal decision in state $((f_1 = 3, l_1 = 2), (f_2 = 1, \delta_2 = 2, l_1 = 3); f = 2)$ is Spawn, while in $((f_1 = 3, l_1 = 3), (f_2 = 1, \delta_2 = 2, l_2 = 2); f = 2)$ it is Reject. Still, both states have $|S(x)| = 7$, an order of family 2 arrives, and the order of families and the due-date slacks of the runs are the same. The only difference is in the length of the first and second run.

4.5.3 Asymmetry in Job Size, Arrival Rate and Reward

Next we study the influence of asymmetries in job size, arrival rate and reward on the performance of the threshold policy. Figure 4.3 shows for $N = 3$, $h = 12$, $s = 1$ the reward rate and the fraction of accepted orders of family 1 and 2 as obtained by the optimal policy (i.e., $\lambda J^\star$, $W_1^\star$ and $W_2^\star$) and best threshold policy (i.e., λJ^t, W_1^t and W_2^t) for three scenarios: *asymmetry in job size*, with $b = r = (3, 1, 1)$ and $q = (1/3, 1/3, 1/3)$; *asymmetry in arrival rate*, with $q = (0.1, 0.8, 0.1)$ and $b = r = (1, 1, 1)$; and *asymmetry in reward* with $r = (1, 2, 2)$, $b = (1, 1, 1)$ and $q = (1/3, 1/3, 1/3)$. The figure shows that, except for the scenario $q = (0.1, 0.8, 0.1)$, the threshold policy performs remarkably well. The case with asymmetry in arrival rate is, however, exceptional: just one family has a high arrival rate and, consequently, does not have to compete with the other families. As a result, the threshold policy does not work well, and there is also no need to decrease c.

Of particular interest is also the observation that the optimal policy becomes less fair with respect to fraction of accepted order of family 1 and 2 when the load increases. For instance, in scenario $r = (1, 2, 2)$ the two families with a high reward per accepted order receive under the optimal policy a much higher acceptance fraction than the family with a low reward per accepted order. This difference is much less pronounced under the threshold policy. On the other hand, the performance of the best threshold policy is just a few percent lower than the optimal policy. Apparently, sharing the resource in a rather fair way, as the threshold policy achieves, does not necessarily have to come at the expense of a large penalty on the total reward.

Next we investigate if the loss in performance due to family asymmetries can be repaired by making the threshold parameter family dependent. Let c_f denote the threshold parameter for family f. For a given state $x = (\sigma; f)$, the family-dependent threshold policy always applies the Combine action if $C \in A(x)$, while

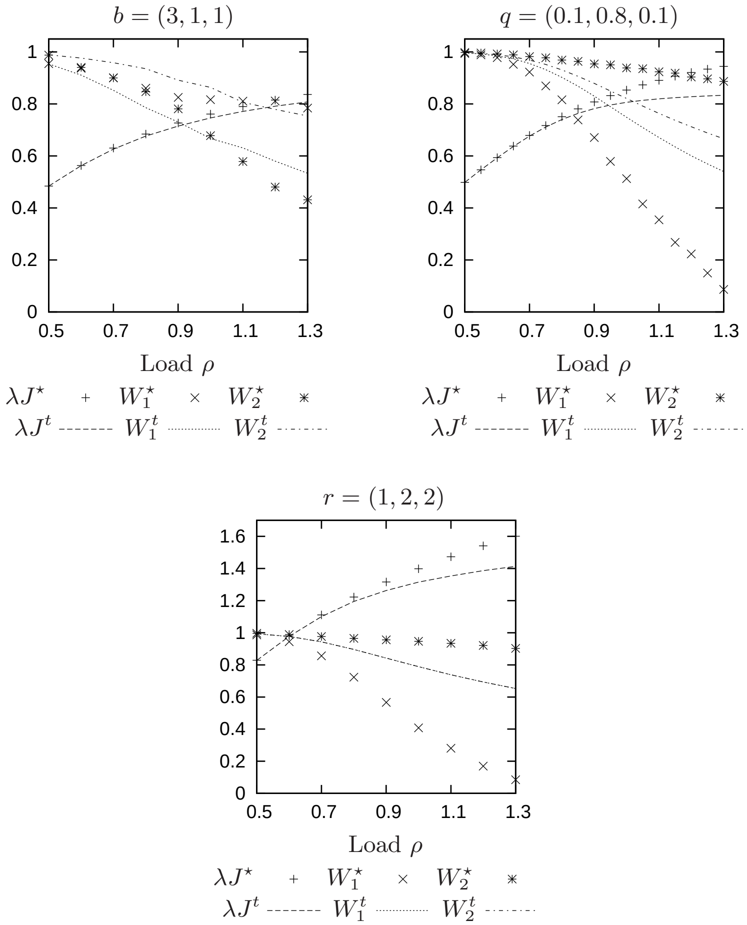

Figure 4.3: The reward rate and the fraction of accepted orders of family 1 and 2 as obtained by the optimal policy (i.e., $\lambda J^\star$, $W_1^\star$ and $W_2^\star$) and best threshold policy (i.e., λJ^t, W_1^t and W_2^t) as a function of the load ρ. (Due to symmetry, the fraction of accepted orders for families 2 and 3 are equal; hence the results for family 3 are not shown.) The upper left panel shows the results for $b = r = (3, 1, 1)$, the upper right panel for $q = (0.1, 0.8, 0.1)$ and the lower panel for $r = (1, 2, 2)$.

if $C \notin A(x)$ it chooses the Spawn action provided $|S(x)| \leqslant c_f\, h$. Figure 4.4 shows the performance of the best family-dependent threshold policy for the same three asymmetric cases considered in Figure 4.3. It is apparent that family-dependent threshold parameters can be found such that the performance difference between the family-dependent threshold and the optimal policy is negligible.

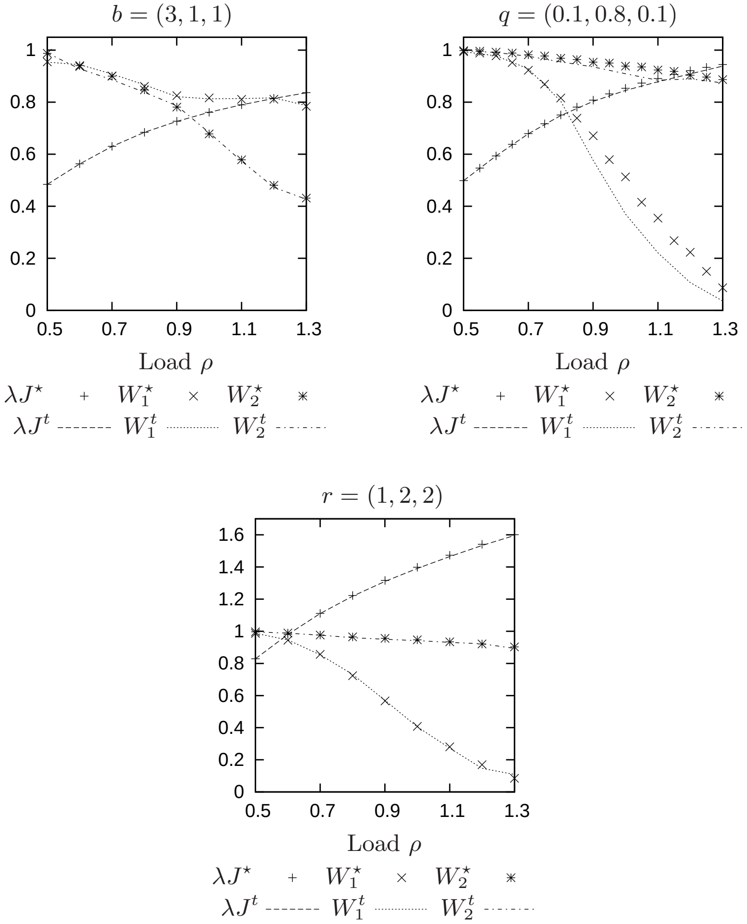

Figure 4.4: The cases are identical to those in Figure 4.3 except that now the threshold $c = (c_1, c_2, c_3)$ is family-dependent instead of uniform for all families.

4.5.4 Structure of the Optimal Policy for Asymmetric Instances

Now that we have seen that the (family-dependent) threshold policy can also give (near to) optimal performance for asymmetric instances, provided the threshold parameters are chosen suitably, it becomes of interest to investigate whether the optimal Spawn–Reject decision has a threshold structure for asymmetric instances. To do so, we redefine the sets $\mathcal{S}^s_m(a)$ in (4.11) as follows

$$\mathcal{S}^s_{m,g}(a) = \{x \mid S \in A(x), f = g, |S(x)| = m, \pi^\star(x) = a\}, \quad \text{if } a \in \{S, R\}. \tag{4.12}$$

For example, $\mathcal{S}^s_{6,1}(S)$ is the set of states with $|S(x)| = 6$, an arriving job of family 1 and for which it is optimal to spawn a new run rather than reject the arriving order.

Table 4.4 shows $\eta^\star\left(\mathcal{S}^s_{m,g}(S)\right)$ and $\eta^\star\left(\mathcal{S}^s_{m,g}(R)\right)$ for $m = 2, \ldots, 10$ and $g = 1, 2$, while $N = 2$, $b = (1, 1)$, $r = (2, 1)$, $s = 1$, $h = 10$, $\rho = 1.2$ and $q = (1/3, 2/3)$. Clearly, the optimal Spawn–Reject decision is based only on $|S(x)|$ and the family of the arriving order, and there is a threshold at $|S(x)| = 10$ for arriving orders of family 1 and at $|S(x)| = 6$ for family 2 orders, just as for the best family-dependent threshold policy. However, the results for the Combine–Reject decision show that for this set of parameters it is not optimal to always combine when $C \in A(x)$.

Table 4.4: Frequency table for Spawn–Reject and Combine–Reject decision. The parameters of the case are as follows: $N = 2$, $h = 10$, $\rho = 1.2$, $s = 1$, $b = (1, 1)$, $r = (2, 1)$, and $q = (1/3, 2/3)$.

	$\lvert S(x)\rvert$	2	3	4	5	6	7	8	9	10	Total
Fam. 1	Spawn	0.48	0.58	0.89	0.87	0.83	0.77	0.69	0.64	0.22	5.97
	Reject	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
Fam. 2	Spawn	0.56	0.68	1.27	1.39	2.08	0.00	0.00	0.00	0.00	5.97
	Reject	0.00	0.00	0.00	0.00	0.00	3.71	4.08	3.87	3.21	14.87

	Total
Combine	57.53
Reject	10.99

To investigate this observation in more general circumstances we consider a full factorial design of asymmetric instances that differ with respect to the number of families ($N \in \{2, 3, 4\}$), setup time ($s \in \{1, 2\}$), arrival rate, job size ($b_f \in \{1, 2\}$), reward ($r_f \in \{1, 2\}$), load ($\rho \in \{0.7, 0.9, 1.1\}$) and lead time ($h \in \{10, 15, 20, 25\}$ for $N = 2$; $h \in \{9, 12, 15\}$ for $N = 3$; and $h = 10$ for $N = 4$). The results show more generally that when job sizes or rewards are asymmetric it is not optimal to always combine if $C \in A(x)$. The optimal Spawn–Reject decision has still a threshold structure, but only when $N = 2$. Thus, we conclude that an optimal (family-dependent) threshold policy can only be found when $N = 2$ and job sizes and rewards are symmetric.

As for the symmetric instances it is interesting to investigate the quality of the threshold policy for asymmetric instances. Figure 4.5 shows two histograms displaying the performance difference between the optimal policy and the best threshold policy, and between the optimal policy and the best family-dependent threshold policy.

