INTEGRATING COMBINATORIAL SCHEDULING WITH INVENTORY MANAGEMENT AND QUEUEING THEORY

by

Daria Terekhov

A thesis submitted in conformity with the requirements
for the degree of Doctor of Philosophy
Graduate Department of Mechanical and Industrial Engineering
University of Toronto

Abstract

Integrating combinatorial scheduling with inventory management and queueing theory

Daria Terekhov
Doctor of Philosophy
Graduate Department of Mechanical and Industrial Engineering
University of Toronto
2013

The central thesis of this dissertation is that by combining classical scheduling methodologies with those of inventory management and queueing theory we can better model, understand and solve complex real-world scheduling problems.

In part II of this dissertation, we provide models of a realistic supply chain scheduling problem that capture both its combinatorial nature and its dependence on inventory availability. We present an extensive empirical evaluation of how well implementations of these models in commercially available software solve the problem. We are therefore able to address, within a specific problem, the need for scheduling to take into account related decision-making processes.

In order to simultaneously deal with combinatorial and dynamic properties of real scheduling problems, in part III we propose to integrate queueing theory and deterministic scheduling. Firstly, by reviewing the queueing theory literature that deals with dynamic resource allocation and sequencing and outlining numerous future work directions, we build a strong foundation for the investigation of the integration of queueing theory and scheduling. Subsequently, we demonstrate that integration can take place on three levels: conceptual, theoretical and algorithmic. At the conceptual level, we combine concepts, ideas and problem settings from the two areas, showing that such combinations provide insights into the trade-off between long-run and short-run objectives. Next, we show that theoretical integration of queueing and scheduling can lead to long-run performance guarantees for scheduling algorithms that have previously been proved only for queueing policies. In particular, we are the first to prove, in two flow shop environments, the stability of a scheduling method that is based on the traditional scheduling

literature and utilizes processing time information to make sequencing decisions. Finally, to address the algorithmic level of integration, we present, in an extensive future work chapter, one general approach for creating hybrid queueing/scheduling algorithms. To our knowledge, this dissertation is the first work that builds a framework for integrating queueing theory and scheduling.

Motivated by characteristics of real problems, this dissertation takes a step toward extending scheduling research beyond traditional assumptions and addressing more realistic scheduling problems.

Acknowledgements

This research has been supported by the Natural Sciences and Engineering Research Council of Canada, the Canadian Foundation for Innovation, the Ontario Research Fund, the Ontario Ministry for Research and Innovation, the Ireland Industrial Development Agency, Alcatel-Lucent, Microway Inc., IBM ILOG, SGS Doctoral Completion Award, and the Department of Mechanical and Industrial Engineering.

There are many people who have helped me throughout my Ph.D. program - to properly acknowledge all of you, I would likely need to write a document as long as this dissertation, and I am not sure that the University of Toronto will approve of that. The following is therefore just an abstract.

Firstly, I would like to thank my supervisor, Professor J. Christopher Beck, for being the optimal supervisor for me. You have been patient and supportive throughout my Ph.D. program, and have taught me not just how to do research and present the results, but also about academia in general. Thank you also for giving me the opportunity to attend many conferences, for arranging the summer internship at Alcatel-Lucent Bell Labs, and for helping me to establish connections.

Next, I would like to thank Professor Douglas G. Down, for the excellent support and guidance in queueing theory. I greatly appreciate all of the time you have invested in explaining (and re-explaining!) the material that was difficult for me. Thank you very much for your help!

I would also like to thank members of my thesis committee, Professor C.-G. Lee, Professor B. Balciouglu, Professor M. Trick and Professor T. C. Y. Chan, for comments that have helped improve this dissertation.

Thank you also to Mustafa K. Doğru and Ulaş Özen for your guidance during my internship at Alcatel-Lucent Bell Labs and during working on the assembly scheduling paper, for providing such an interesting scheduling problem for me to work on, and for all your advice.

I would like to thank Tony T. Tran, for doing the preliminary work for Chapter 7 (i.e., initial implementations of the polling system and dynamic flow shop, initial experiments), and for having the insight that *makespan* should perform better than *completionTime* in the polling system. In addition, I would like to thank you for the very helpful discussions and explanations.

I would like to thank Maliheh Aramon Bajestani, for countless discussions (which, being very loud, have likely distracted many people from their work) of everything grad school-related, for pushing me to finish (including reminding me of due dates for my thesis chapters!) and for being a great friend.

Thank you to Tiago S. Vaquero, for useful advice and for providing a lot of technical support for my work, ranging from installing Eclipse and LED (which I didn't even know existed before you showed them to me) to helping me figure out how to include Russian words in these acknowledgements.

I would also like to thank all the friends that have supported me during the years of my program, including my friends from the tennis club, my suitemates at Graduate House, and my friends and colleagues at TIDEL.

Thank you to Gabriel Watson for your support, sweetness and thoughtfulness over the last year of this program. Thank you also for cooking dinner and washing dishes when I was busy, for the numerous thesis-related jokes, for providing me with a mouse (which really made my work faster), for amazing quotes such as "A scheduler's life-long challenge is that of beating time into his/her own rules, only to find out later it was time that ruled them to dance in synch with its beat", and for all the fun times that allowed me to take breaks from thesis-writing.

Спасибо всем родственникам за поддержку! Баба Люба и Деда Вова, я выучилась на доктора!

Thank you to my parents, for helping me to achieve my goal, for advising me and supporting me through the ups-and-downs of graduate school, for driving me to and from the train station, for providing bundles of supplies, for helping me move in and out of numerous residences and apartments, for assisting with optimal scheduling, for encouraging me to do my best, and for helping me realize, that, after all, the back-room/front-room problem is not really the most important problem in the world.

Contents

List of Tables

List of Figures

Part I

Introduction

Chapter 1

Introduction

The central thesis of this dissertation is that by combining classical scheduling methodologies with those of inventory management and queueing theory we can better model, understand and solve complex real-world scheduling problems. Real-world scheduling problems are combinatorial in nature, dynamic and affected by many sources of uncertainty, and closely related to other decision-making processes of the environment they exist in. The classical scheduling literature has focused on solving isolated, static and deterministic versions of these problems; that is, problems that correspond to a snapshot of the environment at a particular point in time, in which all relevant task characteristics are known with certainty and in which the dependence on other decision-making processes is not considered. Motivated by characteristics of real problems, this dissertation takes a step toward extending scheduling research beyond these traditional assumptions.

Firstly, we investigate the premise that a scheduling problem rarely exists in isolation: it is usually part of an environment in which many other complex decision-making problems need to be solved. In manufacturing and supply chain applications, the production of items requires raw materials and component parts. Thus, when a production schedule is created, the availability of these components has to be taken into account. When decisions regarding the timing and quantity of component part replenishments are made, one needs to consider how and when the components are going to be used during production. In spite of being closely related, these scheduling and inventory decisions have mostly been addressed separately in the literature. In this dissertation, we study a supply chain scheduling problem with both types of decisions. By explicitly representing the dependence of scheduling on inventory replenishments, we are able to more accurately model the real problem that motivated our study, and to address, within a specific problem, the need for scheduling to take into account related decision-making processes.

Secondly, we focus on two areas that have developed very different techniques for dealing

with resource allocation and sequencing: queueing theory and deterministic scheduling. Of particular interest are dynamic scheduling problems, in which the set of jobs changes dynamically over time. The goal in such problems is to determine how the available resources are to be allocated among competing requests with the objective of optimizing the performance of the system. For example, in a manufacturing facility, incoming customer orders need to be processed on various machines. Classical scheduling research has mostly focused on devising effective methods for dealing with the complex combinatorics of such problems while assuming a static and deterministic environment. Thus, in the context of a manufacturing facility, a typical problem considered by the scheduling community would involve sequencing the production of a fixed set of products on the available machines. Queueing theory, on the contrary, generally assumes a relatively simple combinatorial structure, but models the stochastic and dynamic properties of systems. For instance, a queueing model is able to represent the arrivals of customer orders via stochastic processes, but would not be able to capture the detailed resource constraints of a scheduling model. We demonstrate that integrating concepts, ideas and methodologies from queueing theory and scheduling is beneficial for modelling, understanding and solving scheduling problems with both a combinatorial structure and dynamic properties.

The detailed motivations for our work are presented in Chapter 2. The contributions of this dissertation are given in the following section.

1.1 Summary of Contributions

The three main contributions of this dissertation are:

1. We make a step toward the development of a general framework for integrating scheduling and inventory decision-making by presenting novel models of an assembly environment in which scheduling decisions depend on the availability of component parts.

2. We are the first to demonstrate that combining problem settings and concepts from queueing theory and scheduling can lead to novel insights about scheduling in dynamic environments. Specifically, we obtain a new understanding of the trade-off between short-run and long-run objectives in dynamic scheduling.

3. We are the first to prove, in two environments, the stability of a scheduling method that is based on the traditional scheduling literature and utilizes processing time information to make sequencing decisions. Thus, we show that theoretical integration of queueing and scheduling can lead to long-run performance guarantees for scheduling algorithms that have previously been proved only for queueing policies.

In addition,

4. Using mixed-integer and constraint programming, we model and solve a scheduling problem with inventory constraints and an assembly structure which is based on a real supply chain problem. Our models are implemented using commercially available software and extensively empirically evaluated. As a result, our work provides a baseline on how well the problem can be solved without resorting to problem-specific algorithms.

5. We provide an overview of scheduling methods developed in queueing theory. By highlighting the differences and similarities between the ways that resource allocation and sequencing problems are addressed in the queueing theory and scheduling literatures, we establish a clear relationship between two fields that have developed mostly independently. We also describe numerous opportunities for integration of queueing and scheduling, which have the potential to lead to a better understanding of dynamic scheduling. Our review of past work and future directions builds a strong foundation for the investigation of the integration of queueing theory and scheduling.

6. It is shown that periodic scheduling methods can perform better than queueing-based dispatching rules for optimization of long-run performance. However, the choice of objective function for each sub-problem, which is a proxy measure for the long-run objective, may not always be obvious. Thus, our work suggests that, for general dynamic scheduling problems, it is important to investigate various sub-problem objective functions in order to develop an effective scheduling method.

7. We are the first to demonstrate the applicability of the fluid model methodology for proving stability of, and deriving new insights about, periodic scheduling policies based on processing times.

8. We propose and study a novel scheduling environment: a polling system with a gated, cyclic discipline and a server that is a two-machine flow shop. We prove stability of the *first-come, first-served* policy and of a periodic scheduling approach in this setting, and show that our proof extends to the case when the server is an M-machine flow shop or a d-stage flexible flow shop with M machines at each stage.

1.2 Overview of Dissertation

There are four parts in this dissertation.

Part I: Introduction The first part of this dissertation consists of the current introductory chapter, and Chapter 2, which presents the practical and theoretical motivations for our work.

Part II: Assembly Scheduling with Inventory Constraints Part II focuses on a scheduling problem with inventory constraints in a small supply chain consisting of two manufacturing facilities and a merge-in-transit facility, which is further referred to as the assembly scheduling problem. This part shows that a realistic scheduling problem can be addressed by combining ideas from deterministic scheduling and inventory management.

Chapter 3 provides a review of the literature on the underlying combinatorial scheduling problem as well as on inventory management. We also survey previous work in combinatorial scheduling with inventory constraints.

Chapter 4 presents three mixed-integer programming (MIP) models and a constraint programming (CP) model for the assembly scheduling problem with inventory constraints, and empirically evaluates their relative performance. Results show that when there are no components shared among the two manufacturers, the MIP model based on time-index variables is the best for proving optimality for problems with short processing times whereas the CP model tends to perform better than the others for problems with a large range of processing times. When shared components are added, the performance of all models deteriorates, with the time-indexed MIP providing the best results.

Chapter 5 outlines directions for future work related to the assembly scheduling problem, which include the development of better solution techniques, problem extensions and the creation of a general framework for integration of inventory management and scheduling.

Part III: Integrating Scheduling and Queueing Theory for Dynamic Scheduling Problems Part III demonstrates that combining concepts, ideas and methodologies from queueing theory and scheduling leads to a better understanding of realistic scheduling problems, which are both combinatorial and dynamic in nature. This part of the dissertation is based on the view that integration of the two areas can take place on three different levels: conceptual, theoretical and algorithmic.

Chapter 6 is a review of queueing theory models and methods that are relevant to scheduling in dynamic settings. This review highlights the similarities and differences between the methodologies developed in queueing and scheduling, and notes opportunities for combining the two areas. The review provides a foundation for the investigation of the integration of queueing and scheduling.

Chapter 7 demonstrates the benefits of combining queueing and scheduling on a conceptual level. It studies two dynamic flow shop environments: a novel polling system that is an exten-

sion of systems traditionally examined in queueing theory, and a dynamic two-machine flow shop, which is important in scheduling as well as queueing research. In both cases, the objective is to minimize the mean flow time of jobs (i.e., the time between the arrival of a job to the system and its completion time). Our experiments show that in the polling system, a scheduling method that optimizes the makespan at each decision point provides the best performance, while in the dynamic flow shop, an approach based on minimizing the mean flow time works better. We formally analyze these performance differences and show that they are a function of the specific structural properties of the scheduling problems. Our analysis shows that short-run and long-run objectives affect our two related flow-shop environments differently.

Chapter 8 shows that theoretical integration of queueing and scheduling can lead to long-run performance guarantees for scheduling algorithms that have previously been formulated only for queueing policies. Specifically, it analyzes stability of the two-machine flow shop and the polling system with a two-machine flow shop server that are presented in Chapter 7. In the dynamic flow shop, stability of a scheduling approach that periodically solves static deterministic sub-problems is shown using a sample path argument. In the polling system, stability of *first-come, first-served* and of a periodic scheduling method is proved using the fluid model methodology of Dai (1995). In the dynamic flow shop, we show that our sample path proof does not extend to the case with more than two machines; in the polling system, however, the fluid model proofs of stability extend to environments where the server is an M-machine flow shop or a d-stage flexible flow shop with M machines.

Chapter 9 continues the theoretical integration of queueing and scheduling, proposing fluid limit analysis as a tool for gaining insights about scheduling algorithm performance. The chapter focuses on the fluid limits of work arriving to and present at the second machine of a static two-machine flow shop. Analysis of these limits leads to: the identification of a key algorithm feature responsible for optimizing makespan; observations regarding the usefulness of fluid limits for predicting algorithm performance; and a further understanding of stability and instability of periodic scheduling methods that employ processing time information.

Chapter 10 outlines future work on the integration of queueing theory and scheduling, classifying it according to the three levels of integration (conceptual, theoretical and algorithmic). For the conceptual and theoretical levels, a summary of the relevant ideas from Chapter 6 is presented first, followed by future work arising from Chapters 7, 8 and 9 and then by additional ideas that have not been mentioned in the previous chapters. The last section of the chapter deals with the algorithmic level of integration, outlining a general framework for creating hybrid queueing/scheduling models, and discussing two examples of how it can be applied.

Part IV: Conclusion In Chapter 11, which is the only chapter of Part IV, we summarize the contributions of this dissertation, discuss ideas for combining the work presented in Parts II and III, and conclude.

Chapter 2

Motivations

In this chapter, we provide the practical and theoretical motivations for the work presented in this dissertation.[1] The practical motivation of supply chains is presented first, followed by the research-oriented motivation of integrating scheduling with related areas of research in Section 2.2. In Section 2.3, we briefly discuss the problems we have chosen to study in this dissertation. The chapter concludes in Section 2.4.

2.1 Practical Motivation: Supply Chains

Consider a supply chain with M manufacturing facilities producing different types of products, S suppliers providing component parts for these factories, and L merge-in-transit (MIT) facilities, where products from different manufacturers are combined according to customer order specifications. A schematic representation of this setting is given in Figure 2.1.

The manufacturers are focused factories.[2] Thus, the production area of each facility is separated into sub-areas according to different *product families*. A sub-area is composed of several assembly lines, each of which is dedicated to a particular *product type*.[3] Product types are configured in different ways to create specific *products*. The fabrication of each *product* requires component parts, which are procured from suppliers. Some types of components are shared among several products, while others are used only by one specific product.

[1] Parts of this chapter have been published in a workshop paper (Terekhov et al., 2010), a conference paper (Terekhov et al., 2012c), and a journal paper (Terekhov et al., 2012a). This material is used with permission of the corresponding copyright holders, the Association for the Advancement of Artificial Intelligence © and Elsevier ©, and the papers' co-authors.

[2] The term *focused factory* was created in the paper by Skinner (1974) and refers to grouping of products or processes that have similar characteristics (e.g., similar manufacturing equipment or market demand) so that each group would be produced in an isolated plant or "plant-within-plant" (Souza et al., 2001).

[3] Boysen et al. (2009) discuss the different types of assembly lines.

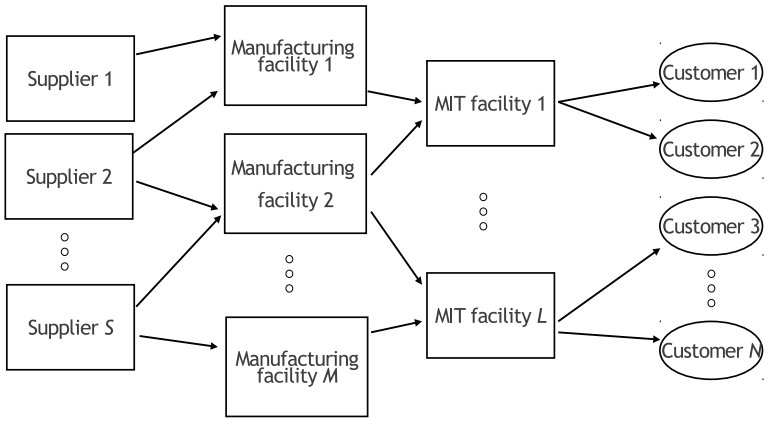

Figure 2.1: An Example of a Supply Chain.

Customer orders arrive dynamically over time. A customer order typically requires products from several manufacturers in various quantities. Once products are completed, they are sent to an MIT facility, where they are packaged together according to customer specifications. Each customer order has a due date and a weight that represents its priority or the penalty that the company has to pay if the order is delivered to the customer after the due date.

Such a setting exists, for example, in the supply chain of Alcatel-Lucent – a global corporation providing telecommunications equipment and solutions to service providers, enterprises and governments. Network deployment projects often generate customer orders that are composed of products from different families (e.g., base stations, routers) manufactured in different plants or at different production lines in the same facility. Finished products are shipped to the nearest MIT facility (typically a regional warehouse) close to the customer, where they are merged into a single order with the necessary ancillary equipment like cables and connectors, and the order is shipped out to the customer location.

In managing a supply chain such as the one described above, the goal is to achieve customer satisfaction at minimum cost (Maravelias and Sung, 2009). It is well-known from the literature that the efficiency of a supply chain can be improved "through proper coordination of material, financial and information flows" (Maravelias and Sung, 2009, p. 1). Material coordination, in turn, requires effective production scheduling. Specifically, for every arriving customer order, it is necessary to determine the time at which manufacturing of each of the constituent products will begin. This scheduling problem possesses the following three characteristics:

1. It is *combinatorial* in nature: each order is composed of multiple products, and there are many possible ways to sequence the fabrication of these products at different facilities.

2. It is *dynamic* and affected by many sources of *uncertainty*. The dynamism of the problem

arises from changes in the supply chain situation as orders arrive, are manufactured and then sent to the customers. The sources of uncertainty include delays in the arrival of component parts, machine breakdowns and adjustments in customer order specifications, among others.[4]

3. It is *related to other decision-making processes* of the environment. For example, production scheduling of the manufacturing facilities depends on the availability of component parts (inventory management decisions), availability of workers (staffing decisions), availability and routing of transportation between the manufacturers and the MIT facilities (logistics decisions).[5]

In fact, these three characteristics are true for most real-world scheduling problems. However, classical scheduling research[6] has focused mainly on addressing the combinatorics of isolated, static, deterministic problems. Thus, the aim of this dissertation is to provide a step toward creating methodologies to effectively address more realistic scheduling problems. We believe that these problems cannot be solved using purely the tools of scheduling; the complexity of the above characteristics warrants integration of scheduling methodologies with related fields of study.

2.2 Theoretical Motivation: Integration of Scheduling with Related Fields

From a research perspective, integration of scheduling methodologies with those of related fields of study is interesting and can lead to the development of a richer framework for solving resource allocation and sequencing problems. In this dissertation, we focus on combining scheduling with inventory management and queueing theory.

[4]We employ a very general definition of the term *uncertainty* in this dissertation. It describes environments that have at least one problem parameter that is not known with certainty at the time of scheduling, and includes cases with both known and unknown distributions for values of these parameters.

[5]Of particular interest is the work on inventory routing (Dror and Ball, 1987; Moin and Salhi, 2006) which, similarly to this dissertation, combines two fields of research in order to better model and solve real problems in supply chain settings. We do not discuss this work in detail in the current dissertation, since we are combining scheduling with inventory and queueing theory, rather than combining inventory with other areas. Investigation of the connections between inventory routing and scheduling is an interesting direction for future work, as is mentioned in Section 11.2.2.

[6]In this dissertation, we do not utilize the developments of stochastic scheduling (Pinedo, 2003; Baker and Trietsch, 2009) and online scheduling (Pruhs et al., 2004; Pruhs, 2007). However, we talk about stochastic scheduling as part of our future work in Sections 5.2.3 and 11.2.1. The relationship between stochastic problems and the problem type we study in this dissertation is discussed in Section 6.3. Online scheduling is discussed in Section 6.2.

2.2.1 Combining Scheduling and Inventory Management

As mentioned earlier, effective supply chain management necessitates proper coordination of material flows across the supply chain (Maravelias and Sung, 2009). Such coordination involves all steps in the transformation of raw materials into final products sent to the customer. Scheduling and inventory management have addressed different aspects of this coordination.

The inventory management literature is focused on the product: it aims to determine the quantity that should be delivered to a facility and the timing of this delivery. In inventory models, the manufacturing process of an item is typically treated as a black box which specifies the amount of time it takes to procure an item (lead time), and/or a production cost. An inventory model is therefore typically not concerned with the details of the production process.

The scheduling literature, on the contrary, *is* focused on the manufacturing process. In particular, it deals with the allocation of resources to the items that need to be manufactured and with the sequencing of production on each resource. Scheduling models can represent the availability of component parts using release dates, inventory costs of finished products using the notion of earliness, and costs of delays in delivery of completed items using the concept of tardiness. However, they have traditionally[7] not been concerned with the details of component part procurement or with what happens to the products after they are manufactured.

Thus, inventory management and scheduling have focused on different aspects of supply chain management. Each of these fields has traditionally included only an abstract representation of the other in its models. In this work, we partially remove the barrier between the two areas by including a detailed representation of an inventory replenishment policy within a scheduling model. By doing so, we are able to address a realistic scheduling problem that is both combinatorial and dependent on another decision-making process in its environment.

2.2.2 Combining Scheduling and Queueing Theory

The problem of scheduling in a dynamic environment involves a long time horizon and has to somehow deal with all the possible realizations of the job arrival process and of the job characteristics. The ultimate goal in solving this problem is to construct a schedule that would be optimal for the specific realization that actually occurs. The quality should be close to that of the schedule that could have been constructed if all of the uncertainty had been revealed *a priori*. Clearly, this is a difficult task, because to make a decision, one can use only the information that is known with certainty at that particular decision point and the stochastic properties of scenarios that may occur in the future. In addition, the effect of the decision on

[7] There has been some recent work linking inventory and scheduling, which is discussed in Chapter 3.

both short-run and long-run performance has to be considered. To deal with such problems, queueing theory and scheduling have adopted different approaches.