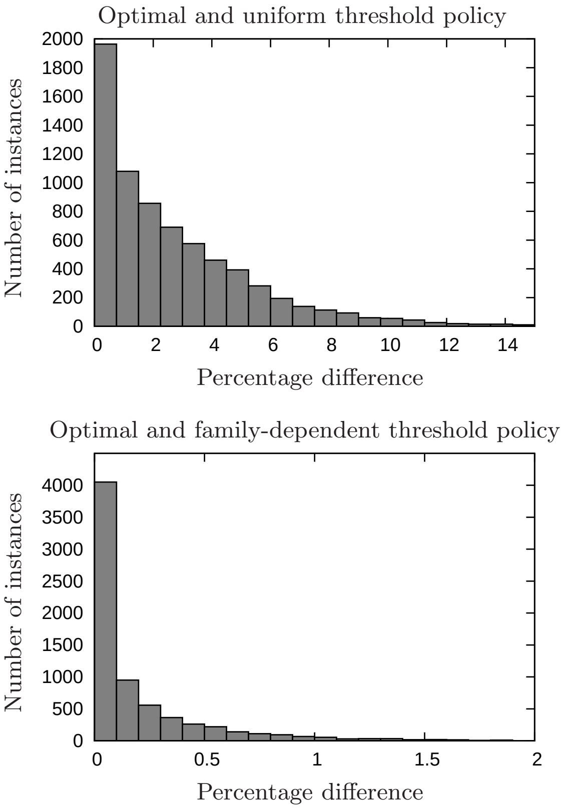

Figure 4.5: Histogram of percentage of performance difference between the optimal and uniform threshold policy (upper panel) and the optimal and family-dependent threshold policy (lower panel) for a large number of asymmetric instances.

The upper histogram shows that the threshold policy is rather robust to asymmetries in the family parameters. For the majority of cases the best threshold policy performs within a few percent of the optimal policy. The lower histogram shows that the best family-dependent threshold policy achieves near optimal performance; it never performs less than 2% optimal, and in the vast majority of the cases the performance is within 0.1% optimal.

4.6 Conclusions and Extensions

We developed a Markov decision model of the customized stochastic lot scheduling problem with strict lead times and rejections. With this MDP we can compare a heuristic policy (the threshold policy) to the optimal policy of the MDP. It is shown, within the numerical regimes considered here, that when the families are symmetric in job size, arrival rate and reward, it is possible to find a threshold parameter c such that the difference in performance between the threshold and optimal policy is negligible. Moreover, the performance of the threshold policy proves actually rather robust to asymmetries in the family parameters. This is rather remarkable as the heuristic is primarily designed to resolve contention for the machine capacity among symmetric families. Second, it appears that the threshold policy is usually more fair than the optimal policy in that, for instance, the family with the smallest reward per order, receives a higher share of the capacity under the heuristic than under the optimal policy. Third, by making the threshold c family-dependent, i.e., c_f for family f, the threshold policy can again be tuned to perform (near) optimal for asymmetric cases. The analysis based on MDPs has a clear advantage in that it is exact, rather than heuristic. On the other hand, this approach suffers from a state space explosion, thereby making it difficult to study situations with many families, long lead times, or many types of actions.

The Markov decision model of the CSLSP makes it relatively straightforward to consider numerous extensions. (To keep the number of variations within this study within limits we chose not to incorporate these extra degrees of freedom here.) For instance, job sizes may become random, as long as the support of distribution is restricted to, for instance, $\{1, \dots, 8\}$ when the number of families is 3 and the lead time is 30. The distribution may also become family dependent, just as setup times. It is also possible to make the service time stochastic, by a simple modification of the shift operator T. However, when service times become stochastic, the lead times may no longer be guaranteed, hence the model should be extended with other performance measures, for instance tardiness.

As a first attempt to gain insight into good acceptance and scheduling policies for the CSLSP, we restricted the set of scheduling actions to the two actions that are most interesting for the planner: Combine and Spawn. In the original CSLSP, however, any action leading to a permutation of the schedule in which orders are produced in time is an admissible scheduling action for the planner. A next suggestion for further research is to refine and extend the set of admissible scheduling actions available to the planner. An interesting action to consider is 'Change Sequence' which refines the Combine action by allowing the planner to change the sequence of runs such that an arriving order can be combined with a run of the same family in the schedule.

The exploration of family-dependent lead times is of interest as well. However, as lead time differences appears to give rise to strong unfairness among the product families, it is unclear whether this will be a practically viable implementation. We address this problem in the next chapter.

It would be also of interest to find a simple analytic model to estimate the good (optimal) values, possibly family dependent, for the threshold parameter c. The current frequency table approach works well, but is based on the knowledge of the optimal policy. However, for large non-trivial systems it is impossible to find an explicit form for the optimal policy, at least within reasonable time and resources. Hence, we need other methods to efficiently find the best values for the thresholds.

A final suggestion for further research is to consider a mixed environment in which only some orders have a strict lead time, while other orders are make-to-stock. These latter orders can be used to fill the schedule and produce longer runs. On the other hand, inserting such jobs somewhere in the 'middle' of the schedule pushes back orders with strict lead times, so that there is less room for future combinations. It would be of practical interest to design simple but useful insertion decisions rules for this mixed case.

Chapter 5

Order Acceptance: Family-Dependent Lead Times

To make the order acceptance and scheduling decisions for the production situation of Chapter 4 (CSLSP) easier, it is common practice to use standard lead times for all families. That is, the supplier commits itself to delivering all orders within the same lead time independent of the family of the order, so that as a consequence all lead times are equal to the shortest desirable lead time. While allowing for family-dependent lead times makes the planning situation more complicated, it may result in a significant gain in profit compared to using standard lead times as more orders can be accepted. However, it is not so clear from literature how order acceptance and scheduling policies can exploit lead time differences, whether the gain in profit outweighs the extra efforts of dealing with additional planning complexity and how this trade-off depends on the characteristics of the production situation.

The goal of this chapter is to develop a tractable and implementable policy for the CSLSP with family-dependent lead times and offer managerial insights into when it is attractive to use family-dependent or standard lead times. We do so by extending the Markov decision process (MDP) model developed in Chapter 4 such that it allows for family-dependent lead times. Our numerical results show that for a large number of instances the optimal order acceptance and scheduling policy for the MDP can use the differences in family lead times to generate significant benefits for the supplier. A detailed analysis of the data provides some insight into when it is (not) interesting to allow for family-dependent lead times, although it remains difficult to predict by how much precisely the performance will increase. We also develop a set of practically simple (threshold) policies to control the acceptance and scheduling of orders and present a method to tune the involved thresholds. Numerical results show that the threshold policies perform (very) well compared to the optimal policy in a wide range of parameter settings, including product family asymmetries in arrival rate, job size, job reward, and lead times.

5.1 Introduction

We consider a supplier in the batch process industry that sells different items to multiple downstream customers. This supplier has limited capacity available to serve randomly arriving customer orders. Orders are grouped into families with similar production characteristics, and when production changes from one family to another, a setup time is incurred. The supplier is allowed to reject orders, but if he accepts an order he commits to delivering it in time. Accepting orders is interesting as it results in a family-dependent reward. Clearly, one of supplier's goals is to maximize the long run profit. This, however, is a difficult problem since the acceptance decisions and scheduling decisions, due to the setup times, interact. This joined acceptance and scheduling problem is commonly referred to the Customized Stochastic Lot Scheduling Problem (CSLSP) with strict lead times, c.f. Winands et al. (2011) and Chapter 4.

Typically suppliers make such production situations somewhat simpler by imposing a (hierarchical higher level) planning structure. One traditional approach is to generate *cyclic production schemes* and use these schedules to quote customer order delivery dates, see e.g., Bertrand et al. (1990). This procedure, however, leads to highly variable due-date quotations. The delivery date may be quite early (the process has just been set up for that product family), but may also be quite far away in the future (the manufacturing of that product family has just been finished, and the order has to wait for the complete length of the cycle). Clearly, long or unpredictable supplier lead times are unattractive for customers downstream in the supply chain: in make-to-stock situations it leads to maintaining large downstream stock levels, while in make-to-order situations it leads to low customer satisfaction. Thus, to limit the lead times, and the inherent variability, cycle times are usually rather short. This, however, has another drawback: the utilization is low because the fraction of time spent on setups is large. Moreover, since orders arrive randomly so that there may be low demand for some families during the cycle horizon and large demand for others, the predefined cyclic structure hinders the exploitation of streaks of good luck. In particular, Van Foreest et al. (2010) show for the CSLSP with strict lead times that simple threshold policies considerably outperform cyclic policies. Thus, cyclic policies, although simple to understand, have considerable drawbacks in terms of utilization of the bottleneck and due-date variability.

Another approach to reduce the complexity of the production situation, for instance in MRP, is to quote standard lead times. That is, the supplier commits itself to delivering all orders within the same lead time independent of the family of the order, so that as a consequence all lead times are equal to the shortest desirable lead time. One reason to use standard lead times is that it is not so clear how to exploit situations with family-dependent lead times. Families with long lead times have more 'opportunity' to claim the available capacity, thereby potentially leading to unfair division of capacity among the families. As an undesirable consequence, the long run profit of a naive policy might even decrease in a production situation in which low reward families are offered long lead times as compared to a situation in which the low reward families have the same (or shorter) lead times than the high reward families. The use of standard lead times is therefore often seen in process industries, see e.g. Ten Kate (1995).

However, allowing for family-dependent lead times may result in a significant gain in profit compared to using standard lead times as more orders can be accepted. An interesting problem is therefore to investigate whether the gain in profit outweighs the extra efforts of dealing with additional planning complexity and how this trade-off depends on the characteristics of the production situation. The goal of this chapter is to develop a tractable and implementable policy for the CSLSP with family-dependent lead times and offer managerial insights into when it is attractive to use family-dependent or standard lead times. We do so by extending the Markov decision process (MDP) model developed in Chapter 4 such that it allows for family-dependent lead times. Our numerical results show that for a large number of instances the optimal order acceptance and scheduling policy for the MDP can use the differences in family lead times to generate significant benefits for the supplier. A detailed analysis of the data, however, shows that it is difficult to find structure in how individual model parameters influence the extra gain that can be obtained from using family-dependent lead times. To provide more insight, we introduce the concept of relative reward rate γ_f for family f, which is a measure that considers several model parameters jointly. It turns out that γ_f is a reasonable indicator when it is (not) interesting to allow for family-dependent lead times, although it does not predict by how much precisely the performance will increase.

The optimal policy we find by solving the MDP is not very useful since it is too complex for practical purposes. We therefore also develop a set of practically simple (threshold) policies to control the acceptance and scheduling of orders and present a method to tune the involved thresholds. Numerical results show that the threshold policies perform (very) well compared to the optimal policy in a wide range of parameter settings, including product family asymmetries in arrival rate, job size, job reward, and lead times.

The structure of the chapter is as follows. In Section 5.2 we describe the production situation and the allowed (admissible) decisions. Section 5.3 discusses the structure of the threshold policies. In Section 5.4 we study the performance of the optimal policy, and evaluate the performance of heuristic policies. Here we also introduce a refinement of the threshold policy introduced earlier and show that this refinement is a substantial improvement. Section 5.5 concludes. Finally, we refer to Section 4.2 for a review on literature related to the CSLSP.

5.2 Model Framework

In Section 5.2.1 we present a model of the CSLSP with family-dependent lead times. In Section 5.2.2 we address the decision structure of policies that accepts/rejects and schedules orders. Finally, Section 5.2.3 defines relevant performance indicators and the objective function.

5.2.1 Model

The production situation at the supplier is modeled as a single machine that receives a stream of orders at arrival epochs $0 = T_0 \leqslant T_1 \leqslant T_2, \ldots$. We assume that the interarrival times of customer orders $T_{i+1} - T_i$ are independent and identically distributed, integer valued, and geometrically distributed with success parameter p, that is, according to

$$\mathrm{P}(T_{i+1} - T_i = k) = p(1-p)^k, \quad \text{for } k = 0, 1, 2, \ldots$$

Observe that the inter-arrival time can be zero and thereby multiple orders may arrive at the same time. Since $\mathrm{E}\,(T_{i+1} - T_i) = (1-p)/p$, the arrival rate of customer orders $\lambda = p/(1-p)$. Arriving orders belong to family f, which is one of N possible families, with probability q_f, independent of anything else. Thus, the arrival rate of family f is $\lambda_f = \lambda q_f$. A job of family f requires b_f time units of service, where b_f is deterministic and integer valued. A job can be rejected upon arrival, but if accepted it is scheduled for service such that it can be produced within a constant lead time of length h_f, which is family dependent. Whenever two subsequent orders in the schedule belong to different product families a setup of (integer valued) duration s, which is the same for all families, is inserted between these two orders. When an order arrives at an empty system, a setup is not necessary if the last produced order is of the same family as the arriving order. Service of orders and setups is non-preemptive, and the server is assumed to never fail, so that all accepted orders can be produced in time.