Queueing theory has taken the viewpoint that, since it is impossible to create an optimal schedule for every single sample path in the evolution of the system, one should aim to achieve optimal performance in some *probabilistic* sense (e.g., in expectation) over a long time horizon. This goal could be attained by construction of a policy based on the global stochastic properties of the system. For example, a policy could specify how start time decisions should be made each time a new job arrives or a job is completed. However, the schedule resulting from such a policy, while being of good quality in expectation, may be far from optimal for the particular realization of uncertainty that occurs. Moreover, queueing theory generally studies systems with simple combinatorics, as such systems are more amenable to rigorous analysis of their stochastic properties.

In the scheduling community, a dynamic scheduling problem is generally viewed as a collection of linked static problems. This viewpoint implies that methods developed for static scheduling problems become directly applicable to dynamic ones. Such methods can effectively deal with complex combinatorics and can optimize the quality of the schedules for each static sub-problem. However, the methods based on solving linked static problems tend to overlook the long-run performance and the stochastic properties of the system, with notable exceptions being the work on anticipatory scheduling (Branke and Mattfeld, 2002) and online stochastic combinatorial optimization (Van Hentenryck and Bent, 2006).

Thus, queueing theory and scheduling have differing views on dynamic problems. In this work, we leverage these differences in order to gain a better understanding of dynamic scheduling, and use the integration of the two as a means to address problems that are both combinatorial and dynamic.

2.3 The Approach of this Dissertation

Based on our practical and theoretical motivations we chose to explore several problems in this dissertation, each of which exhibits the realistic characteristics discussed in Section 2.1 and provides a good basis for studying the integration of research areas presented in Section 2.2.

2.3.1 Scheduling and Inventory Management

The problem we use as a basis of our study of scheduling with inventory is a special case of the supply chain described in Section 2.1, with two manufacturing facilities and one MIT facility. We assume that all facilities have the same owner and that there is a centralized decision-

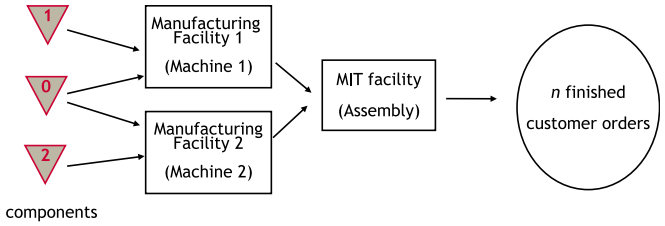

Figure 2.2: Schematic representation of the assembly scheduling problem.

maker who can optimize the performance of the overall system. Adopting a high level view of the problem, we model the two manufacturers as unary-capacity machines producing *sub-assemblies* which are then sent to an assembly machine that represents the MIT facility. In other words, a *sub-assembly* is assumed to be composed of the necessary quantity of products belonging to a customer order that need to be processed at a particular manufacturing facility. The processing times for each sub-assembly are determined by the quantity and configuration of products that are requested by the customer. We model the MIT facility as an infinite-capacity assembly resource (or, equivalently, as a single machine with a negligible processing time for each customer order). A customer order cannot be processed by this assembly machine unless the two sub-assemblies have both been completed. Figure 2.2 is a schematic representation of this setup.

To be manufactured, each sub-assembly requires component parts, the mix and quantity of which are dependent on the type and quantity of products making up the sub-assembly. The components may be unique to a sub-assembly, or may be shared among all or a subset of the sub-assemblies scheduled on a machine or on both machines. Figure 2.2 depicts the different categories of components as labelled triangles: triangle 1 represents all components that are consumed only at machine 1, which includes components unique to particular sub-assemblies processed on machine 1, and components shared among sub-assemblies on machine 1; triangle 2 represents all components used only at machine 2; triangle 0 corresponds to the components shared among the two machines. All components are replenished periodically at known points in time.

Each customer order has a due date and a weight for representing the penalty the company has to pay if the order is not completed before the due date. Given a set of n customer orders, each consisting of two sub-assemblies with specific processing times and component requirements, the goal of the problem is to schedule each sub-assembly on the corresponding manufacturing machine so as to minimize the total weighted order tardiness. We refer to this

problem as the assembly scheduling problem with inventory constraints.

Clearly, this problem is considerably simpler than the supply chain problem discussed in Section 2.1. However, the problem exhibits two of the characteristics of real scheduling problems: it is combinatorial and related to other decision-making processes of its environment, i.e., inventory management. By solving it, we push scheduling research to address more realistic problems, since we explicitly model the dependence of scheduling on the inventory replenishment policy. We also take a fundamental step in the direction of creating a general framework for making both types of decisions, since it is necessary to model and understand the effect of a fixed replenishment policy on the construction of a schedule.

2.3.2 Scheduling and Queueing Theory

The integration of queueing theory and scheduling can take place on three levels: conceptual, theoretical and algorithmic. At the conceptual level, we aim to combine concepts, ideas and problem settings from the two areas. Thus, the two problems we investigate have properties of systems typically studied in queueing theory as well as in classical scheduling research. Specifically, we consider a flow shop and a polling system with a flow shop server. The flow shop environment was chosen because it is one of the simplest classical systems for which queueing theory and scheduling propose differing approaches. Similarly, replacing the single machine server by a flow shop server in a polling system is one of the simplest ways to introduce combinatorial scheduling into a pure queueing model. The two settings are dynamic and possess an underlying combinatorial structure.

In both environments, we assume knowledge of processing time distributions and that the actual processing time of a product becomes known with certainty at the time of the order's arrival. The first of these is a typical queueing assumption, while the second is a classical scheduling one. This combination of assumptions is usually not addressed in the literature. It is also motivated by the supply chain of Section 2.1. In particular, processing time distributions for *product types* can be obtained based on historical data; at the same time, there may be essentially no variance in the processing time of a product once its configuration is specified by the customer.

The theoretical level of integration deals with combining methodologies and theoretical notions. In particular, we introduce the queueing notion of stability into the scheduling literature. We show that queueing methodologies can be used to obtain long-run performance guarantees for scheduling algorithms that have previously been available only for queueing policies. This study is based on the flow shop environments mentioned above.

The algorithmic level of integration concerns the development of hybrid queueing/scheduling

algorithms. Our ideas for this level are discussed in terms of a wider range of problems as part of our future work.

It is clear that the problems we study are not as complex as the supply chain problem described in Section 2.1. However, our work shows that combining ideas, concepts and methodologies from queueing theory and scheduling can be beneficial for obtaining a new understanding of scheduling problems which are both combinatorial and dynamic. Thus, it is a step toward developing methodologies for addressing realistic supply chain problems.

2.4 Conclusion

In this chapter, we presented the practical and theoretical motivations for this dissertation. From the practical perspective, we know that real-world scheduling problems are combinatorial, dynamic and uncertain, and related to other decision-making processes of their environment. Since scheduling research has traditionally focused on combinatorial properties of systems, there is a need to address the remaining two characteristics. From a theoretical perspective, we know that, within operations research, some areas of study address related problems in different ways. In particular, inventory management and scheduling have focused on different aspects of supply chain management, while queueing and scheduling have adopted contrasting viewpoints of dynamic scheduling problems. Motivated by practice and theoretical research goals, we study the integration of scheduling with both inventory management and queueing theory with the objective of addressing realistic scheduling problems.

The next part of this dissertation examines the problem described above in Section 2.3.1. In part III, we study the combination of scheduling and queueing theory as discussed in Section 2.3.2.

Part II

Assembly Scheduling with Inventory Constraints

Chapter 3

Combining Scheduling and Inventory Management: A Literature Review

In the previous chapter, we described the supply chain scheduling problem that serves as the motivation for the work presented in this part of the dissertation. This problem is combinatorial, stochastic and dynamic, and involves both scheduling and inventory decisions. Due to the problem's complexity, in Chapter 4 we study its static and deterministic version, focusing on the combinatorics and the impact of inventory on scheduling.

In the current chapter, we firstly review the literature on the underlying combinatorial scheduling problem. Secondly, we present a brief overview of the work done in inventory management. We conclude this section by surveying previous work in combinatorial scheduling that involves inventory (component availability) constraints.[1]

3.1 Combinatorial Scheduling

We start this section by presenting fundamental scheduling notions, followed by a review of the literature on problems having the assembly structure of the supply chain problem of interest.

3.1.1 Fundamental Scheduling Notions

Scheduling is the decision-making process of allocating scarce resources to tasks over time with the goal of optimizing one or several objectives (Pinedo, 2003). For example, in a manufacturing environment, the resources are the machines, while the tasks correspond to different steps in the production process of an item; in a university, the resources are the lecture rooms

[1] Parts of the work presented in this chapter have been published in a journal paper (Terekhov et al., 2012a). This material is used with permission of the copyright holder, Elsevier ©, and the paper's co-authors.

and the tasks are the lectures; in an airport, the resources are the runways and the tasks are the take-offs and landings of planes (Pinedo, 2003, 2009). In the case of the supply chain scheduling problem described in the previous chapter, the resources are the different facilities of the supply chain, while the tasks correspond to the production of sub-assemblies or the assembly and packaging of orders.

Since most of the early scheduling research was concerned with manufacturing environments, the terms typically used to describe a scheduling problem are based on the manufacturing vocabulary (Baker and Trietsch, 2009). Thus, resources are referred to as *machines* and tasks as *jobs*. Frequently, jobs are composed of multiple production steps called *operations* or *activities*.

A significant portion of the combinatorial scheduling literature focuses on problems which correspond to a snapshot of the environment at a particular point in time (i.e., are *static*) and in which all relevant job characteristics are known with certainty (i.e., are *deterministic*). These assumptions allow for a deeper study of the combinatorics of these problems, and are the assumptions we adopt in this part of the dissertation. The reader is referred to the books of Pinedo (2003) and Baker and Trietsch (2009) for overviews of static, *stochastic* problems, where a given set of n jobs with characteristics that are not known with certainty, but rather are specified by probability distributions, has to be scheduled. Chapter 6 covers work on *dynamic* scheduling, which considers the evolution of the system over a long period of time and takes into account new job arrivals. Section 6.3 discusses the relationship between the various types of scheduling problems.

Formally, a static, deterministic scheduling problem aims to determine the start times of a set of n jobs, where each job j may be characterized by (Pinedo, 2003):

- a *processing time*, p_{ij}, the amount of time for which job j requires the use of machine i;
- a *release date*, r_j, the earliest time at which the job may begin;
- a *due date* or *deadline*, d_j, the date by which a job should be finished (e.g., the date the job is promised to be sent to the customer); if d_j is a due date, then the job may complete after d_j while incurring a penalty; if d_j is a deadline, then the job is not allowed to complete any later than d_j;
- a *weight*, w_j, the priority factor or cost associated with the job.

The assignment of start times to jobs is constrained by the capacity of the resources. The scheduling horizon is assumed to be discretized into time slots.

The notation used for describing a scheduling problem is $\alpha|\beta|\gamma$, where α represents the machine environment, β represents job processing characteristics and constraints, and γ is

the objective function (Graham et al., 1979; Pinedo, 2003). The machine environment field (α) usually states the number of machines and the relations among them. Some examples of machine environments are (Pinedo, 2003):

- a single machine, denoted as 1;
- a flow shop, denoted as Fm, where there are m machines in series, and every job needs to be processed on each machine and follows the same route through the machines;
- a job shop, denoted as Jm, where there are m machines and each job has its own (predetermined) processing route through the machines.

In this dissertation, we make the assumption, unless stated otherwise, that all machines are of *unary capacity*, which implies that at most one job can be processed on a machine at any point in time.

The job characteristic field (β) may state, for example, whether the problem involves release dates (r_j) or preemptions ($prmp$). If preemptions are allowed, then a job's processing on a machine may be interrupted by another job prior to its completion. In this dissertation, we consider problems without preemptions only.

Following the standard notation in scheduling (Pinedo, 2003), we let C_j denote the completion time of a job (i.e., the time at which it exits the system). Some examples of typical minimization objectives (γ) are:

- makespan, $C_{max} = \max\{C_1, \ldots, C_n\}$, the completion time of the set of n jobs;
- total weighted completion time, $\sum w_j C_j$;
- total weighted tardiness, $\sum w_j T_j$, where the tardiness $T_j = \max\{C_j - d_j, 0\}$.