The reward structure for the supplier is simple. We set the acceptance reward for a job of family f equal to $r_f > 0$ and the earliness cost to zero. The underlying motivation is that we assume that serving orders early is acceptable when this potentially leads to accepting more orders. Hence, the reward for accepting an order must be higher than the cost of producing early. We assume without loss of generality that there is no penalty associated with rejecting an order. We also take the setup cost equal to zero. However, inserting setups arbitrarily cannot be optimal, since a setup takes away a position in the schedule thereby preventing potential rewards. There is no tardiness cost, as jobs cannot be late.

5.2.2 Decisions

Clearly at each arrival epoch it is necessary to decide whether to accept or reject the arriving order, and if it is accepted, it is required to decide where to insert the order in the schedule. In this section we illustrate the acceptance/rejection and scheduling decisions available to (the planner of) the production system by means of an example. The formal specification of these decisions is rather involved. We refer the reader to Section 4.4 for the details.

Consider first the reference schedule in Table 5.1 which contains orders and setups, all of unit length. We assume 8, 15, and 8 as lead times for families 1, 2, and 3, respectively. Clearly, the schedule contains two *runs* of orders: the first run is of family 1 and contains two orders; the second run is of family 2 and contains three orders. We also assign a *slack* to each accepted job, which is the amount of time that is available to insert other, later arriving, orders in front of the accepted job. For instance, the order in position 4 has a due-date of 12 and therefore 8 time units are available to insert other jobs in front of it before it will be late. Since setups cannot be late they do not have a slack. Below we discuss the Combine, Spawn and Reject action.

Table 5.1: The reference schedule.

Position	1	2	3	4	5	6
Family	1	1	s	2	2	2
Due-date	3	4	-	12	14	15
Slack	2	2	-	8	9	9

The *Combine* action tries to combine an arriving order with a run of its ‘kin’ in the schedule, with the aim of reducing the fraction of setups. By simple pairwise interchange arguments, see e.g., Pinedo (2008), it is easy to see that the optimal sequence of jobs within a run satisfies the Earliest Due-Date (EDD) rule—the reward cannot increase by inserting arriving orders before orders of the same family—hence, the Combine action adjoins accepted orders of family f only to the *end* of the last run of family f in the schedule. To illustrate, suppose an order of family 1 arrives with a lead time of 8 time units. The schedule in Table 5.1 contains one run of family 1, that is, at positions 1 and 2. The Combine action tries to join the new order with this run by inserting it at position 3 and shifting the already present orders at positions 4–6 back to positions 5–7. Clearly, the shifted orders will not be too late as a result of the insertion. The Combine action then leads to the new schedule depicted in Table 5.2. The Combine action is not allowed in case one of the shifted orders would have been too late.

Table 5.2: The schedule that results after a new order of family 1 has been combined at position 3 at the end of the first run.

Position	1	2	3	4	5	6	7
Family	1	1	1	s	2	2	2
Due-date	3	4	8	-	12	14	15
Slack	2	2	5	-	7	8	8

The *Spawn* action tries to ‘spawn’ a new generation of its family in a fashion similar to the EDD rule. Suppose an order of family 3 arrives with a lead time of 8 time units. The schedule in Table 5.1 contains no run of family 3 and therefore in order to accept the new order it is necessary to spawn a new run of family 3. Observe that there is sufficient slack to spawn a new run of family 3 at three positions in the schedule: at the beginning of the schedule; after the run of family 1; and after the run of family 2. However, as service of orders and setups is non-preemptive, it is not allowed to spawn a run at the beginning of this schedule; thus the first option is not allowed. Table 5.3 shows the resulting schedule after spawning the arrival behind the run of family 1 and the run of family 2. We see this schedule is in EDD order, and, it provides *combination potential*: since the order of family 3 has a slack of 4 it still allows to accept orders in front of it.

Table 5.3: The schedule that results after an order of family 3 is spawned in Schedule 1 behind the run of family 1.

Position	1	2	3	4	5	6	7	8
Family	1	1	s	3	s	2	2	2
Due-date	3	4	-	8	-	12	14	15
Slack	2	2	-	4	-	6	7	7

In Table 5.4 the order of family 3 has been spawned behind the run of family 2. Now the slack of the new order has been reduced to zero, and we say that this order is *tight*, which implies that no other order can be inserted in front of it.

Table 5.4: The schedule after the order of family 1 is spawned in Schedule 1 behind the run of family 2.

Position	1	2	3	4	5	6	7	8
Family	1	1	s	2	2	2	s	3
Due-date	3	4	-	12	14	15	-	8
Slack	2	2	-	6	7	7	-	0

It is clear that the schedule in Table 5.3 is the more favorable of the two. We therefore allow the Spawn action to only insert new runs such that the EDD order is maintained within the schedule after the acceptance.

These examples also show that the Combine action does not insert a setup, while the Spawn action does. For this reason we prefer the Combine action over the Spawn action when the schedule allows both actions.

The *Reject* action is trivial: it just rejects the arriving order.

5.2.3 Performance Indicators and Objective Function

To define the performance indicators of interest for a given acceptance and scheduling policy π let $A^\pi(k)$ be the sum of the rewards of accepted orders among the first k arrivals. Then the *long run expected reward per arriving order under policy* π, $J(\pi)$, takes the form

$$J^\pi = \lim_{k\to\infty} \frac{\mathrm{E}\left(A^\pi(k)\right)}{k}. \tag{5.1}$$

The objective is to find the maximal long run average reward of the production system and the optimal policy $\pi^\star$, i.e., we solve

$$J^\star := J^{\pi^\star} = \max_\pi J^\pi, \qquad \pi^\star = \arg\max_\pi J^\pi, \tag{5.2}$$

where the maximization is taken over the class of stationary and non-anticipative (with respect to the arrival process) policies.

In the numerical analysis we use policy iteration, (see, e.g., Tijms, 2003, Chapter 6), to compute the performance measures.

5.3 Heuristic Policies

In this section we introduce two simple heuristic policies: a *greedy* policy and a *threshold* policy. It is known, see Van Foreest et al. (2010), that the greedy policy does not work well under high loads. However, as this policy is nearly trivial, while the threshold policy still requires some slight tuning (of the family-dependent thresholds), it is of interest to see how much can be gained by the threshold policy as compared to the greedy policy.

The greedy policy first tries to combine a new arriving order with a run. If Combine is not possible, it tries to spawn the order. If both Combine and Spawn are not allowed, the policy rejects the order altogether. Observe that the greedy policy does not restrict the Spawn decision at all. However, it is apparent that a good policy should regulate, in some sense, this action. To see this, suppose that a new arriving order spawns a new run and suppose that this run is tight. Then no further order can be combined in front of this new run, thereby removing all combination potential in the schedule.

A simple heuristic method to prevent such ‘tight’ spawns to occur is to use a threshold for the Spawn decision: a new run may only start when the first job in the new run is not too tight (i.e. has sufficient due-date slack), thereby leaving room for later orders to combine with other runs in the schedule. This idea results in the threshold policies we study in this chapter. Given a set of family-dependent threshold parameters $0 \leqslant c_f < h_f$, where h_f is the family-dependent lead time, the heuristic threshold policy has the following structure:

- Choose Combine if the schedule and the arriving order allow this.
- If Combine fails, choose Spawn only if allowed and the due-date slack of the arriving order is greater or equal than c_f after acceptance of the order.
- otherwise, Reject.

Observe that the greedy policy can be obtained as a special case of this threshold policy by setting $c_f = 0$ for all families f, that is, spawning a run is always allowed if it is possible.

A remaining issue is to actually determine the best threshold. In Section 5.4.4 we describe this in detail.

5.4 Numerical Study

In this section we investigate the influence of quoting family-dependent lead times on the performance of the CSLSP under the optimal, threshold and greedy policy for the scenarios discussed next.

5.4.1 Scenarios

To investigate the effect of using family-dependent lead times instead of standardized lead times on the system performance, we compute the long run expected reward per arriving order for a full factorial design of scenarios, with parameters as specified in Table 5.5, which lead to some 17K scenarios.

Table 5.5: The parameter values considered in the full factorial design.

# Families	Parameters	Values
$N = 2$	Load	$\rho = 0.7, 0.9, 1.1$
	Setup time	$s = 1$
	Job size	$b = (1, 1), (1, 2), (2, 1)$
	Lead time	$h_1 = 6, 8, 10; \quad h_2 = h_1, h_1 + 2, \ldots, 24$
	Reward	$r_1/b_1 = 1; \quad r_2/b_2 = 0.5, 0.75, \ldots, 1.5$
	Arrival fraction	$q = (0.1, 0.9), (0.2, 0.8) \ldots, (0.9, 0.1)$
$N = 3$	Load	$\rho = 0.7, 0.9, 1.1$
	Setup time	$s = 1$
	Job size	$b = (1, 1, 1), (1, 1, 2), (2, 2, 1)$
	Lead time	$h_1 = h_2 = 6, 8, 10; \quad h_3 = h_1, h_1 + 2, \ldots, 20$
	Reward	$r_1/b_1 = r_2/b_2 = 1; \quad r_3/b_3 = 0.5, 0.75, \ldots, 1.5$
	Arrival fraction	$q = (1, 1, 1)/3, (1, 1, 2)/4, (2, 2, 1)/5, (2, 1, 1)/4,$ $(2, 1, 2)/5$
$N = 4$	Load	$\rho = 0.7, 0.9, 1.1$
	Setup time	$s = 1$
	Job size	$b = (1, 1, 1, 1), (1, 1, 1, 2), (2, 2, 2, 1)$
	Lead time	$h_1 = h_2 = h_3 = 6, 8; \quad h_4 = h_1, h_1 + 2, 10, 12$
	Reward	$r_1/b_1 = r_2/b_2 = r_3/b_3 = 1;$ $r_4/b_4 = 0.5, 0.75, \ldots, 1.5$
	Arrival fraction	$q = (1, 1, 1, 1)/4,\ (1, 1, 1, 2)/5,\ (2, 2, 2, 1)/7,$ $(2, 2, 1, 1)/6,\ (1, 1, 2, 2)/6$

For notational convenience we use vectors h, b, q and r, to denote the lead times, job size, arrival fraction and reward per family. For instance, $b = (b_1, b_2) = (1, 2)$

specifies the scenario in which the job size of the first (second) family is 1 (2).

The motivation behind the choices is as follows. The load of the system $\rho = \lambda \sum_f q_f b_f$ varies from light (0.7) to heavy (1.1); higher or lower values appear less relevant to study from a practical point of view. The job size b_f can be small (1) or large (2). The parameters N and h determine the size of the state space and are therefore bounded by computer memory. To analyze the effect of the increasing lead times of product families we increased the lead time of the last family (i.e. family N) in steps of 2 time units. Hence for each scenario with fixed N, ρ, b, r, q, s we have a 'reference' case with standard lead times $h = (h_1, \ldots, h_1)$ and cases with increased lead times for family N, i.e. $h = (h_1, \ldots, h_1, h_N)$ with $h_N > h_1$. Finally, we set the reward per unit processing time r_f/b_f of the families with standard lead times to 1, and vary the reward per unit processing time of family N from small (0.5) to high (1.5).

5.4.2 Effect of Family-Dependent Lead Times on the Reward per Arriving Order

As a measure to compare the influence of family-dependent lead times we use the *relative gain*

$$R^\star = \frac{J^\star(h_N)}{J^\star(h_1)},$$

where $J^\star(x)$ is the long run expected reward per arriving order associated with the optimal policy for a scenario with fixed N, ρ, b, r, q, s and $h = (h_1, \ldots, h_1, x)$. Thus, we compare the long run expected reward per arriving order of a scenario with $h = (h_1, \ldots, h_1, h_N)$ to the 'reference' case with standard lead times $h = (h_1, \ldots, h_1)$.

Figure 5.1 shows *jitter* plots[1] for the relative gain of the optimal policy for all scenarios of Table 5.5 as a function of h_N. Taking $h_1 = 6$, 8, and 10, in the upper middle, lower left, and lower right panel shows the influence of h_1. The results in all three plots show clearly that $R^\star$ never decreases, and that the gain can be substantial, sometimes even up to more than 15%. The figures show that the

[1] A jitter plotter is a scatter plot in which each data point is shifted by a random amount. The aim of a jitter plot is to make the clustering in a collection of data points visually more clear. This graph is created by shifting each data point horizontally by a random amount. If we'd just plotted the data in a true, two-dimension fashion, too many of the points would've overlapped, making it difficult to detect clustering. See Janert (2010) for more information on jitter plots.

highest relative gains are obtained for scenario's that have the smallest values of h_1, which is intuitive since for these scenario's the reference cases have the tightest lead times.

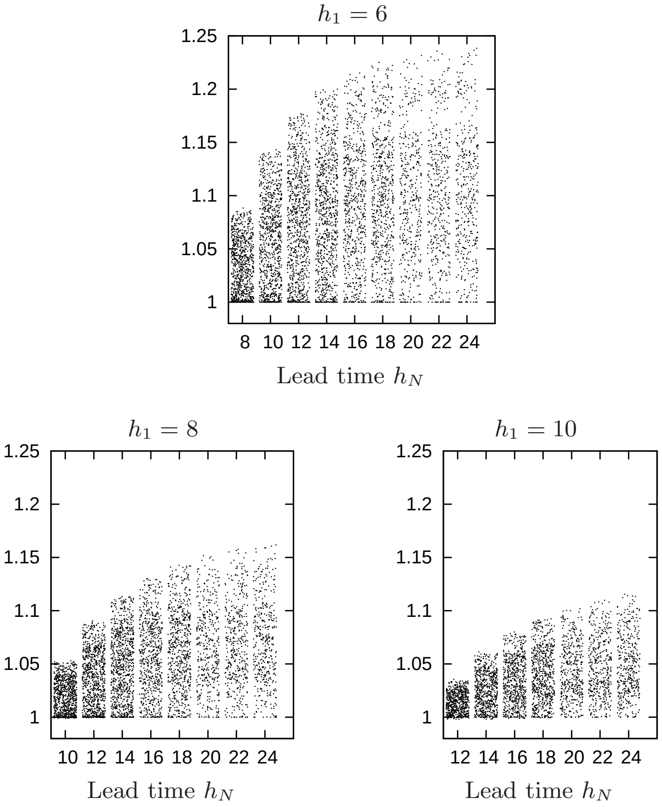

Figure 5.1: Jitter plots of the relative gain $R^\star$ as a function of the lead time h_N for the optimal policy. The upper middle, lower left and lower right panel correspond to $h_1 = 6, 8$, and 10.

In trying to understand better how individual model parameters influence the relative gain we made numerous multi-dimensional scatterplots of the data[2]. This further analysis, however, did not reveal any significant structure. A measure that provides more insight is the *relative reward rate* of family f

$$\gamma_f = \frac{\beta_f}{N^{-1}\sum_{i=1}^{N}\beta_i}, \tag{5.3}$$

where $\beta_f = \lambda q_f r_f / b_f$ is the reward rate of family f per unit job. Thus, families

[2] The data files can be obtained from the author at request.

with $\gamma_f > 1$ 'bring in relatively more money' per unit job. We therefore expect that increasing the lead time of orders with high γ has a stronger influence on $R^\star$ than increasing the lead time of orders with a low γ.

Next, for a given bar of dots in Figure 5.1 we are interested in whether γ can indeed explain if a scenario ends up in the upper or lower part of bar. For instance, take the third bar in the upper middle panel of Figure 5.1 corresponding to $h_1 = 6$ and $h_N = 12$. Then we would like γ to explain for each scenario in this bar if it has a high or low value of $R^\star$. To visualize the dependence of $R^\star$ on γ for given values of h_1 and h_N, Figure 5.2 shows $R^\star$ as a function of γ_N for scenario's with $h_1 = 6$, $h_N = 12$ (left panel) and $h_1 = 6$, $h_N = 24$ (right panel). The figure shows that $R^\star$ increases up to $\gamma_N \approx 1.3$ but then starts to decrease. The reason is that here the arrival fraction of family N is high compared to the other families, so that there is not much competition for the capacity between the families (family N orders claim most). The graphs for other parameter values (not included here) show similar behavior so that we conclude that γ_f can serve as a rough indicator whether family f should be allowed longer lead times.

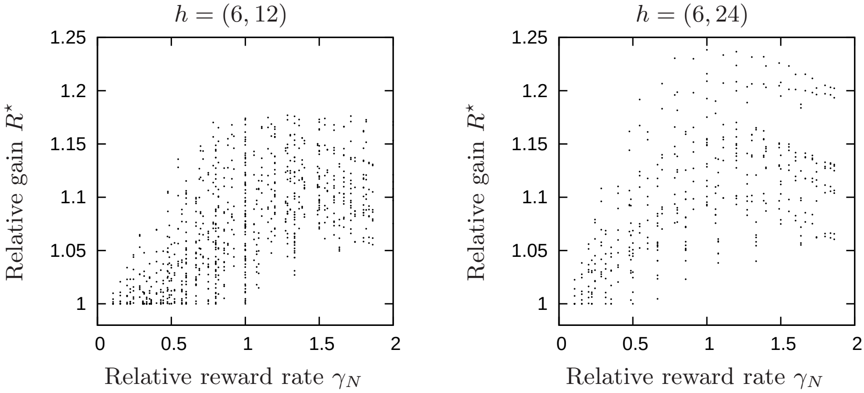

Figure 5.2: The relative gain $R^\star$ as a function of the relative reward rate γ_N for the optimal policy. The left and right panel correspond to $h = (6, 10)$ and $(6, 24)$.

5.4.3 Analysis of the Threshold Policy

The results of the previous subsection show that under the optimal policy significant extra rewards can be obtained when the lead times of families with a high relative reward rate γ are allowed to become longer than the lead times of the other families. Chapter 4 showed that the structure of the optimal policy is hard to characterize. In this section we explore the performance of the simpler heuristic policies of Section 5.3.

Similar to Figure 5.1 we show in Figure 5.3 the relative gain $R^t = J^t(h_N)/J^t(h_1)$, where $J^t(\cdot)$ is the long run expected reward per arriving order achieved by the threshold policy.

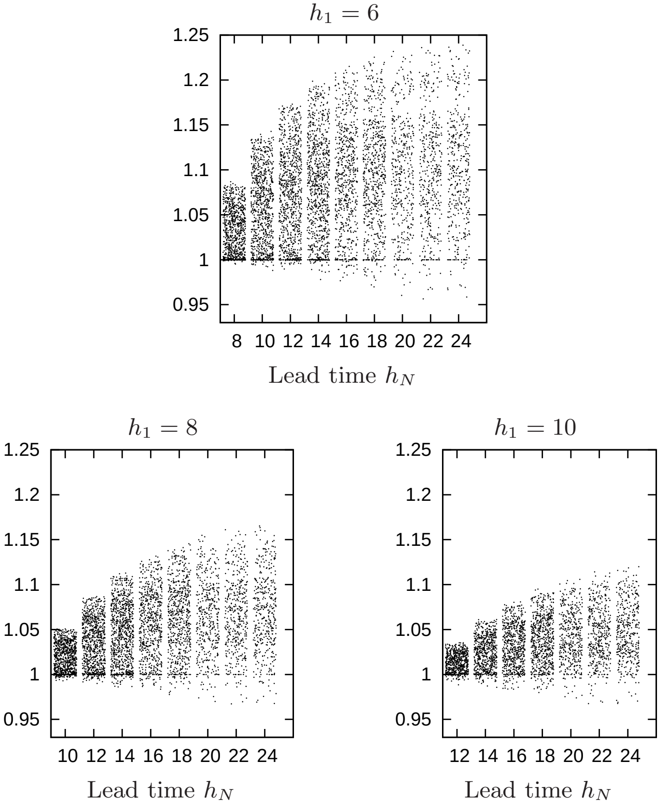

Figure 5.3: Jitter plots of the relative gain R^t as a function of the lead time h_N for the threshold policy. The upper middle, lower left, and lower right panel correspond to $h_1 = 6, 8$, and 10.

The jitter plots show that in the majority of the cases the threshold policy is also able to achieve a significant gain. This is, however, not always the case: we see that for some situations the relative gain R^t is less then one, so that it is actually detrimental to the system as a whole to allow one family to have a larger lead time than the other families.

To obtain a better understanding of the occasional poor performance of the threshold policy we use a *frequency table approach*, c.f., Haijema et al. (2009) and Section 4.5.2, to analyze one such problematic case. Table 5.6 shows a frequency table for Combine–Reject decision of the optimal policy of the scenario $N = 2$, $h = (6, 8)$, $\rho = 1.1$, $s = 1$, $b = (1, 1)$, $r = (1.0, 0.5)$, and $q = (0.5, 0.5)$.

Table 5.6: Frequency table for Combine–Reject decision of the optimal policy of the scenario $N = 2$, $h = (6, 8)$, $\rho = 1.1$, $s = 1$, $b = (1, 1)$, $r = (1.0, 0.5)$, and $q = (0.5, 0.5)$.

	L	0	1	2	3	4	5	6	7	Total
Fam. 1	Combine	3.74	6.20	10.64	7.49	5.53	4.49	—	—	39.09
	Reject	0.00	0.00	0.00	0.00	0.00	0.00	—	—	0.00
Fam. 2	Combine	1.53	3.27	5.49	1.48	0.00	0.00	0.00	0.00	11.77
	Reject	0.00	0.19	1.30	1.75	0.74	0.17	0.14	0.08	4.37

Let L_f be the length of the run to which an arriving order of family i can be joined in the schedule (for instance, $L_1 = 2$ in the scenario of Table 5.1 and $L_2 = 3$). Then, for instance, the number 10.64 in the first row is the percentage of arriving orders for which the following three conditions are satisfied simultaneously: 1) the arriving order belongs to family 1; 2) the optimal policy chooses to combine it; 3) the length of the run to which the arriving order can be joined in the schedule is $L_1 = 2$. The number 1.30 in the fourth row is the percentage of arriving orders that belong to family 2 but are rejected by the optimal policy since the run length is $L_2 = 2$.

Now it is evident from Table 5.6 that the optimal policy always combines jobs of family 1 whenever possible; thus, the Combine–Reject decision of the threshold policy and the optimal policy are identical for this family. However, for orders of family 2 the behavior of the optimal policy is less simple. In fact, when $L_2 = 1$ the optimal policy rejects the arriving order for some states, and for other states it combines the order; we see similar behavior when $L_2 = 2$ and 3. Thus, for family 2 the optimal policy and the threshold policy are not identical. A similar analysis carried out for the Spawn–Reject decision shows that the Spawn–Reject decision of the optimal policy is of threshold type.

In hindsight, the fact that the optimal policy does not use a threshold for family 2 orders is natural. Family 2 orders are less profitable than family 1 orders. Thus, allowing runs of family 2 orders to grow and thereby claim capacity results in less room for the more profitable orders of family 1. This observation also shows the flaw in the threshold policy: It accepts any order whenever this order can be combined, even when such an order is unattractive on the long run.