An empty α field implies a single-machine environment, while an empty β refers to the fact that preemptions are not allowed (Pinedo, 2003). Thus, $F2||\sum C_j$ describes a two-machine flow shop scheduling problem with the goal of minimizing the sum of completion times.

The literature on scheduling is enormous, and, therefore, in the remainder of this chapter we survey only the work that is directly relevant to modelling and solving the scheduling problem with inventory constraints that was introduced in Section 2.3.1. For more details on deterministic scheduling, the reader is referred to the applications-oriented book of Pinedo (2009), the theory-oriented book of Pinedo (2003), and the books by Baker and Trietsch (2009) and Leung (2004).

3.1.2 Scheduling with an Assembly Structure Among Machines

One of the defining characteristics of our inventory and scheduling problem is the fact that sub-assemblies produced at the two facilities have to be packaged together at the MIT facility. Adopting a high level view of the problem, we can model the two manufacturing facilities as unary capacity machines producing sub-assemblies which are then sent to an assembly machine that represents the MIT facility. In the absence of inventory constraints, our problem can be formulated as one of the following problems in the literature: order scheduling, assembly flow shop scheduling or assembly job shop scheduling.

In *order scheduling* (Leung et al., 2005b; Lin and Kononov, 2007), orders are assumed to be composed of m operations, each requiring a machine. Operations on different machines may be performed concurrently, and an order is complete when all m operations have been performed. The scheduling problem in such an environment has also been referred to as the *concurrent job shop scheduling problem* (Roemer, 2006), the *problem of scheduling customer orders* (Ahmadi and Bagchi, 1990), the *open shops with jobs overlap problem* (Leung et al., 2005c) and *scheduling with bundled operations* (Li and Vairaktarakis, 2007). The order scheduling literature considers scheduling with identical parallel machines (Yang and Posner, 2005), any of which can process any product of an order, and with dedicated machines (Leung et al., 2007; Li and Vairaktarakis, 2007; Sung and Yoon, 1998). Of interest in this part of the dissertation is the dedicated machine environment, denoted in the α field by PDm. The literature includes work on complete and heuristic approaches for solving scheduling problems arising in this environment, as well as on the development of theoretical properties. A mixed-integer programming (MIP) formulation for this setting with the minimum total weighted completion time objective is given by Ahmadi et al. (2005). This formulation is very similar to the *positionalVariables* model that we develop in Section 4.3.2. More recent work on the concurrent open shop problem includes the paper by Mastrolilli et al. (2010) and a book chapter by Huang and Lin (2007).

The *assembly flow shop* literature deals with a more general problem: it explicitly models the assembly operation that is necessary for combining the products produced by the first-stage parallel operations. Initial work in this area was done by Lee et al. (1993), based on two first-stage machines and the objective of minimizing makespan, C_{max}, the completion time of the last scheduled job on the assembly machine. Potts et al. (1995) generalize many of the results of Lee et al. to the m-machine case. Specifically, they show that to address the problem of makespan minimization in an m-machine assembly flow shop, it is sufficient to look for permutation schedules, i.e., ones in which the order of processing the jobs is the same on all machines. They also prove the problem's strong NP-completeness based on a reduction from the 3-PARTITION problem. Following these initial results, Sun et al. (2003) focus on the

creation of better heuristics, while Hariri and Potts (1997) develop a better branch-and-bound algorithm. Additional methods for solving assembly flow shop problems include tabu search and particle swarm optimization (Allahverdi and Al-Anzi, 2006). The reader is referred to the papers by Hejazi and Saghafian (2005) and Blocher and Chhajed (2008) for extensive reviews of the literature on assembly flow shop scheduling and related problems.

Another relevant area is *assembly job shop scheduling*, which studies more general settings than does assembly flow shop scheduling. The assembly flow shop scheduling literature has mostly dealt with the examination of theoretical results, such as conditions for the optimality of permutation schedules and development of specialized branch-and-bound methods and heuristics with performance guarantees; assembly job shop scheduling has focused on the creation of more complex models as well as heuristics or effective dispatching rules for a variety of objectives (Yokoyama, 2008). For example, the papers by Pathumnakul and Egbelu (2006), Thiagarajan and Rajendran (2005) and Philipoom et al. (1991) examine various dispatching rules. McKoy and Egbelu (1998) and Pathumnakul and Egbelu (2006) give MIP formulations for the assembly job shop problem based on extensions of the standard disjunctive formulation (Pinedo, 2003) for scheduling problems. A time-indexed MIP is given by Masin et al. (2007). Chen and Ji (2007) present a disjunctive MIP formulation for an advanced planning and scheduling problem with orders consisting of multiple levels of production.

In addition to the assembly structure among facilities, our problem possesses another interesting characteristic: scheduling of production at the manufacturers depends on the availability of component parts from upstream suppliers. To understand this aspect of the problem, we consult the inventory management literature, surveyed next.

3.2 Inventory Management

We start this section by presenting classical concepts in inventory management, followed by a review of the literature which addresses inventory availability in scheduling models.

3.2.1 Classical Inventory Management Concepts

Inventory management, as defined by the American Production and Inventory Society, is a branch of business management that deals with the planning and control of inventories (Toomey, 2000). More specifically, the role of inventory management is to maintain desired stock levels of various products (Toomey, 2000). The goal of an inventory control system is, then, to determine the *timing* and *order quantities* of replenishments for these products (Axsäter, 2006).

Inventory management is of utmost importance for managing supply chains (Axsäter, 2006), and, therefore, is crucial for the supply chain of interest in this dissertation. Specifically, in our setting, an inventory management policy controls the arrival of component parts to the manufacturers whose production we aim to schedule. Studying this problem under the assumption of a fixed inventory policy provides a first step toward solving the more realistic problem in which the component replenishment policy needs to be determined in addition to the schedule of production at the manufacturers (see future work in Section 5.3). Note also that if we assume that a schedule of production at the manufacturers is fixed, then we obtain an inventory management problem in which demand for the components is defined in terms of that schedule.

In general, inventory management problems are based on four cost types (Axsäter, 2006; Beyer et al., 2010):

- *holding* or *carrying* cost, which is the cost of keeping an item in inventory for a period of time. It consists, for example, of the opportunity cost of tied-up capital, material handling costs and storage costs. The holding cost is typically determined as a percentage of the dollar value per unit of time.

- *ordering* or *setup* costs, which are fixed costs associated with an order. They are generally independent of the size of the replenishment, and may include costs for setup of a machine as well as administrative, transportation and material handling costs.

- *shortage* or *stockout* costs, which are incurred if an item is not available when there is customer demand for it. In such a case, the order is backlogged if the customer agrees to wait; otherwise the order is lost. If the order is backlogged, costs are incurred due to price discounts for late delivery, administration, material handling and transportation. In the case of a lost order, the revenue from the sale is lost. In both cases, there is also a loss of goodwill which can affect sales in the long run but whose cost is difficult to estimate. In some cases, shortage costs are replaced by *service level constraints*, which refer to stock availability in the *expected* or *probabilistic* sense and do not include other aspects of service (Chen and Krass, 2001).

- *purchase* or *production* cost, which is the cost of producing or buying items. It is usually specified as a cost per unit multiplied by the quantity produced or purchased.

According to Chen and Krass (2001), the literature on inventory management is frequently divided into the study of *full cost* models and *partial cost* models with service level constraints. In full cost models, the goal is to determine a policy for replenishing the inventory that minimizes the sum of holding, procurement[2] and shortage costs. In the partial cost model, the

[2]Procurement costs usually refer to purchase and ordering costs combined.

goal is to find a replenishment policy that minimizes the sum of holding and procurement costs subject to a service level constraint. As stated by Chen and Krass (2001), many authors (Bookbinder and Tan, 1988; Cohen et al., 1988; Nahmias, 1993) have noted that models with service level constraints are more popular in real inventory systems than are full cost models. However, service level models have generally received less attention in the theoretical inventory literature (Chen and Krass, 2001).

The inventory ordering systems studied in the literature are divided into periodic-review and continuous-review systems. Both of these are based on the notions of *inventory position* and *inventory level* (Axsäter, 2006), which are defined as follows:

- inventory position = stock on hand + outstanding orders - backorders,
- inventory level = stock on hand - backorders,

where the *stock on hand* corresponds to the physical amount of inventory available, the *outstanding orders* are the replenishment orders that have not yet arrived, and the *backorders* are the items that have been requested by the customer but not yet delivered.

A continuous-review inventory system is based on monitoring the inventory position continuously, while periodic review considers the inventory position only at specific given points in time (Axsäter, 2006). Continuous review is typically used for items with low demand, while periodic review is used for items with high demand. The distinction between continuous and periodic-review approaches is also relevant for dynamic scheduling systems discussed in Chapter 6 (see Section 6.1.2.3.2 in particular).

Typical inventory policies studied in the literature are the (R, Q) policy and the (s, S) policy (Axsäter, 2006). Under the (R, Q) policy, a replenishment order of size Q is placed whenever the inventory position decreases to or below the re-order point R. Under the (s, S) policy, when the inventory position decreases to or below s, an order is made to replenish the inventory to S. In our scheduling problem with inventory constraints, we assume a simpler replenishment policy. That is, we assume that a fixed quantity of items arrives periodically during our scheduling horizon. This replenishment policy is based on periodic review and is similar to the (R, Q) policy in that a batch of Q items is ordered at every review point. However, unlike in the (R, Q) policy, the decision to order is not dependent on the inventory position. Considering other policies for the assembly scheduling problem is one direction for future work discussed in Chapter 5.

As in the case of scheduling, the literature on inventory problems is very large, and hence is not fully surveyed here. The reader is referred to the books by Axsäter (2006), Beyer et al. (2010), Nahmias (1993), Silver et al. (1998) and Toomey (2000). In the next section, we survey the work on scheduling models that explicitly take into account the availability of inventory.

3.2.2 Inventory Constraints in Scheduling

Inventory (component availability) constraints have been considered in both the project scheduling literature and the machine scheduling literature.

Resource-constrained project scheduling problems (RCPSPs) deal with scheduling of various activities that are part of a project and related by precedence constraints. Each activity requires resources in order to be performed. The assembly scheduling problem with inventory constraints considered in this part of the dissertation can be viewed as a special case of the RCPSP: the assembly structure can be modelled via precedence constraints between operations on first-stage machines and the assembly machine, while each component type corresponds to a resource. RCPSP literature surveys include those by Brucker et al. (1999), Hartmann and Briskorn (2010), Herroelen et al. (1998) and Özdamar and Ulusoy (1995). Neumann and Schwindt (2002) look at a project scheduling problem with cumulative resources, which are periodically replenished and have a maximum and minimum allowable level of inventory.

Kolisch (2000) and Kolisch and Hess (2000) study assembly scheduling in the presence of component constraints. Both papers consider a make-to-order environment in which a set of end-products has to be assembled for customers by specific due dates. The objective is to assign component parts to operations, and schedule operations subject to resource availability, assembly area capacity and technological precedence constraints so as to minimize the total weighted tardiness. Kolisch (2000) proposes a MIP model based on an RCPSP view of the problem and a list scheduling heuristic. Kolisch and Hess (2000) develop a biased random sampling approach and a tabu search-based large-step optimization method. Since our problem is a special case of the problem studied in these papers, we included an adaptation of Kolisch's MIP model in our study.

Few papers have been published that focus on the investigation of scheduling under inventory constraints in machine scheduling problems (specializations of the project scheduling problem). One of the earliest papers on this subject is by Beck (2002), who extends the job shop scheduling problem by representing inventory production, consumption and storage in a constraint-directed framework. Grigoriev et al. (2005) establish the complexity of scheduling a single machine with raw material constraints and the goal of either minimizing the number of tardy jobs or the makespan. They show that some variants of the problem are strongly NP-hard even when unit processing times are assumed, although polynomially-solvable cases exist. Additionally, the performance of several heuristics is compared against a MIP model for the problem of minimizing makespan with multiple shared components.