5.4.4 An Improved Threshold Policy

To repair the deficiency just indicated we propose to adapt the threshold policy such that it also includes a threshold on the Combine action. Given a set of family-dependent threshold parameters $0 \leqslant c_f < h_f$ and $0 \leqslant d_f \leqslant h_f$, the improved threshold policy has the following structure:

- If Combine is allowed, choose Combine only if $L_f < d_f$.
- If Combine fails, choose Spawn only if allowed and the due-date slack of the arriving order is greater or equal than c_f after acceptance of the order.
- otherwise, Reject.

Before we present the performance of this improved threshold policy we describe how we use frequency analysis to obtain good thresholds for the Combine action. The idea is to set the threshold such that the improved policy and the optimal policy have maximal overlap. Formally, write $f_{f,C}(j)$ ($f_{f,R}(j)$) for the percentage of states for which the optimal policy chooses to combine (reject) an order of family f when $L_f = j$. For example, in Table 5.6, we have $f_{2,C}(2) = 5.49$. Now compute for all levels L_f the number

$$F_f(L_f) = \sum_{j=0}^{L_f-1} f_{f,C}(j) + \sum_{j=L_f}^{h_f} f_{f,R}(j)$$

and set the threshold d_f at the level at which $F_f(L_f)$ is maximal. Thus, for the case of Table 5.6, we take $d_1 = h_1 = 6$. That is, no matter the length of the run of family 1 orders, the new order is combined with this run. For family 2, we see that $F_2(0) = 4.37$, $F_2(1) = 1.53 + 4.37 = 5.90$, etcetera, leading to an optimal threshold value of $d_2 = 3$.

To find optimal thresholds for the Spawn–Reject decision we use a similar approach. Note for completeness that in this case we do not use the family run length L_f of the arriving order, but the due date slack as explained in Section 5.3.

The relative gains R^{it} for this improved threshold policy are shown in Figure 5.4. It is apparent that the improved threshold policy works much better. Hardly any 'bad' scenarios remain, and even when the performance does decrease it is by a tiny amount (i.e., less than 0.5%). A similar comparison shows that the 'old' threshold policy never outperforms the improved policy. This is as expected since the improved policy contains the 'old' threshold policy as a special case.

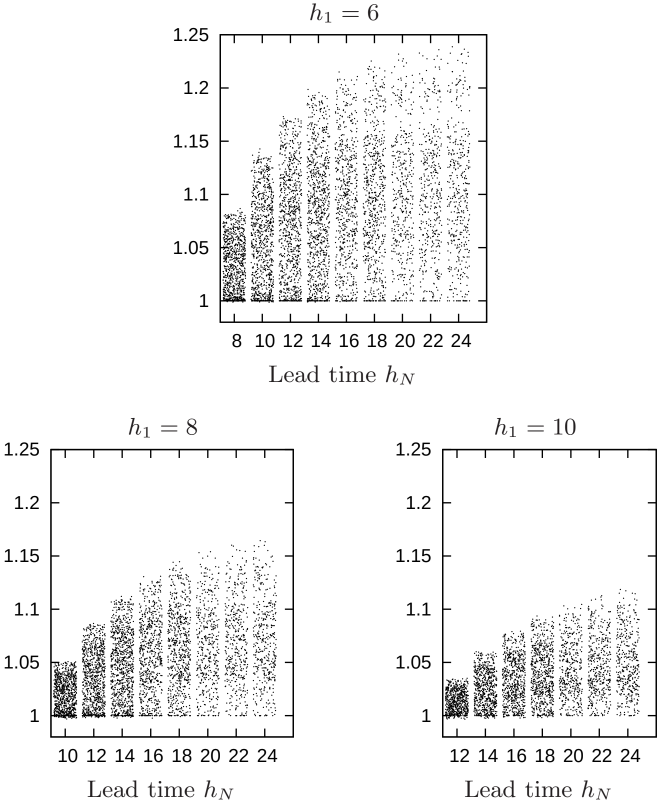

Figure 5.4: Jitter plots of the relative gain R^{it} as a function of the lead time h_N for the improved threshold policy. The upper middle, lower left, and lower right panel correspond to $h_1 = 6, 8$, and 10.

We finally compare in Figure 5.5 the performance of all three heuristic policies to the optimal policy. The figure shows three jitter plots of $J^p(\cdot)/J^\star(\cdot)$ where p indicates, in order, the greedy policy (g), the threshold policy (t) and its improved version (it). These jitter plots make dramatically clear that the improved threshold policy is better than the old threshold policy, which in turn is (much) better than the greedy policy. Note also that the improved threshold policy performs within 1% of the optimal policy.

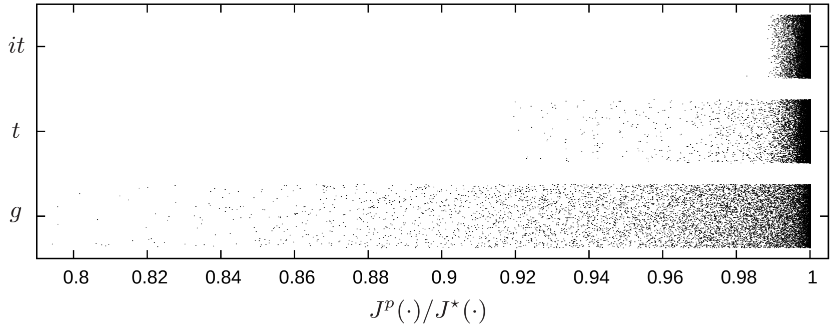

Figure 5.5: Jitter plots of the long run expected reward per arriving order of the greedy (g), threshold (t) and improved threshold (it) policies relative to the performance of the optimal policy.

5.5 Summary and Extensions

In this chapter we considered the Customized Stochastic Lot Scheduling Problem (CSLSP) in which orders have family-dependent strict lead times, i.e., any order that cannot be delivered before its due-date has to be rejected at arrival. Moreover, we allowed the lead time to be dependent on the family. The reason to consider such systems is that previous work shows that extending the lead time can significantly increase the performance of the system. However, not all families allow such long lead times. Thus, it might be interesting to allow for differences in the family lead times, such that the families that require short lead times do not unnecessarily constrain the lead times of the other families, and the system can still benefit by achieving a higher average reward per arriving order.

As in Chapter 4 we modeled the production system as a Markov decision process (MDP) and computed the optimal policy. We showed for a large number (> 17K) of scenarios that under the optimal policy of the MDP an increase in lead time of one family never has a detrimental effect on the performance; in fact in many cases significant extra rewards can be obtained. However, increasing one lead time does not always results in extra rewards. To better understand when it is (not) interesting to allow for asymmetric lead times we introduced the concept of relative reward rate γ_f for family f. It turns out that γ_f is a reasonable indicator when some lead times can be increased, however, it does not predict by how much precisely the performance will increase. It would certainly be of interest to develop a simple analytical model to estimate the system performance.

Van Foreest et al. (2010) contains one such model, but we have been unable to extend this such that it can cope with the situation we deal with in this chapter. Such models should be helpful to understand in which cases the long run profit increases when some families are allowed to use longer lead times.

We then developed a class of policies that uses a threshold to control the Spawn action. A performance analysis shows that this class of policies works well in the majority of the cases, but not always so. In fact, in some instances the performance actually degrades under the threshold policy when the lead time increases, contrary to what we see for the optimal policy: then the performance never becomes less. Thus, this class of threshold policies is not entirely robust.

To repair this problem we propose to also use thresholds for the Combine action. Interestingly, the best such improved threshold policy structurally perform within 1% of the optimal policy, and are robust against asymmetric lead times.

Finally, we included the results of the greedy policy, which can be considered as a special case of the threshold policies in the sense that all thresholds are 'switched off'. The greedy policy is very simple indeed, and it is of interest to see whether the more involved threshold policies provide sufficient extra reward to motivate their deployment. It turns out that the greedy policy does not perform well, hence we do not propose such policy as a viable option.

Chapter 6

Summary and Suggestions for Further Research

In this thesis we analyzed and developed order acceptance and order release policies to control queues in make-to-order (MTO) production systems. Controlling the time orders spend waiting in queues is crucial for realizing short and reliable delivery times, two performance measures which are of strategic importance for many MTO companies. Order acceptance and order release are the two most important production control mechanisms to influence the length of these queues. In order to draw our attention to major trade-offs that MTO companies have to consider in their order acceptance and order release decisions we focused our models on the main characteristics of MTO systems, such as random (batch) order arrival, routing variability, fixed capacities, setup times and due-dates.

The main research objectives of this thesis are: (i) to better understand the underlying mechanisms of good order acceptance and order release policies for MTO production environments; and (ii) to use these insights to develop simple order acceptance and order release policies to control queues in MTO production systems so that delivery dates can be met, whilst good use is made of the available capacity. In the first part of this thesis, that consists of Chapter 2, we focused on the order release decision while Chapters 3, 4 and 5 form the second part in which we looked at the order acceptance decision. In what follows, we summarize the main findings from each individual chapter and provide suggestions for further research.

In Chapter 2 we studied the throughput time performance of three order release policies that control the number of orders on the shop floor using no more than a set of cards: CONWIP, m-CONWIP and POLCA. Due to their ease of control, these so-called *unit-based* pull systems are widely implemented in practice. However, all research into the performance of these pull systems focuses on shop floor throughput time and workload levels and not on the performance indicator that is most relevant for MTO companies: total throughput time. Previous research also shows that the total throughput time performance of pull systems largely depends on their capability to create a balanced distribution of the workload among the workstations on the shop floor and that many systems lack this capability. We developed a simulation model to compare the throughput time performance of CONWIP, POLCA and m-CONWIP. Our simulations show that unit-based pull systems can improve the workload balance on the shop floor and reduce the average total throughput time. More specifically, CONWIP has no workload balancing capability and our results show that limiting the workload in a CONWIP controlled MTO production system increases the average total throughput time of orders. The overlapping loops in the POLCA system bring forward some workload balancing capability compared with CONWIP, but they do not perfectly detect and signal an imbalance in workload. As a result, POLCA faces a longer average total throughput time for a given shop floor throughput time than m-CONWIP, the system with the best workload balancing capability for the stylized production system we considered.

To identify whether unit-based pull systems can reduce the average total throughput time, we simulated an MTO system with divergent routings and three production stages in each routing. This divergent ‘topology’ perfectly suits pull systems that are able to balance workload. A suggestion for further research is to extend our research to other topologies such as convergent and Jackson networks or typologies with longer routings. In the paper of Ziengs et al. (2011) we make a first a start in this direction by studying the workload balancing capability of POLCA for a divergent topology with long routings.

A next suggestion for further research is to use a queueing approach to analyze the throughput time performance of unit-based pull systems. The main advantage of a queueing approach compared to simulation is that it allows for a quick evaluation of many alternatives of pull system configurations for a given (MTO) production system. For instance, mixed multi-class queueing networks can be used to analyze the performance of variants of CONWIP (such as m-CONWIP) that use a control loop for one or more routings in the production system. Literature on mixed queueing network is however scarce. The most important reason for this lack of literature is that the network is very difficult to analyze analytically. Most of the work on performance evaluation is therefore based on approximate analysis. Several approximation algorithms for mixed queueing networks are available in the literature, see e.g., Avi-Itzak and Heyman (1973), Baynat and Dallery (1996) and Buitenhek et al. (2000). Our research shows that these approximate algorithms are not precise enough to show the workload balancing effect we found in Chapter 2 for the m-CONWIP system, so it is of interest to refine these methods. Performance evaluation of more complicated pull structures such as POLCA is probably the most challenging direction for further research.

Although we showed in Chapter 2 that unit-based release policies can reduce the total throughput time in MTO production system by balancing the workload, the magnitude of the effect is rather small. In the second part of the thesis we therefore focused on a stronger instrument to control queues in MTO production systems: order acceptance.

In Chapter 3 we proposed a queueing approach for studying the performance of simple order acceptance policies for an MTO production system in which orders arrive and are serviced in batches by a single production process. We modeled the production system as a batch arrival batch service (bulk) queue with restricted accessibility. Our main contribution in this chapter is the development of a simple, numerically stable, and efficient algorithmic method that allows the performance evaluation of a general class of queueing systems that covers many bulk queueing systems with restricted accessibility as special cases. By means of numerical experiments we illustrated how our method can be used to compute relevant performance measures, such as the average time of orders in the system (i.e., throughput time), moments of the number of accepted orders and rejection probabilities for arriving orders.