There is also a series of papers motivated by the problem of scheduling trucks at a transshipment centre. Briskorn et al. (2010) model the problem with a single gate at the centre as a machine scheduling problem where trucks correspond to jobs and the gate corresponds to a ma-

chine. When a truck is unloaded, inventory at the centre is replenished; when items are loaded onto a truck, inventory is being used up. This problem, like the one studied by Beck (2002), allows for scheduling of replenishment activities (we assume the replenishment policy is fixed), but we consider the more complex assembly scheduling environment. Moreover, the problem we consider can be viewed as a special case of Briskorn et al.'s (2010) directions for future work: we address the total weighted tardiness objective, the case when jobs have different release dates and there is more than one machine, as well as having different types of inventory and different sub-assemblies produced on the two machines. Briskorn et al. (2009), Briskorn and Leung (2010) and Briskorn (2010) address the same inventory-constrained single machine scheduling environment as Briskorn et al. (2010), proposing genetic algorithm, branch-and-bound and variable very large neighbourhood search approaches, respectively. Briskorn et al. (2012) study properties of optimal solutions and design branch-and-bound and dynamic programming algorithms for the single machine problem with the goal of minimizing the total weighted completion time.

3.3 Summary

Our problem of interest is different from the literature on scheduling with an assembly structure among machines in several respects. Firstly, to our knowledge, the papers that deal with a two-stage, two-machine structure consider neither the possibility of different operations of a job having distinct release dates nor component availability constraints. In this sense, our work generalizes and extends the problems that have been addressed in the order scheduling and assembly flow shop literature. Secondly, unlike most of the work on assembly job shop scheduling, this dissertation presents extensive experimental results comparing *complete* approaches. Thirdly, in addition to MIP models, we consider the use of constraint programming, which has been shown to be effective for scheduling problems (Baptiste et al., 2006; Fox, 1983) but has not, to our knowledge, been previously compared to MIP models in the context of assembly scheduling.

In addition, the problem we study can be seen as an extension of the problems in the machine scheduling with inventory literature. This literature has studied classical single-machine and job-shop environments only, and has not, to our knowledge, considered assembly-type precedences between jobs and objectives based on orders. The RCPSP literature, on the contrary, addresses problems that are more complex than ours due to a greater number of resources, components and interactions between activities. Because of such complexity, Kolisch and Hess (2000) and Kolisch (2000) concentrate on the development of heuristic approaches. For our problem, investigation of complete methods is much more suitable, and hence is our focus.

3.4 Conclusion

The assembly scheduling problem with inventory constraints introduced in the previous chapter combines aspects of problems studied in scheduling and inventory management research. In this chapter, we have provided an overview of the relevant literature from both of these areas. Specifically, we firstly presented a summary of fundamental scheduling concepts and a review of scheduling in assembly environments. Secondly, we presented classical inventory management notions and surveyed the literature on scheduling with inventory constraints. Finally, we discussed how our approach differs from previous work on similar problems.

In the next chapter, we give a formal statement of the assembly scheduling problem with inventory constraints. We study some of its theoretical properties, and then present three mixed-integer programming models and a constraint programming model for solving it. We empirically evaluate the performance of the models on an extensive set of test instances. In Chapter 5, we present some ideas for future work on this problem.

Chapter 4

Solving An Assembly Scheduling Problem Using Complete Methods

In this chapter, we consider a scheduling problem with component availability constraints in a supply chain consisting of two manufacturing facilities and a merge-in-transit facility.[1] The realistic supply chain problem that motivates this study is presented in Chapter 2. Three mixed-integer programming (MIP) models and a constraint programming (CP) model are compared in an extensive numerical study. Results show that when there are no components shared among the two manufacturers, the MIP model based on time-index variables is the best for proving optimality for problems with short processing times whereas the CP model performs better than the others for problems with a large range of processing times. When shared components are added, the performance of all models deteriorates, with the time-indexed MIP providing the best results.

This chapter has three main contributions. Firstly, we develop three MIP models and a CP model for solving a realistic scheduling problem with inventory constraints and an assembly structure. All of these models are implemented using commercially available software. Secondly, we provide an extensive experimental evaluation of their effectiveness. Thirdly, by explicitly modelling the dependence of scheduling decisions on the availability of component parts, we contribute to the literature on the integration of inventory and scheduling decisions, which is necessary for solving realistic supply chain problems.

This chapter is organized as follows. In Section 4.1, we present a detailed description of the problem. We discuss theoretical properties of the problem in Section 4.2. Three MIP models are given in Section 4.3 and a constraint programming model is stated in Section 4.4. In Section 4.5, experimental results are presented. Section 4.6 concludes the chapter.

[1]The work presented in this chapter has been published as a journal paper (Terekhov et al., 2012a). This material is used with permission of the copyright holder, Elsevier ©, and the paper's co-authors.

4.1 Problem Description

We consider an environment with $m = 2$ unary capacity *first-stage* machines[2] and an infinite capacity *second-stage* machine. A set of n orders (jobs) is given, such that each order j, $j = 1, \ldots, n$, consists of two sub-assemblies (activities). Sub-assembly i of order j, denoted a_{ij}, has to be processed on machine i and requires p_{ij} units of time. A sub-assembly also needs components. These components may be specific to a sub-assembly, or may be shared among different sub-assemblies or machines.

Each activity a_{ij} has release date r_{ij} which corresponds to the time at which all component parts that are unique to this sub-assembly (i.e., not required by any other sub-assemblies) become available. For all components that are shared among the sub-assemblies processed on the same machine or among multiple machines, it is assumed that there are R replenishments: they occur at times 0, $\lceil H/R \rceil$, $\lceil 2H/R \rceil$, ..., $\lceil (R-1)H/R \rceil$, where $H = \max_i \{\max_j r_{ij} + \sum_{j=1}^{n} p_{ij}\}$. For each component type z, $\lceil (Q_z - l)/R \rceil$ units become available at the time of the lth replenishment, $l = 0, \ldots, R-1$, with Q_z denoting the total quantity of component z necessary for processing the given n orders.[3] This structure implies that the replenishment quantities are as equal as possible regardless of whether R divides H.

Let A_i be the set of all machine-specific components for machine i, $i = 1, 2$. Each activity a_{ij} requires α_{jz} units of machine-specific component z, $z \in A_i$. Similarly, every activity a_{ij} requires β_{ijz} units of component z, $z \in B$, where B denotes the set of components shared among the machines.

Without component availability constraints, all jobs can be scheduled within H time units. However, with the addition of such constraints, the start time of an activity depends on the time at which all components necessary for its processing become available. Thus, the actual scheduling horizon is $timeHorizon = H$ if the maximum release date is after the last replenishment time point, and $timeHorizon = \lceil (R-1)H/R \rceil + \sum_{j=1}^{n} p_{ij}$ otherwise.

Each order j has a due date d_j and a weight w_j. If order j is completed after d_j, its tardiness, T_j, is the difference between its completion time, C_j, and d_j; if the order is completed at or before d_j, then $T_j = 0$. The goal of the problem is to assign start times to the activities on each of the first-stage machines so as to minimize the total weighted tardiness of orders, $\sum_{j=1}^{n} w_j T_j$. This problem can be denoted as $PD2|r_{ij}, components| \sum w_j T_j$, where $components$ refers to the presence of component availability constraints.[4]

[2] Our models can be easily extended to the case with more than two machines.

[3] $Q_z = \sum_{l=0}^{R-1} \lceil (Q_z - l)/R \rceil$, which is an identity expressing the partition of Q_z into R *as-equal-as-possible* parts in non-increasing order (Graham et al., 1988, Equation (3.24)). The identity holds since Q_z is an integer and R is a positive integer.

[4] Refer to Section 3.1.1 for an explanation of the three field notation for describing scheduling problems.

The $PD2|r_{ij}, components| \sum_{j=1}^{n} w_j T_j$ problem is a generalization of $PD2\,|r_{ij}| \sum_{j=1}^{n} w_j T_j$, and, consequently, of $PD2|r_j| \sum_{j=1}^{n} w_j T_j$, where the release dates of sub-assemblies on both machines are identical. Furthermore, these problems generalize $PD1|| \sum_{j=1}^{n} T_j$, which, as noted by Leung et al. (2007), is NP-hard since it is equivalent to $1|| \sum_{j=1}^{n} T_j$ (which has been proved NP-hard by Du and Leung (1990)). Hence, it follows that all of its generalizations, including $PD2|\, r_{ij}, components \,| \sum_{j=1}^{n} w_j T_j$ are also NP-hard.

4.2 Theoretical Properties

Permutation schedules have been shown to be optimal for $PD2|| \sum_{j=1}^{n} w_j T_j$ (Cai et al., 2008; Potts et al., 1995). Hence, a natural question is whether the addition of inventory constraints changes this property.

For the $PD2|| \sum_{j=1}^{n} w_j T_j$ problem, Lemma 2.1 (ii) of the paper by Leung et al. (2005a) states that if there exists a machine on which all jobs require a shorter processing time than on another, then this machine can be disregarded. Our Lemma 4.2.1 is a similar result for $PD2|r_{ij}| \sum_{j=1}^{n} w_j T_j$.

Lemma 4.2.1. *Consider the $PD2|r_{ij}| \sum_{j=1}^{n} w_j T_j$ problem. If $r_{1j} \leq r_{2j}$ ($r_{2j} \leq r_{1j}$) and $p_{1j} \leq p_{2j}$ ($p_{2j} \leq p_{1j}$) $\forall j$, then machine 1 (2) can be disregarded in constructing the schedule. It is in fact sufficient to solve the single-machine problem for machine 2 (1) and schedule jobs on machine 1 (2) in the same order. This also corresponds to the case in which it is sufficient to consider permutation schedules.*

Proof. Suppose $r_{1j} \leq r_{2j}$ and $p_{1j} \leq p_{2j}$ $\forall j$. Let $s_{ij}(S)$ and $C_{ij}(S)$ be the start time and completion time of a_{ij} in a schedule S. For every schedule S that is feasible for machine 2 and was constructed by ignoring machine 1, there is a corresponding feasible schedule on machine 1 with $s_{1j}(S) = s_{2j}(S)$ $\forall j$. It is feasible for machine 1 since (a) $s_{1j}(S) = s_{2j}(S) \geq r_{2j} \geq r_{1j}$ and (b) for every pair of jobs j and l such that j precedes l in the schedule S, $s_{2j}(S) + p_{2j} \leq s_{2l}(S)$, which implies that $s_{1j}(S) + p_{1j} \leq s_{2j}(S) + p_{2j} \leq s_{2l}(S) = s_{1l}(S)$.

For every job j, $C_{1j}(S) = s_{1j}(S) + p_{1j} \leq s_{2j}(S) + p_{2j} = C_{2j}$ since $s_{1j}(S) = s_{2j}(S)$ and $p_{1j} \leq p_{2j}$. Therefore, $T_j = \max\{0, C_j - d_j\} = \max\{0, \max\{C_{1j}, C_{2j}\} - d_j\} = \max\{0, C_{2j} - d_j\}$ $\forall j$, and the objective value of S is determined by the schedule on machine 2.

Thus, the overall optimal schedule can be constructed by finding the optimal schedule on machine 2 and then setting the start times on machine 1 according to $s_{1j}(S) = s_{2j}(S)$ $\forall j$. The same logic applies to the reverse case when $r_{2j} \leq r_{1j}$ and $p_{2j} \leq p_{1j}$ $\forall j$. We can conclude that under the conditions stated in the lemma, it is sufficient to consider permutation schedules. □

j	w_j	r_j	p_{1j}	p_{2j}	d_j
1	100	5	1	2	7
2	1000	1	4	2	5
3	1	0	2	2	5

Table 4.1: Example of a $PD2|r_j|\sum_{j=1}^{n} w_j T_j$ problem for which there is no optimal schedule that is a permutation schedule.

j	r_{1j}	r_{2j}	p_{1j}	p_{2j}	d_j
1	0	2	3	4	6
2	4	0	1	1	6

Table 4.2: Example of a $PD2|r_{ij}|\sum_{j=1}^{n} T_j$ problem for which there is no optimal schedule that is a permutation schedule.

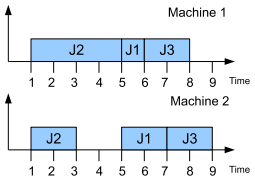

Figure 4.1: Permutation Schedule 1 for Example 1 (Table 4.1)

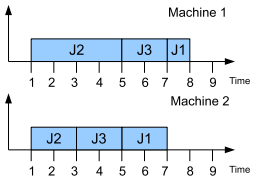

Figure 4.2: Permutation Schedule 2 for Example 1 (Table 4.1)

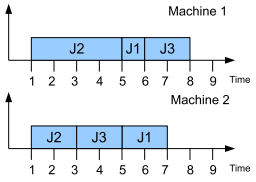

Figure 4.3: Non-permutation Schedule for Example 1 (Table 4.1)

However, if no assumptions about processing times are made, then even in the special cases when $r_{1j} = r_{2j}$ or when the weights of all orders are identical, it is not sufficient to consider permutation schedules for the $PD2\ |r_{ij}|\sum_{j=1}^{n} w_j T_j$ problem. We illustrate these observations via two examples.

Firstly, consider the problem instance in Table 4.1, in which $r_{1j} = r_{2j}\ \forall j$. Since there are 3 orders, there are 3! = 6 possible permutation schedules. However, due to the high weight and a relatively tight due date for order 2, the optimal schedule on both machines should have order 2 scheduled first. Thus, we need to consider only two permutation schedules: 2, 1, 3 (permutation schedule 1, shown in Figure 4.1) and 2, 3, 1 (permutation schedule 2, shown in Figure 4.2). The total weighted tardiness values of permutation schedules 1 and 2 are 4 (order 3 is tardy) and 102 (orders 1 and 3 are tardy), respectively. A non-permutation schedule with objective value 3 is given in Figure 4.3. There is no optimal schedule for this problem instance that is a permutation schedule.

The second example is shown in Table 4.2: the weights of all orders equal 1, but the release dates of activities belonging to the same order are different. The two permutation schedules are shown in Figures 4.4 and 4.5, with total weighted tardiness values of 1 and 2, respectively. The objective value of the non-permutation schedule in Figure 4.6 is 0. This example shows that even when all weights are 1, non-permutation schedules need to be considered.

Thus, for the $PD2|r_{ij}|\sum_{j=1}^{n} w_j T_j$ problem, it is in general not correct to restrict search to permutation schedules. Consequently, the same is true for $PD2|r_{ij}, components|\sum_{j=1}^{n} w_j T_j$.

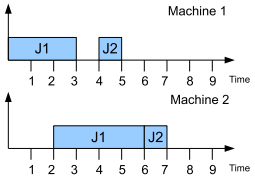

Figure 4.4: Permutation Schedule 1 for Example 2 (Table 4.2)

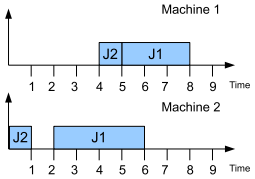

Figure 4.5: Permutation Schedule 2 for Example 2 (Table 4.2)

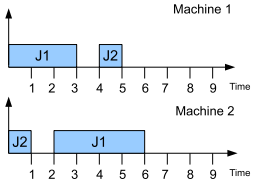

Figure 4.6: Non-permutation Schedule for Example 2 (Table 4.2)

We propose to solve the $PD2|r_{ij}, components| \sum_{j=1}^{n} w_j T_j$ problem using MIP and CP techniques.

4.3 Mixed-Integer Programming Models

We consider four models for our problem: the MIP model developed by Kolisch and Hess (2000), two MIP models based on extensions of single machine models (Keha et al., 2009), and a constraint programming model. In this section, we present the three MIP models. As is generally the case for scheduling models, time is discretized into periods $1, 2, \ldots, timeHorizon$. Each period t starts at time $t-1$ and ends at time t.

4.3.1 The *timeIndexed* Model

The *timeIndexed* model is based on decision variables x_{ijt}, which equal 1 if the processing of order j on machine i (equivalently, activity a_{ij}) *starts* at time t. The *timeIndexed* model is presented in expressions (4.1)–(4.13).

As defined previously, T_j and w_j are the tardiness and the weight of order j, respectively. The objective in the model is to minimize the total weighted tardiness of the set of n available orders. Equation (4.2) ensures that a_{ij} begins exactly at one point in time. In Equation (4.3), the variables C_{ij}, each of which represents the completion time of order j on machine i, are defined. Constraint (4.4) states that the completion time of order j, C_j, has to be greater than or equal to the completion time of each of its component activities. Constraint (4.5) states that at most one a_{ij} can be started on machine i at any point in time during the scheduling horizon. Constraints (4.6) and (4.7) ensure that each activity starts after its release date and in time for it to be completed during the time horizon. Constraints (4.8) define the tardiness of each order in terms of its completion time and its due date. The fact that the x_{ijt} variables are binary and the non-negativity of the completion time variables are given in constraint (4.9).

$$\min \quad \sum_{j=1}^{n} w_j T_j \tag{4.1}$$

$$\sum_{t=0}^{timeHorizon-p_{ij}} x_{ijt} = 1, \ \forall\, i,j \tag{4.2}$$

$$C_{ij} = \sum_{t=0}^{timeHorizon-p_{ij}} t x_{ijt} + p_{ij}, \ \forall\, i,j \tag{4.3}$$

$$C_j \geq C_{ij}, \ \forall\, i,j \tag{4.4}$$

$$\sum_{j=1}^{n} \sum_{\tau=\max\{t-p_{ij}+1,0\}}^{t} x_{ij\tau} \leq 1, \ \forall\, i,t \tag{4.5}$$

$$x_{ijt} = 0 \ \forall\, t < r_{ij}, \ \forall\, i,j \tag{4.6}$$

$$x_{ijt} = 0 \ \forall\, t > timeHorizon - p_{ij}, \ \forall\, i,j \tag{4.7}$$

$$T_j \geq C_j - d_j, \ T_j \geq 0 \ \forall\, j \tag{4.8}$$

$$C_j \geq 0, \ C_{ij} \geq 0, x_{ijt} \in \{0,1\}, \ \forall\, i,j,t \tag{4.9}$$

For each machine-specific component type z used for processing activities on machine i, constraint (4.10) has to be included in the model. This constraint states that the number of components used by all activities by the end of a replenishment period has to be smaller than or equal to the number of components that have become available by that time. Constraint (4.11) states that the total number of components of type z used by the end of the time horizon has to be smaller than or equal to the total number of these components that becomes available from time 0 to time $timeHorizon$. Constraint (4.11) is different from (4.10) because H and $timeHorizon$ are not the same. Constraints (4.12) and (4.13) are the equivalent of (4.10) and (4.11) for each component z shared between machines.

For each $z \in A_i$, $i = 1, 2$,

$$\sum_{j=1}^{n} \sum_{\tau=0}^{\left\lceil \frac{H(L+1)}{R} \right\rceil - 1} \alpha_{jz} x_{ij\tau} \leq \sum_{l=0}^{L} \left\lceil \frac{Q_z - l}{R} \right\rceil, L = 0, 1, \ldots, R-2, \tag{4.10}$$

$$\sum_{j=1}^{n} \sum_{\tau=0}^{timeHorizon-1} \alpha_{jz} x_{ij\tau} \leq Q_z \tag{4.11}$$

For each $z \in B$,

$$\sum_{j=1}^{n} \sum_{\tau=0}^{\left\lceil \frac{H(L+1)}{R} \right\rceil - 1} (\beta_{1jz} x_{1j\tau} + \beta_{2jz} x_{2j\tau}) \leq \sum_{l=0}^{L} \left\lceil \frac{Q_z - l}{R} \right\rceil, L = 0, \ldots, R-2 \quad (4.12)$$

$$\sum_{j=1}^{n} \sum_{\tau=0}^{timeHorizon-1} (\beta_{1jz} x_{1j\tau} + \beta_{2jz} x_{2j\tau}) \leq Q_z. \quad (4.13)$$

4.3.2 The *positionalVariables* Model

The *positionalVariables* model is based on two sets of decision variables: u_{ijk}, $i = 1, 2,\ j = 1, \ldots, n,\ k = 1, \ldots, n$, and γ_{ik}, $i = 1, 2,\ k = 1, \ldots, n$. u_{ijk} equals 1 if order j is assigned to position k in the processing sequence on machine i, and 0 otherwise. γ_{ik} corresponds to the completion time of the order in position k on machine i. As in the *timeIndexed* model, there are two additional sets of variables, $\{C_{ij}, \forall i, j\}$ and $\{T_j, \forall j\}$. C_{ij} denotes the completion time of a_{ij}, while T_j represents the tardiness of order j.

Constraints (4.15) and (4.16) define the completion time of the job in position k on each machine, while (4.17) and (4.18) express the relationship between the C_{ij} and the u_{ijk} variables. In our experiments, $\mathcal{M}$ is set equal to the length of the time horizon (i.e., H for the case without components and $timeHorizon$ for the case with components). The relationship between the completion times of activities and the completion time of the order is given in expression (4.19). Equations (4.20) and (4.21) state that there has to be a one-to-one correspondence between activities and positions in the processing sequence on each machine. Constraints (4.22) define the tardiness of each order in terms of its completion time and its due date. The fact that the u_{ijk} variables are binary and the non-negativity of the completion time variables are given in constraint (4.23).

$$\min \quad \sum_{j=1}^{n} w_j T_j \quad (4.14)$$

$$\gamma_{ik} \geq \gamma_{i,k-1} + \sum_{j=1}^{n} p_{ij} u_{ijk}, \ \forall\, i, \ \ k = 2, \ldots, n, \quad (4.15)$$

$$\gamma_{ik} \geq \sum_{j=1}^{n} (p_{ij} + r_{ij}) u_{ijk}, \ \forall\, i, k \quad (4.16)$$

$$C_{ij} \geq \gamma_{ik} - \mathcal{M}(1 - u_{ijk}), \ \forall\, i, j, k \quad (4.17)$$

$$C_{ij} \leq \gamma_{ik} + \mathcal{M}(1 - u_{ijk}), \ \forall\, i, j, k \quad (4.18)$$

$$C_j \geq C_{ij}, \ \forall\, i,j \tag{4.19}$$

$$\sum_{k=1}^{n} u_{ijk} = 1, \ \forall\, i,j \tag{4.20}$$

$$\sum_{j=1}^{n} u_{ijk} = 1, \ \forall\, i,k \tag{4.21}$$

$$T_j \geq C_j - d_j, \ T_j \geq 0, \ \forall\, j, \tag{4.22}$$

$$C_j \geq 0, \ C_{ij} \geq 0, \ u_{ijk} \in \{0,1\}, \ \gamma_{ik} \geq 0, \ \forall\, i,j,k. \tag{4.23}$$

Constraints (4.24)–(4.26) are necessary for all $z \in A_i$, $i = 1, 2$, and all $z \in B$. The first of these states that the level of component z inventory at the *end* of replenishment period 0 (i.e., at time $\lceil \frac{H}{R} \rceil$), denoted I_{0z}, is equal to the number of components available at the beginning of the time horizon, $\lceil \frac{Q_z}{R} \rceil$, minus the amount used during that period, $used_{0z}$. Equation (4.25) states that, at the end of the lth replenishment period, $l = 1, \ldots, R-1$, the inventory level, I_{lz}, has to be equal to the amount that has been left over from the previous period plus the amount that has become available at the beginning of that period, $\lceil \frac{Q_z - l}{R} \rceil$, minus the amount used during that period, $used_{lz}$. Constraint (4.26) ensures that all components are used by the end of the last replenishment period, i.e., time $timeHorizon$.