A limitation of the stylized production situation we considered in Chapter 3 is that two important characteristics of many MTO systems are not included: setup times and due-dates. In Chapters 4 and 5 we therefore extended the production situation of Chapter 3 as follows. We considered a production system that produces different items on a single machine. Customer orders drive the production and belong to product families, and have family-dependent lead time, size, and profit margin. When production changes from one family to another a setup time is incurred. Orders are to be delivered on-date to customers and orders may be rejected if these orders cause late deliveries. This production situation is referred to in the literature as the Customized Stochastic Lot Scheduling Problem (CSLSP) with strict lead times. For this production situation it is critical to selectively accept and schedule customer orders, so that neither manufacturing capacity gets wasted on setups nor high profit earning orders are turned down because low profit earning orders have been previously accepted.

In Chapter 4 we provided a Markov decision process (MDP) formulation for the CSLSP with strict lead times. In this MDP we restricted the set of scheduling actions to two actions, Combine and Spawn, which are most relevant for the planner of the production system: the Combine action controls the formation of runs of orders of the same family while the Spawn action controls the generation of new runs of product families. Given these two scheduling actions we showed that a threshold type of policy has near optimal performance. The threshold policy restricts the generations of new runs by rejecting arriving orders when the length of the schedule exceeds a predefined threshold, thereby leaving room for later orders to combine with other runs in the schedule. Compared to the optimal policy of the MDP, the threshold policy is easy to understand and implement, for instance in a spreadsheet, hence has large practical value.

A limitation of the model of Chapter 4 is that we assumed that orders have standard lead times. That is, we assumed that the supplier commits itself to delivering all orders within the same lead time independent of the family of the order, so that as a consequence all lead times are equal to the shortest desirable lead time. While allowing for family-dependent lead times makes the planning situation more complicated, it may result in a significant gain in profit compared to using standard lead times as more orders can be accepted. However, it is not so clear from literature how order acceptance and scheduling policies can exploit lead time differences and whether the gain in profit outweighs the extra efforts of dealing with additional planning complexity.

In Chapter 5 we therefore explored the effect of quoting family-dependent lead times on the long run average reward for the CSLSP. By extending the MDP model of Chapter 4 such that it allows for family-depedent lead times, we showed that under the optimal policy an increase in lead time of one order family never has a detrimental effect on the performance; in fact in many cases significant extra rewards can be obtained. Also the threshold policy defined above works well in the majority of the cases, but not always. In fact, in some instances the performance actually degrades under the threshold policy when the lead time increases, contrary to what we see for the optimal policy. Thus, the threshold policy is not entirely robust. To repair this problem we proposed to also use thresholds for the Combine action. Interestingly, our results show that the best such improved threshold policy structurally performs within 1% of the optimal policy, and is robust against asymmetric lead times.

Although variations of the CSLSP with strict lead times have been investigated previously, the analysis by means of MDPs has not been addressed before in the literature. As a first attempt to gain insight into good acceptance and scheduling policies using an MDP approach, we restricted the set of scheduling actions available to planner of production system to two basic actions. In the original CSLSP, however, any action leading to a permutation of the schedule in which orders are produced in time is an admissible action. An interesting direction for further research is to refine and extend the set of actions such that the MDP model closer represents the original CSLSP. In Section 4.6 we proposed one such action (i.e., the 'Change Sequence' action) that could be of interest to consider for the planner.

A next suggestion for further research is to develop an analytic model to determine good parameters for threshold policy. In Chapter 4 and 5 we used the optimal policy of the MDP to determine good threshold parameters. A major drawback of this approach is that for realistic problem instances it is impossible to find the optimal policy of the MDP, at least within reasonable time and resources. Hence, we need other methods to efficiently find the best values for the thresholds.

To conclude, the starting point of our research was that order acceptance and release policies that are easy to understand and thereby easy to implement in practice can help MTO companies to improve their delivery performance. With respect to order release we showed that some unit-based pull systems can improve simultaneously total and shop floor throughput time performance while previous research could only show this for more complicated load-based systems. With respect to order acceptance we showed that simple threshold type of policies have near optimal performance. Most research, however, focuses on complicated policies to improve performance. We showed that relatively simple order acceptance and release policies can already lead to significant performance improvements.

Bibliography

Aalto, S. 2000. Optimal control of batch service queues with finite service capacity and linear holding costs. *Mathematical Methods of Operations Research* **51**(2) 263–285.

Asmussen, S. 2003. *Applied Probability and Queues*. Springer-Verlag, New York.

Avi-Itzak, B., D. P. Heyman. 1973. Approximate queuing models for multiprogramming computer systems. *Operations Research* **21**(6) 1212–1230.

Bagchi, T. P., J. G. C. Templeton. 1973. A note on the $M^X/G^Y/1, K$ bulk queueing system. *Journal of Applied Probability* **10**(4) 901–906.

Baker, K. R. 1984. Sequencing rules and due-date assignments in a job-shop. *Management Science* **30**(9) 1093–1104.

Baker, K. R., J. W. M. Bertrand. 1981. An investigation of due-date assignment rules with constrained tightness. *Journal of Operations Management* **1**(3) 109–120.

Baynat, B., Y. Dallery. 1996. A product-form approximation method for general closed queueing networks with several classes of customers. *Performance Evaluation* **24**(3) 165–188.

Bekker, R. 2004. Finite buffer queues with workload-dependent service and arrival rates. Ph.D. thesis, Eindhoven University of Technology, The Netherlands.

Bekker, R., S. C. Borst, O. J. Boxma, O. Kella. 2004. Queues with workload-dependent arrival and service rates. *Queueing Systems* **46**(3–4) 537–556.

Bertrand, J. W. M., J. C. Wortmann, J. Wijngaard. 1990. *Production Control: A Structural and Design Oriented Approach*. Elsevier, Amsterdam.

Blackstone Jr., J. H., D. T. Phillips, G. L. Hogg. 1982. A state-of-the-art survey of dispatching rules for manufacturing job shop operations. *International Journal of Production Research* **20**(1) 27–45.

Boxma, O. J., H. Levy, J. A. Weststrate. 1991. Efficient visit frequencies for polling tables: Minimization of waiting cost. *Queueing Systems* **9**(1–2) 133–162.

Bradt, L. J. 1983. The automated factory: Myth or reality? *Engineering: Cornell Quartely* **17**(3) 26–31.

Breithaupt, J.-W., M. J. Land, P. Nyhuis. 2002. The workload control concept: theory and practical extensions of Load Oriented Order Release. *Production Planning and Control* **13**(7) 625–638.

Buitenhek, R., G.-J. Van Houtum, W. H. M. Zijm. 2000. AMVA-based solution procedures for open queueing networks with population constraints. *Annals of Operations Research* **93**(1–4) 15–40.

Burke, P. J. 1975. Delays in single-server queues with batch inputs. *Operations Research* **23**(4) 830–833.

Buzacott, J. A., J. G. Shanthikumar. 1993. *Stochastic Models of Manufacturing Systems*. Prentice Hall, Englewood Cliffs, New Jersey.

Carr, S., I. Duenyas. 2000. Optimal admission control and sequencing in a make-to-stock/make-to-order production system. *Operations Research* **48**(5) 709–720.

Çelik, S., C. Maglaras. 2008. Dynamic pricing and lead-time quotation for a multiclass make-to-order queue. *Management Science* **54**(6) 1132–1146.

Çinlar, E. 1975. *Introduction to Stochastic Processes*. Prentice-Hall, Englewood Cliffs, New Jersey.

Chang, S. H., D. W. Choi, T.-S. Kim. 2004. Performance analysis of a finite-buffer bulk-arrival bulk-service queue with variable server capacity. *Stochastic Analysis and Applications* **22**(5) 1151–1173.

Chaudry, M., J. G. C. Templeton. 1983. *A First Course in Bulk Queues*. John Wiley & Sons, New York.

Cheng, T. C. E., M. C. Gupta. 1989. Survey of scheduling research involving due date determination decisions. *European Journal of Operational Research* **38**(2) 156–166.

Cohen, J. W. 1969. Single server queue with restricted accessibility. *Journal of Engineering Mathematics* **3**(4) 256–284.

Courtois, P. J., J. Georges. 1971. On a single server finite queuing model with state-dependent arrival and service processes. *Operations Research* **19**(2) 424–435.

Deb, R. K. 1978. Optimal dispatching of a finite capacity shuttle. *Management Science* **24**(13) 1362–1372.

Deb, R. K., R. F. Serfozo. 1973. Optimal control of batch service queues. *Advances in Applied Probability* **5**(2) 340–361.

Dshalalow, J. H. 1997. Queueing systems with state dependent parameters. J.H. Dshalalow, ed., *Frontiers in Queueing: Models and Applications in Science and Engineering*. CRC Press, Boca Raton, Florida, 61–116.

Dudin, A. N., A. A. Shaban, V. I. Klimenok. 2005. Analysis of a queue in the $BMAP/G/1/N$ system. *International Journal of Simulation: Systems, Science and Technology* **6**(1–2) 13–23.

Ebben, M. J. R., E. W. Hans, F. M. Olde Weghuis. 2005. Workload based order acceptance in job shop environments. *OR Spectrum* **27**(1) 107–122.

Fernandes, N. O., S. do Carmo-Silva. 2006. Generic POLCA – A production and materials flow control mechanism for quick response manufacturing. *International Journal of Production Economics* **104**(1) 74–84.

Fowler, J. W., G. L. Hogg, D. T. Phillips. 1992. Control of multiproduct bulk service diffusion/oxidation processes. *IIE Transactions* **24**(4) 84–96.

Framinan, J. M., P. L. Gonzalez, R. Ruiz-Usano. 2003. The CONWIP production control system: review and research issues. *Production Planning and Control* **14**(3) 255–265.

Fransoo, J. C., T. Wäfler, J. R. Wilson, eds. 2011. *Behavioral Operations in Planning and Scheduling*. Springer, Heidelberg.

Gaalman, G. J. C., M. Perona. 2002. Workload control in job shops: an introduction to the special issue. *Production Planning and Control* **13**(7) 565–567.

Garey, M. R., D. S. Johnson. 1979. *Computers and Intractability: A Guide to the Theory of NP-completeness*. Freeman, New York.

Gaury, E. G. A. 2000. Designing pull production control systems: customization and robustness. Ph.D. thesis, University of Tilburg, The Netherlands.

Germs, R., J. Riezebos. 2010. Workload balancing capability of pull systems in MTO production. *International Journal of Production Research* **48**(8) 2345–2360.

Germs, R., N. D. Van Foreest. 2010. Loss probabilities for the $M^X/G^Y/1/K+B$ bulk queue. *Probability in the Engineering and Informational Sciences* **24**(4) 457–471.

Germs, R., N. D. Van Foreest. 2011a. Admission policies for the customized stochastic lot scheduling problem with strict due-dates. *European Journal of Operational Research* **213**(2) 375–383.

Germs, R., N. D. Van Foreest. 2011b. Analysis of finite-buffer state-dependent bulk queues. Under revision.

Germs, R., N. D. Van Foreest. 2011c. Order acceptance and scheduling policies for a make-to-order environment with setup times and family-dependent lead times. Submitted.

Gupta, U. C., T. S. S. Srinivasa Rao. 1998. On the analysis of single server finite queue with state dependent arrival and service processes: $M(n)/G(n)/1/K$. *OR Spectrum* **20**(2) 83–89.

Gutiérrez-Jarpa, G., G. Desaulniers, G. Laporte, V. Marianov. 2010. A branch-and-price algorithm for the vehicle routing problem with deliveries, selective pickups and time windows. *European Journal of Operational Research* **206**(2) 341–349.