Let δ_{ijl} be 1 if a_{ij} is started in the lth replenishment period, and 0 otherwise. Expression (4.27) states that if activity a_{ij} starts in replenishment period l, then its start time occurs at or after the beginning of this period. Equation (4.28) ensures that a_{ij} is assigned to exactly one replenishment period. Constraint (4.29), valid for all $z \in A_i$, $i = 1, 2$, and constraint (4.30), valid for all $z \in B$, state the relationship between $used_{lz}$ and δ_{ijl}. The non-negativity of I_{lz} and $used_{lz}$ variables for all replenishment periods l and all components z is given in constraint (4.31), together with the requirement for δ_{ijl} to be binary.

$$I_{0z} = \left\lceil \frac{Q_z}{R} \right\rceil - used_{0z}, \ \forall\, z, \tag{4.24}$$

$$I_{lz} = I_{(l-1),z} - used_{lz} + \left\lceil \frac{Q_z - l}{R} \right\rceil, \ \ l = 1, \ldots, R-1, \ \forall\, z, \tag{4.25}$$

$$I_{R-1,z} = 0, \ \forall\, z, \tag{4.26}$$

$$C_{ij} - p_{ij} \geq \sum_{l=0}^{R-1} \delta_{ijl} \left\lceil \frac{lH}{R} \right\rceil, \quad \forall\, i,j \tag{4.27}$$

$$\sum_{l=0}^{R-1} \delta_{ijl} = 1, \quad \forall\, i,j \tag{4.28}$$

$$used_{lz} = \sum_{j=1}^{n} \delta_{ijl} \alpha_{jz}, \quad \forall\, l, \ \ z \in A_i, \forall\, i \tag{4.29}$$

$$used_{lz} = \sum_{j=1}^{n}\sum_{i=1}^{2}\delta_{ijl}\beta_{ijz}, \quad \forall\, l,\ \ z \in B \tag{4.30}$$

$$I_{lz} \geq 0,\ used_{lz} \geq 0, \delta_{ijl} \in \{0,1\},\ \forall z \in \{A_1 \cup A_2 \cup B\}, i, j, l \tag{4.31}$$

4.3.3 Kolisch & Hess Model

Kolisch and Hess (2000) propose a MIP model for a generalization of our problem. We present this model, specialized to $PD2|\ r_{ij}, components|\ \sum_{j=1}^{n} w_j T_j$, using their original notation.

Let A be the set of orders to be processed, with each order consisting of three operations: the first operation corresponds to processing on the first machine, the second operation corresponds to processing on the second machine, while the third operation is performed on the assembly machine. Let J denote the total number of operations, that is, $J = 3|A|$. For each assembly operation a, there is a set of predecessor operations, P_a, which consists of all operations that need to be performed before assembly can be started. For our problem with two first stage machines, the set of predecessors for each assembly activity a consists of the operations performed on machines 1 and 2.

There is a set of binary decision variables x_{jt}, $j = 1, \ldots, J$, such that $x_{jt} = 1$ if operation j is started at time t, and 0 otherwise. To reduce the number of x_{jt} variables, Kolisch and Hess (2000) first apply forward and backward recursion to compute the earliest start times, ES_j, and the latest start times, LS_j, for all operations. In our implementation, for non-assembly operations, ES_j is set to the release date of the corresponding activity, and LS_j is set to $timeHorizon$ minus the activity's processing time. For assembly operations, ES_j is the maximum of the earliest completion times of the predecessors, and $LS_j = timeHorizon$. The x_{jt} variables can be non-zero only for $t = ES_j, ES_j + 1, \ldots, LS_j$. The variable T_a denotes the tardiness of order a. c_{jr} is a parameter that states the resource requirement of operation j. Specifically, $c_{jr} = 1$ if operation j requires machine r, $r = 1, 2$, and 0 otherwise. Each operation j requires q_{ji} units of part type i, $i = 1, 2, \ldots, I$. The total number of parts which become available by time t is denoted N_{it}. The full MIP model is given below in expressions (4.32)–(4.39), and is further referred to as *KolischModel*. Since we assume that the assembly operation takes negligible time, there is no processing time quantity in constraint (4.36), unlike in Kolisch and Hess' paper.

$$\min \quad \sum_{a=1}^{A} w_a T_a \tag{4.32}$$

$$\sum_{t=ES_j}^{LS_j} x_{jt} = 1, \quad j = 1, 2, \ldots, J \tag{4.33}$$

$$\sum_{t=ES_h}^{LS_h} (t + p_h) x_{ht} \leq \sum_{t=ES_a}^{LS_a} t x_{at}, \quad a = 3, 6, \ldots, J, \quad h \in P_a \tag{4.34}$$

$$\sum_{j=1}^{J} \sum_{\tau=\max\{0, t-p_j+1\}}^{t} c_{jr} x_{j\tau} \leq 1, \quad r = 1, 2, \quad t = 0, 1, \ldots, T \tag{4.35}$$

$$\sum_{t=ES_a}^{LS_a} t x_{at} - d_a \leq T_a, \quad a = 3, 6, \ldots, J \tag{4.36}$$

$$\sum_{\tau=0}^{t} \sum_{j=1}^{J} q_{ji} x_{j\tau} \leq N_{it}, \quad i = 1, \ldots, I, \quad t = 0, \ldots, T \tag{4.37}$$

$$T_a \geq 0, \quad a = 3, 6, \ldots, J \tag{4.38}$$

$$x_{jt} \in \{0, 1\}, \qquad j = 1, 2, \ldots, J, \quad t = ES_j, \ldots, LS_j \tag{4.39}$$

4.4 Constraint Programming Model

4.4.1 Background

Constraint programming is a methodology developed in the artificial intelligence and computer science communities that has been shown to be effective for a variety of scheduling problems (Beck et al., 1998).

Constraint programming provides a flexible modelling language with no restrictions on the type of variables and constraints, and in which problem sub-structures can be represented using global constraints with accompanying efficient domain-reduction algorithms. Domain-reduction, or *propagation*, algorithms remove values from domains of variables that are guaranteed to not be part of a solution to the problem. The notion of a global constraint is best illustrated using an example: suppose we are given a set of integer variables, $\{x_1, x_2, \ldots, x_n\}$, such that each x_i can take values from domain D_i, and we need to ensure that the values assigned to the variables are all distinct. While in mathematical programming such a requirement would need to be represented by a clique of not-equals constraints, in constraint programming, one simply needs to use the global constraint *AllDifferent*. During the solution process, the

presence of *AllDifferent*$(x_1, \ldots, x_n)$ in the model would invoke an efficient inference algorithm based on the idea of finding a maximal matching in a bipartite graph (van Hoeve and Katriel, 2006). The algorithm would remove values from each D_i that cannot participate in any solution to the problem.

In order to solve a problem, constraint programming combines inference with systematic tree search. Specifically, search is performed by assigning a value to a particular variable and by creating branches when there are several values in the variable's domain. The order in which variables and values are considered plays an important role in the efficiency of the search. Every time a variable is assigned a value, inference techniques, such as the one mentioned above for the *AllDifferent* constraint, can be employed to reduce the domains of the variables and hence speed up the search (Kumar, 1992).

4.4.2 Model

A constraint programming model for the $PD2|r_{ij}, components|\sum_{j=1}^{n} w_j T_j$ problem is stated in Equations (4.40)–(4.51). The model is implemented using IBM ILOG Scheduler 6.7, a C++ library based on IBM ILOG Solver offering features specifically for modelling and solving of scheduling and resource allocation problems (Scheduler, 2009).

In our model, two types of resources are used: there is a unary resource for each of the first-stage machines, and a *reservoir* for each component type. *Reservoirs* are resources with maximal and minimal capacity levels that can be replenished and consumed by activities (Scheduler, 2009). We set the level of propagation to the highest level possible (*IloExtended*) for all resources, implying the use of *disjunctive*, *balance* and *precedence graph* constraints for the unary machines, and the use of *timetable* and *balance* global constraints for the reservoirs. The constraint *disjunctive*(a_{ij}, i) represents a unary-capacity single-machine scheduling problem; that is, it ensures that the start times assigned to activities a_{ij} on machine i are such that the activities do not overlap at any point in time. The reader is referred to pages 36–39 of the Scheduler Reference Manual (Scheduler, 2009) for details regarding the rest of the global constraints.

The objective (4.40) is exactly the same as in the MIP models. Equation (4.41) defines C_{ij} as the completion time of activity a_{ij}. The next constraint makes sure that a_{ij} starts after its release date. Constraints (4.43) and (4.44) define the completion time and the tardiness of the order, respectively. Constraint (4.45) states that each machine is a unary capacity resource and invokes the corresponding global constraints. In (4.46), we state that each activity has to be processed on the corresponding unary machine for p_{ij} units of time.

$$\min \quad \sum_{j=1}^{n} w_j T_j \tag{4.40}$$

$$C_{ij} = a_{ij}.\text{end}(), \ \ \forall\, i, j \tag{4.41}$$

$$C_{ij} - p_{ij} \geq r_{ij}, \ \ \forall\, i, j \tag{4.42}$$

$$C_j \geq C_{ij}, \ \ \forall\, i, j \tag{4.43}$$

$$T_j \geq C_j - d_j, \ \ T_j \geq 0, \ \ \forall\, j \tag{4.44}$$

$$machine_i = \text{unaryResource}, \ \ \forall\, i \tag{4.45}$$

$$a_{ij}.\text{requires}(machine_i, p_{ij}), \ \ \forall\, i, j \tag{4.46}$$

$$res_z = \text{reservoir}(Q_z, \left\lceil \frac{Q_z}{R} \right\rceil), \ \ \forall\, z \tag{4.47}$$

$$repl_{lz}.\text{startsAt}(\left\lceil \frac{lH}{R} \right\rceil), \ \ \forall\, z, \ l = 1, \ldots, R-1 \tag{4.48}$$

$$repl_{lz}.\text{produces}(res_z, \left\lceil \frac{Q_z - l}{R} \right\rceil), \ \ \forall\, z, \ l = 1, \ldots, R-1 \tag{4.49}$$

$$a_{ij}.\text{consumes}(res_z, \alpha_{jz}), \ \ \forall\, z \in A_i, \forall\, i, j \tag{4.50}$$

$$a_{ij}.\text{consumes}(res_z, \beta_{ijz}), \ \ \forall\, z \in B, \forall\, i, j \tag{4.51}$$

Equations (4.47)–(4.51) represent the availability and usage of component inventory. Constraint (4.47) defines res_z as the reservoir of components of type z with a capacity of Q_z and an initial amount of $\lceil \frac{Q_z}{R} \rceil$. Recall that a *timetable* and a *balance* constraint are associated with each reservoir in the internal Scheduler representation. In Equation (4.48), we define a set of replenishment activities, $repl_{lz}$ for each z and l, that start at the corresponding replenishment times. Constraint (4.49) states that each replenishment activity $repl_{lz}$ produces $\left\lceil \frac{Q_z - l}{R} \right\rceil$ components to fill the corresponding reservoir, res_z. Equations (4.50) and (4.51) state that each activity a_{ij} consumes α_{jz} components from res_z if z is a machine-specific component, and β_{ijz} components from res_z if z is a shared component.

4.5 Experimental Results

Our experimental setup is based on ideas from papers by Keha et al. (2009) and Akturk and Ozdemir (2000), both of which address the single machine total weighted tardiness problem. We run five types of experiments, each consisting of two problem sets as in the paper by Keha et al. (2009). In problem set 1, processing times on each machine are taken from the discrete uniform distribution in the range between 1 and 10, i.e., $U[1, 10]$. In problem set 2, $p_{ij} \in U[1, 100]$ for experiments 1 to 3, but $p_{ij} \in U[1, 50]$ for experiments 4 and 5 in order

to avoid memory issues with the MIP models. The weights of the orders are drawn from $U[1, 10]$. The release dates and due dates are generated in a manner similar to that of Akturk and Ozdemir (2000). In particular, r_{ij} is taken from $U[0, \alpha \sum_j p_{ij}]$ with $\alpha \in \{0.5, 1.5\}$. To determine the due dates, the values of $slack_{ij}$, $i = 1, 2$, and $j = 1, 2, \ldots, n$, are obtained from $[0, \beta \sum_j p_{ij}]$, with $\beta \in \{0.05, 0.25, 0.5\}$. The due date d_j is then set to $\max\{d_{1j}, d_{2j}\}$, where $d_{ij} = slack_{ij} + p_{ij} + r_{ij}$. Component requirements are generated from $U[1, 10]$. Each of the five experimental settings aims to analyze the impact of a certain factor:

- number of orders (without component availability constraints) [Experiment 1],
- number of orders (with machine specific components) [Experiment 2],
- number of machine-specific components [Experiment 3],
- number of shared components [Experiment 4],
- number of inventory replenishments (for machine-specific and shared components) [Experiment 5].