Haijema, R., J. Van der Wal, N. M. Van Dijk. 2007. Blood platelet production: optimization by dynamic programming and simulation. *Computers and Operations Research* **34**(3) 760–779.

Haijema, R., N. M. Van Dijk, J. Van der Wal, C. S. Sibinga. 2009. Blood platelet production with breaks: optimization by sdp and simulation. *International Journal of Production Economics* **121**(2) 464 – 473.

Haupt, R. 1989. A survey of priority rule-based scheduling. *OR Spectrum* **11**(1) 3–16.

Hendry, L., M. Land, M. Stevenson, G. Gaalman. 2008. Investigating implementation issues for workload control (wlc): A comparative case study analysis. *International Journal of Production Economics* **112**(1) 452–469.

Hendry, L. C., B. G. Kingsman, P. Cheung. 1998. The effect of workload control (WLC) on performance in make-to-order companies. *Journal of Operations Management* **16**(1) 63–75.

Hochbaum, D. S., D. Landy. 1997. Scheduling semiconductor burn-in operations to minimize total flowtime. *Operations Research* **45**(6) 874–885.

Hodes, B., B. Schoonhoven, R. Swart. 1992. On line planning van ovens. Tech. rep., University of Twente, Enschede, The Netherlands.

Hopp, W. J., M. L. Spearman. 2004. To pull or not to pull: What is the question? *Manufacturing & Service Operations Management* **6**(2) 133–148.

Hopp, W.J., M.L. Spearman. 2008. *Factory Physics*. McGraw-Hill/Irwin, New York.

Huang, S., M. Lu, G. Wan. 2011. Integrated order selection and production scheduling under MTO strategy. *International Journal of Production Research* **49**(13) 4085–4101.

Hyer, N. L., U. Wemmerlöv. 2002. *Reorganizing the Factory: Competing Through Cellular Manufacturing*. Productivity Press, Portland, OR.

Janert, P.K. 2010. *Gnuplot in Action: Understanding Data with Graphs*. Manning Publications, Greenwich, CT.

Kanet, J. J. 1988. Load-limited order release in job shop scheduling systems. *Journal of Operations Management* **7**(3) 44–58.

Kim, E., M.P. Van Oyen. 2000. Finite-capacity multi-class production scheduling with setup times. *IIE Transactions* **32**(9) 807–818.

Kleinrock, L., H. Levy. 1988. The analysis of random polling systems. *Journal of Operations Research* **36**(5) 716–732.

Land, M. J. 2004. Workload control in job shops, grasping the tap. Ph.D. thesis, University of Groningen, The Netherlands.

Land, M. J., G. J. C. Gaalman. 1996. Workload control concepts in job shops: A critical assesment. *International Journal of Production Economics* **46/47** 535–548.

Land, M. J., G. J. C. Gaalman. 1998. The performance of workload control concepts in job shops: Improving the release method. *International Journal of Production Economics* **56/57** 347–364.

Little, J. D. C. 1961. A proof for the queuing formula: $l = \lambda w$. *Operations Research* **9**(3) 383–387.

Liu, Z., P. Nain, D. Towsley. 1992. On optimal polling policies. *Queueing Systems* **11**(1–2) 59–83.

MacGregor Smith, J., F. R. B. Cruz. 2005. The buffer allocation problem for general finite buffer queueing networks. *IIE Transactions* **37**(4) 343–365.

Markovitz, D. M., L. M. Wein. 2001. Heavy traffic analysis of dynamic cyclic policies: A unified treatment of the single machine scheduling problem. *Operations Research* **49**(2) 246–270.

Markowitz, D. M., M. I. Reiman, L. M. Wein. 2000. The stochastic economic lot scheduling problem: Heavy traffic analysis of dynamic cyclic policies. *Operations Research* **48**(1) 136–154.

Medhi, J. 2003. *Stochastic Models in Queueing Theory*. Academic Press, San Diego.

Melnyk, S. A., G. L. Ragatz. 1988. Order review/release and its impact on the shop floor. *Production and Inventory Management Journal* **29**(2) 13–17.

Neuts, M. F. 1977. *Algorithmic Methods in Probability Theory*. North-Holland, Amsterdam.

Nobel, R. D. 1989. Practical approximations for finite-buffer queueing models with batch arrivals. *European Journal of Operational Research* **38**(1) 44–55.

Oğuz, C., F. Sibel Salman, Z. Bilgintürk Yalçin. 2010. Order acceptance and scheduling decisions in make-to-order systems. *International Journal of Production Economics* **125**(1) 200–211.

Öztürk, A., S. Kayaligil, N. E. Özdemirel. 2006. Manufacturing lead time estimation using data mining. *European Journal of Operational Research* **173**(2) 683–700.

Pinedo, M. L. 2008. *Scheduling: Theory, Algorithms, and Systems*. Springer, New York.

Puterman, M. L. 1994. *Markov Decision Processes, Discrete Stochastic Dynamic Programming*. J. Wiley & Sons, New York.

Ramasesh, R. 1990. Dynamic job shop scheduling: A survey of simulation research. *Omega* **18**(1) 43–57.

Reiman, M. I., L. M. Wein. 1998. Dynamic scheduling of a two-class queue with setups. *Operations Research* **46**(4) 532–547.

Riezebos, J. 2010. Design of POLCA material control systems. *International Journal of Production Research* **48**(5) 1455–1477.

Righter, R., J. G. Shantikumar. 1998. Multiclass production systems with setup times. *Operations Research* **12**(3) 146–153.

Schellhaas, H. 1983. Computation of the state dependent probabilities in $M/G/1$ queues with state dependent input and state dependent service. *OR Spectrum* **5**(4) 223–228.

Schmidt, E., M. Dada, J. Ward, D. Adams. 2001. Using cyclic planning to manage capacity at Alcoa. *Interfaces* **31**(3) 16–27.

Slomp, J., J. A. C. Bokhorst, R. Germs. 2009. A lean production control system for high-variety/low-volume environments: A case study implementation. *Production Planning & Control* **20**(7) 586–595.

Slotnick, S. A. 2011. Order acceptance and scheduling: A taxonomy and review. *European Journal of Operational Research* **212**(1) 1–11.

Slotnick, S. A., M. J. Sobel. 2005. Manufacturing lead-time rules: Customer retention versus tardiness costs. *European Journal of Operational Research* **163**(3) 825–856.

Stevenson, M., L. C. Hendry, B. G. Kingsman. 2005. A review of production planning and control: The applicability of key concepts to the make-to-order industry. *International Journal of Production Research* **43**(1) 869–898.

Strijbosch, L. W. G., R. M. J. Heuts, M. L. J. Luijten. 2002. Cyclical packaging planning at a pharmaceutical company. *International Journal of Operations & Production Management* **22**(5) 549–564.

Sugimori, Y., K. Kusunoki, F. Cho, S. Uchikawa. 1977. Toyota production system and kanban system: materialization of just-in-time and respect-for-human system. *International Journal of Production Research* **15**(6) 553–564.

Suri, R. 1998. *Quick Response Manufacturing: A Companywide Approach to Reducing Leadtimes*. Productivity Press, Portland, OR.

Suri, R. 2010. *It's About Time: The Competitive Advantage of Quick Response Manufacturing*. Productivity Press, New York.

Suri, R., A. Krishnamurthy. 2003. How to plan and implement polca: A material control system for high-variety or custom-engineered products. Tech. rep., Center for Quick Response Manufacturing, (www.qrmcenter.org), University of Wisconsin-Madison, WI.

Takagi, H. 1990. Queueing analysis of polling models: an update. H. Takagi, ed., *Stochastic Analysis of Computer and Communication Systems*. Elsevier, 267–318.

Takagi, H. 1991. Analysis of finite-capacity polling systems. *Advances in Applied Probability* **23**(2) 373–387.

Takagi, H. 1993. *Queueing Analysis: Finite Systems*. North–Holland, Amsterdam.

Ten Kate, H. 1995. Order acceptance and production control. Ph.D. thesis, University of Groningen, The Netherlands.

Thürer, M. 2011. Workload control: An assessment. Ph.D. thesis, University of Coimbra, Portugal.

Tijms, H. C. 2003. *A First Course in Stochastic Models*. John Wiley & Sons, Chichester.

Tijms, H. C., M. H. Van Hoorn. 1981. Algorithms for the state probabilities and waiting times in single server queueing systems with random and quasirandom input and phase-type service times. *OR Spectrum* **2**(3) 145–152.

Uzsoy, R., C. Y. Lee, L. A. Martin-Vega. 1994. A review of production planning and scheduling models in the semiconductor industry, part ii: Shop-floor control. *IIE Transactions* **26**(5) 44–55.

Van der Zee, D. J., A. Van Harten, P. Schuur. 2001. On-line scheduling of multi-server batch operations. *IIE Transactions* **33**(7) 569–586.

Van Foreest, N. D., J. Wijngaard, J. T. Van der Vaart. 2010. Scheduling and order acceptance for the customised stochastic lot scheduling problem. *International Journal of Production Research* **48**(12) 3561–3578.

Vandaele, N., I. Van Nieuwenhuyse, S. Cupers. 2003. Optimal grouping for a nuclear magnetic resonance scanner by means of an open queueing model. *European Journal of Operational Research* **151**(1) 181–192.

Vandaele, N., I. Van Nieuwenhuyse, D. Claerhout, R. Cremmery. 2008. Load-based POLCA: an integrated material control system for multiproduct, multimachine job shops. *Manufacturing & Service Operations Management* **10**(2) 181–197.

Weiss, H. J. 1979. The computation of optimal control limits for a queue with batch services. *Management Science* **25**(4) 320–328.

Wester, F. A. W., J. Wijngaard, W. H. M. Zijm. 1992. Order acceptance strategies in a production-to-order environment with setup times and due-dates. *International Journal of Production Research* **30**(6) 1313–1326.

Wijngaard, J. 2007. Models for production planning and control. H. Corsten, H. Missbauer, eds., *Productions- und Logistikmanagement*. Liber Amicorum Zäpfel, Vahlen, 195–207.

Winands, E. M. M., I. J. B. F. Adan, G. J. Van Houtum. 2011. The stochastic economic lot scheduling problem: A survey. *European Journal of Operational Research* **210**(1) 1–9.

Ziengs, N., J. Riezebos, R. Germs. 2011. Placement of effective work-in-progress limits in route-specific unit-based pull systems. *International Journal of Production Research* Article in press.

Samenvatting

Dit proefschrift richt zich op de analyse en het ontwerp van orderacceptatie- en ordervrijgaveregels voor het beheersen van wachttijden in klantordergestuurde (MTO) productiesystemen. Het beheersen van de tijd die orders moeten wachten op beschikbare productiecapaciteit is cruciaal voor het realiseren van korte en betrouwbare doorlooptijden, twee prestatie-indicatoren die van strategisch belang zijn voor veel MTO-bedrijven. Orderacceptatie en ordervrijgave zijn de twee belangrijkste productiebeheersingsmechanismen voor het beïnvloeden van de lengte van wachttijden. Om een beter begrip te krijgen van de afwegingen die MTO-bedrijven moeten maken in hun orderacceptatie- en ordervrijgavebeslissingen richt dit proefschrift zich in de modellering op de belangrijkste (zogenoemde eerste orde) kenmerken van MTO-systemen, zoals routing variabiliteit, beperkte productiecapaciteit, omsteltijden, strikte leveringsvoorwaarden en onzekerheid in het aankomstpatroon van orders.

De voornaamste onderzoeksdoelstellingen van dit proefschrift zijn: (i) het verkrijgen van beter inzicht in de onderliggende mechanismen van goede orderacceptatie en ordervrijgaveregels voor MTO-productieomgevingen, en (ii) het gebruiken van deze inzichten om eenvoudige orderacceptatie- en ordervrijgaveregels te ontwikkelen om wachttijden in MTO-productiesystemen te beheersen. Het proefschrift is als volgt ingedeeld: Hoofdstuk 2 richt zich op de ordervrijgavebeslissing, waarna de Hoofdstukken 3, 4 en 5 de orderacceptatiebeslissing behandelen.

Hoofdstuk 2 onderzoekt de doorlooptijdprestaties van drie ordervrijgaveregels die het aantal orders op de werkvloer beheersen door slechts gebruik te maken van een verzameling kaarten: CONWIP, m-CONWIP en POLCA. Door hun eenvoudige bediening worden deze zogenaamde unit-based pull-systemen op grote schaal geïmplementeerd in de praktijk. Echter, al het onderzoek naar de prestaties van deze pull-systemen richt zich op de doorlooptijd en aantallen orders op de werkvloer en niet op de prestatie-indicator die het meest relevant is voor MTO-bedrijven, namelijk de totale doorlooptijd van orders, dus inclusief de tijd die orders wachten op vrijgave naar de werkvloer. Eerder onderzoek toont ook aan dat de totale doorlooptijdprestatie van pull-systemen grotendeels afhangt van hun vermogen om een evenwichtige verdeling van de werklast tussen de werkplekken op de werkvloer te creren en dat in veel systemen dit vermogen ontbreekt. Dit hoofdstuk ontwikkelt een simulatiemodel om de doorlooptijdprestaties van CONWIP, POLCA en m-CONWIP te vergelijken. De simulaties tonen aan dat unit-based pull-systemen de werklastverdeling op de werkvloer kunnen verbeteren en kunnen zorgen voor een vermindering van de gemiddelde totale doorlooptijd. CONWIP heeft geen werklastbalanceringsmogelijkheden en de resultaten laten dan ook zien dat een beperking van de werklast in het CONWIP gecontroleerde MTO-productiesysteem de gemiddelde totale doorlooptijd van orders vergroot. De overlappende lussen in het POLCA-systeem zorgen voor beperkte werklastbalancering in vergelijking met CONWIP. Echter, doordat de POLCA-lussen onvoldoende onbalans in werklast detecteren en signaleren heeft POLCA een mindere doorlooptijdprestatie dan m-CONWIP, het systeem met de beste werklastbalanceringsmogelijkheden.

Hoewel Hoofdstuk 2 laat zien dat de regels voor unit-based vrijgave de totale doorlooptijd in een MTO-productiesysteem kunnen verminderen door het balanceren van de werklast, is de omvang van het effect vrij klein. Het tweede deel van het proefschrift richt zich daarom op een krachtiger instrument om wachttijden in MTO productiesystemen te beheersen: orderacceptatie.

Hoofdstuk 3 introduceert een wachtrijmethode voor het bestuderen van de prestatie van eenvoudige orderacceptatieregels voor een MTO-productiesysteem waarin orders aankomen en worden bediend in batches door één machine. Het productiesysteem is gemodelleerd als een batch-aankomst batch-service (bulk) wachtrijsysteem met beperkte toegankelijkheid. De belangrijkste bijdrage in dit hoofdstuk is de ontwikkeling van een eenvoudige, numeriek stabiele en efficinte algoritmische methode die de prestatie-evaluatie mogelijk maakt van een orde van wachtrijsystemen die vele bulk wachtrijsystemen met een beperkte toegankelijkheid als speciale gevallen omvat. Door middel van numerieke experimenten wordt aangetoond hoe de methode kan worden gebruikt om relevante prestatie-indicatoren te berekenen, zoals de gemiddelde doorlooptijd van orders, momenten van het aantal geaccepteerde orders en de kans op het afwijzen van orders.

Een beperking van de gestileerde productiesituatie uit Hoofdstuk 3 is dat twee belangrijke eigenschappen, die in veel MTO-systemen voorkomen, niet worden meegenomen: omsteltijden en strikte leveringstijden. Hoofdstuk 4 en 5 breiden daarom de productiesituatie van Hoofdstuk 3 als volgt uit. Er wordt in deze hoofdstukken gekeken naar een productiesysteem dat verschillende producten op een enkele machine produceert. Klantorders sturen het productiesysteem en behoren tot verschillende productfamilies. Verder kenmerken klantorders zich door familieafhankelijke leveringstijden, productietijd en winstmarge. Wanneer de productie verandert van de ene familie naar de andere dan kost dit omsteltijd. Orders dienen op tijd te worden geleverd aan klanten en orders kunnen geweigerd worden als ze leiden tot laattijdige leveringen. Deze productiesituatie staat in de literatuur bekend als het Customized Stochastic Lot Scheduling Problem (CSLSP) met strikte leveringstijden. Voor deze productiesituatie is het essentieel om orders selectief te accepteren en in te plannen op de machine, zodat noch productiecapaciteit wordt verspild aan omsteltijd, noch orders met een hoge winstmarge worden afgewezen, omdat orders met een lagere winstmarge in een eerder stadium eerder zijn geaccepteerd.

In Hoofdstuk 4 wordt een Markov beslissingsproces (MDP) formulering gegeven voor het CSLSP met strikte leveringstijden. In dit MDP is de verzameling van scheduling-acties beperkt tot twee acties, Combine en Spawn, die het meest relevant zijn voor de planner van het productiesysteem: de Combine-actie regelt de vorming van (productie)reeksen van orders van dezelfde familie, terwijl de Spawn-actie de generatie van nieuwe reeksen van productfamilies regelt. Gegeven deze twee scheduling-acties laat dit hoofdstuk zien dat een heuristische beslisregel, die op basis van een drempelwaarde orders accepteert, vrijwel een optimale prestatie levert. Deze drempelwaarderegel beperkt de generatie van nieuwe reeksen door het afwijzen van nieuwe orders wanneer de lengte van het schema groter is dan een vooraf bepaalde drempelwaarde. Hierdoor kunnen orders die later arriveren gecombineerd worden met andere reeksen in het rooster. Vergeleken met de optimale beslisregel voor het MDP is de drempelwaarderegel eenvoudig te begrijpen en te implementeren, bijvoorbeeld in een spreadsheet, en heeft het dus grote praktische waarde.

Een beperking van het model van hoofdstuk 4 is dat er aangenomen wordt dat orders standaardlevertijden hebben. Deze aanname houdt in dat de leverancier zich verplicht tot het leveren van alle orders binnen dezelfde vooraf afgesproken doorlooptijd, onafhankelijk van de productfamilie van de order. Als gevolg hiervan moeten alle doorlooptijden gelijk zijn aan de kortste gewenste doorlooptijd. Ondanks dat familieafhankelijke leveringstijden het planningsprobleem ingewikkelder maken, zou dit kunnen resulteren in een significante winsttoename in vergelijking tot het gebruik van standaardlevertijden omdat meer orders kunnen worden geaccepteerd. De literatuur geeft echter geen duidelijkheid hoe orderacceptatie- en schedulingsregels rekening kunnen houden met verschillen in de doorlooptijd en of de winsttoename opweegt tegen de extra inspanningen die de extra planningscomplexiteit met zich meebrengt.

Hoofdstuk 5 onderzoekt daarom het effect van het gebruik van familieafhankelijke doorlooptijden op de lange termijn op de gemiddelde winst voor het CSLSP. Door het MDP-model uit Hoofdstuk 4 uit te breiden naar familieafhankelijke doorlooptijden, laat Hoofdstuk 5 zien dat onder de optimale beslisregel een toename van de doorlooptijd van een productfamilie nooit een nadelig effect op de prestaties heeft; in veel gevallen kan zelfs een aanzienlijke winst worden behaald. Ook de drempelwaarderegel zoals hierboven beschreven werkt goed in de meerderheid van de onderzochte gevallen, maar niet altijd. In bepaalde gevallen zorgt de drempelwaarderegel voor een afname van de prestatie wanneer de doorlooptijd toeneemt, in tegenstelling tot wat we zien voor de optimale beslisregel. Dit laat zien dat de drempelwaarderegel niet helemaal robuust is. Om dit probleem te herstellen wordt een drempelwaarderegel voor de Combine-actie voorgesteld. Een interessante uitkomst is dat de resultaten laten zien dat de bestverbeterde drempelwaarderegel structureel binnen 1% van de optimale beslisregel presteert, en dus robuust is tegen asymmetrische doorlooptijden.

Dankwoord

De afgelopen vier jaar van mijn promotie zijn in een recordtempo voorbij gevlogen. Behalve dat de tijd gevoelsmatig sneller gaat naarmate je ouder wordt, is in mijn beleving de belangrijkste factor het plezier geweest waarmee ik vier jaar lang heb kunnen werken aan mijn proefschrift. Veel collega's en vrienden hebben daar aan bijgedragen en een aantal van hen wil ik hier in het bijzonder bedanken.

Allereerst wil ik mijn promotor Jannes Slomp en mijn copromotoren Jan Riezebos en Nicky van Foreest bedanken voor hun vertrouwen in mij en voor de vrijheid die ik heb gekregen om mijzelf in zeer verschillende richtingen te kunnen ontwikkelen. Jannes wil ik bedanken voor het benadrukken dat toepasbaarheid voorop moet staan op de momenten waarop ik door wilde slaan in het oplossen van geavanceerde, maar weinig relevante, wiskundige modellen. Jan bedank ik voor het verschaffen van gedetailleerd commentaar op mijn stukken, voor het waarborgen van de lijn van mijn proefschrift en voor de hulp na mijn promotie bij het solliciteren. En dan Nicky: bedankt voor de grote inhoudelijke bijdrage die jij hebt geleverd aan mijn proefschrift, dit ondanks het feit dat je pas in het tweede jaar van mijn promotie formeel mijn begeleider bent geworden. Onder jouw begeleiding heb ik enorme vooruitgang op het methodologische vlak kunnen boeken. Jouw enthousiasme voor nieuwe problemen is bovendien een grote stimulans voor mij geweest om mijn carrire in de wetenschap voort te zetten. Ik hoop dat wij in de toekomst nog aan de vele interessante problemen, waar we tijdens mijn promotie aan zijn begonnen, maar die helaas niet in de lijn van dit proefschrift pasten, kunnen samenwerken.

Naast mijn begeleiders wil ik Jacob Wijngaard en Hans Nieuwenhuis bedanken voor de interesse die zij bij mij hebben gewekt voor het bestuderen en toepassen van stochastische modellen.

Verder zijn er veel collega's en vrienden die inhoudelijk niet of zeer weinig aan de totstandkoming van dit proefschrift hebben bijgedragen, maar die op het persoonlijke vlak zeer belangrijk voor mij zijn geweest. De belangrijkste hierin is zonder twijfel Justin, mijn kamergenoot gedurende de vier jaren van mijn promotie. Als ik alle tijd die wij aan ongein hebben besteed effectief in mijn proefschrift had kunnen stoppen, dan had ik waarschijnlijk nooit weekenden of avonden hoeven doorwerken en had ik mijn proefschrift in een eerder stadium kunnen afronden. Daar staat tegenover dat ik altijd met ontzettend veel plezier naar mijn werk ben gegaan en als ik alles nog een keer over mocht doen, dan zou ik jou zeer zeker weer als kamergenoot wensen. Uit de categorie collega's, vrienden, sport- en koffiemaatjes wil ik Onur en Kristian bedanken voor de prettige afleiding waar jullie tijdens mijn promotie voor hebben gezorgd en ik hoop in de toekomst zowel op wetenschappelijk als op persoonlijk vlak nog mooie momenten met jullie te beleven.

Tot slot wil ik mijn familie en mijn vriendin Lutiena bedanken voor hun ondersteuning tijdens mijn promotie. Met name voor Lutiena was het niet gemakkelijk om mij vooral 's avonds en in het weekend aan mijn onderzoek te laten werken. Desondanks was ze altijd vol begrip en vaak een extra stimulans voor het afronden van de puntjes waar ik niet zo gemotiveerd voor was. Ook in deze onzekere periode waarin ik binnen en buiten Nederland solliciteer op een baan in de academische wereld is Lutiena's steun onvoorwaardelijk. Dit terwijl het voor haar minstens zo lastig gaat worden om in een nieuwe omgeving een nieuw leven op te bouwen. Lutiena, ik hou van jou en als dank voor je steun heb ik dit proefschrift aan jou opgedragen.