A time limit of one hour was used in all experiments. The MIP models were implemented in CPLEX 12.1 with a tree memory limit of 900 MB and the *MemoryEmphasis* parameter turned on. These parameter settings were necessary due to the intensive memory requirements of the *timeIndexed* model and *KolischModel*.[5] The number of threads was set to one. The instances for which a particular MIP model ran out of memory are reported below as instances for which the model could not find a feasible solution.

The CP model was implemented in ILOG Solver/Scheduler 6.7, with the goal *IloSequenceForward(env) && IloSetTimesForward(env, totalWeightedTardiness, IloSelFirstActMinEndMin)*. This goal first sequences the activities on each machine and then assigns unique start times, beginning with the activity that has the minimal earliest start time among activities with minimal earliest end times (Scheduler, 2009). The objective value, represented in our implementation by the variable *totalWeightedTardiness*, is bound to its minimum consistent value at the end of the search.

The experiments were performed on a Dual Core AMD 270 CPU with 1 MB cache, 4 GB of main memory, running Red Hat Enterprise Linux 4. Below, we present a summary of our results, followed by the details of each experiment.

[5] Some runs of the *KolischModel* had to be stopped even before search started due to the size of the model.

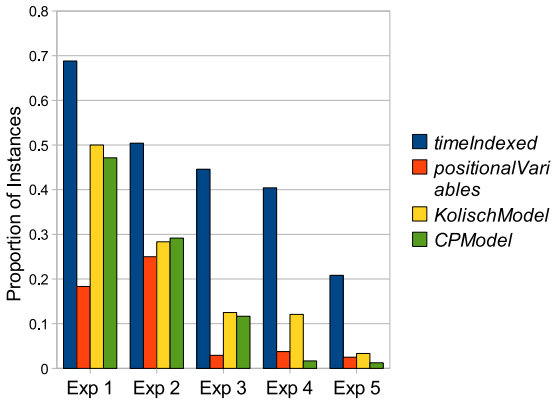

Figure 4.7: Proportion of Instances Solved to Optimality Within One Hour for Problem Set 1.

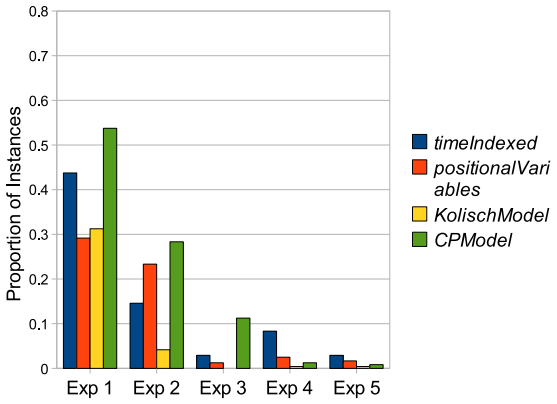

Figure 4.8: Proportion of Instances Solved to Optimality Within One Hour for Problem Set 2.

4.5.1 Summary of Results

Figures 4.7 and 4.8 present the proportion of instances from each experiment for which optimality was proved within one hour by each of the four models. The *timeIndexed* model is the best performer in all experiments for problem set 1 and also in experiments 4 and 5 for problem set 2. The *CPModel* outperforms the rest of the models in experiments 1 to 3 for problem set 2.

Figures 4.9 and 4.10 show the proportion of problem instances for which at least one feasible solution was found within one hour. We see that the *CPModel* finds a feasible solution in all instances. The *timeIndexed* model finds a feasible solution in 97% of instances for problem set 1 and in 86% of instances for problem set 2.

One important observation that can be made from the figures is that the magnitude of the processing times can have a significant effect on the performance of the models. For the *timeIndexed* model and *KolischModel*, longer processing times require a longer time horizon, and, consequently, more binary variables, resulting in a reduction in the number of times optimality is proved for problem set 2 of every experiment. For the *positionalVariables* model and *CPModel*, the magnitude of the processing times has no effect on the number of constraints or the number of variables. This fact explains the insensitivity of the *positionalVariables* model's performance to the change from problem set 1 to 2. For the *CPModel*, a wider range of processing times, as in problem set 2, implies that each problem instance contains both short and long jobs, allowing the *CPModel* to make stronger inferences regarding the jobs' positions in the schedules in experiment 1 to 3. This effect is not seen in experiments 4 and 5 due to the addition of shared component constraints, which deteriorate *CPModel*'s performance significantly.

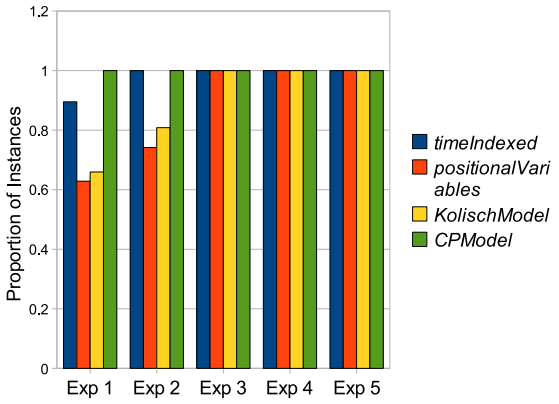

Figure 4.9: Proportion of Instances for Which At Least One Feasible Solution Was Found Within One Hour for Problem Set 1.

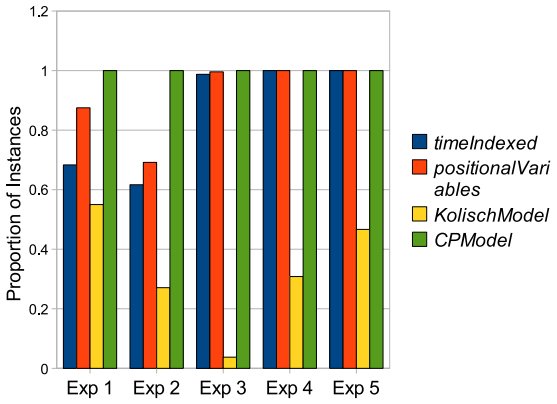

Figure 4.10: Proportion of Instances for Which At Least One Feasible Solution Was Found Within One Hour for Problem Set 2.

More detailed results, presented below, show that the most important parameter affecting performance (other than the range of processing times) is the number of orders. Specifically, for all MIP models, the number of times optimality is proved and the number of times at least one feasible solution is found decrease as n increases. The number of times the *CPModel* finds at least one feasible solution stays constant as n increases, though the number of times optimality is proved also decreases.

Overall, our results highlight the performance of two models. The *timeIndexed* model appears to be the best in terms of proving optimality when the processing times are small. The *CPModel* achieves better performance when the range of processing times is increased, provided there are no components shared between machines. The *CPModel* is the best in terms of finding a feasible solution within one hour, with the *timeIndexed* model being a close second.

Our experiments allowed us to evaluate the performance of the models with respect to two fundamental performance metrics, the number of feasible and optimal solutions found within a time limit. However, this experimental data does not provide a full understanding of the performance of the models and their implementations. Firstly, we do not experiment with varying the emphasis of CPLEX (i.e., the *MIPEmphasis* parameter) from optimality to feasibility. Clearly, having a feasibility emphasis should increase the number of times the MIP models find feasible solutions, but at the same time is likely to decrease the number of solutions proved optimal within the one hour time limit. In future work, it would be interesting to investigate the impact of the solvers' parameters on the relative performance of our models. Secondly, there are simpler ways of finding feasible solutions: all jobs can be scheduled after the last replenishment point or, given a sequence for the jobs, a schedule can be constructed by

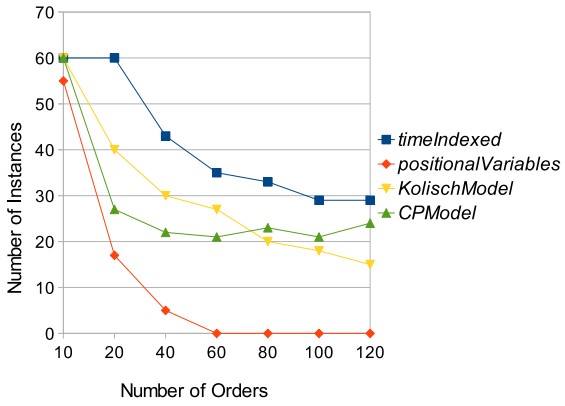

Figure 4.11: Number of Instances Solved to Optimality Within One Hour for Experiment 1 Problem Set 1.

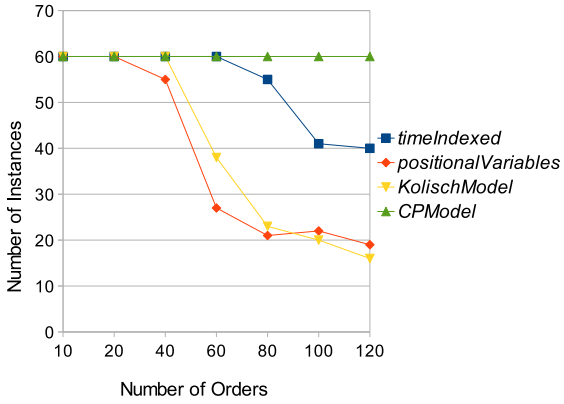

Figure 4.12: Number of Times At Least One Feasible Solution Was Found Within One Hour for Experiment 1 Problem Set 1.

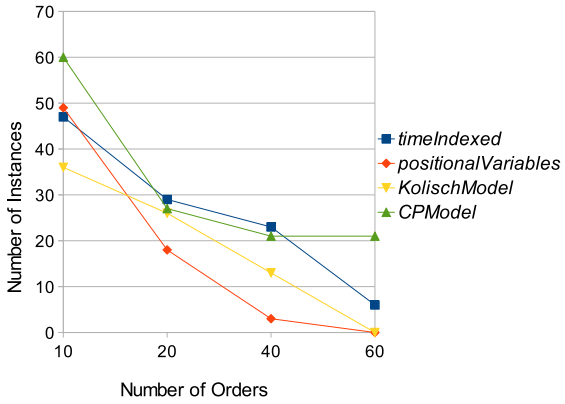

Figure 4.13: Number of Instances Solved to Optimality Within One Hour for Experiment 1 Problem Set 2.

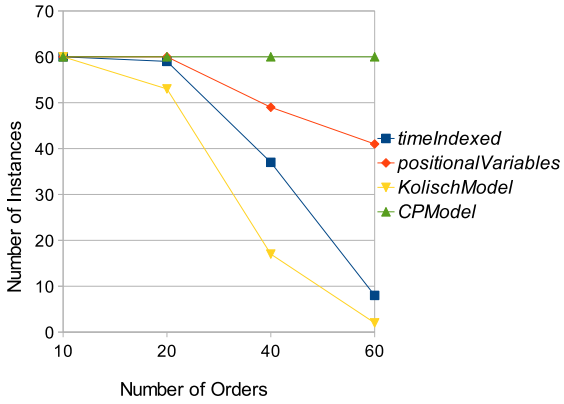

Figure 4.14: Number of Times At Least One Feasible Solution Was Found Within One Hour for Experiment 1 Problem Set 2.

following this sequence and starting the activities as soon as a sufficient number of components is available.[6] Thirdly, more insights about the models can be gained by examining the quality of the feasible solutions found.

4.5.2 Experiment 1

We vary the number of orders in the problem with release dates but without inventory constraints. Ten instances are considered for every combination of α and β for $n \in \{10, 20, 40, 60, 80, 100, 120\}$ in problem set 1 and for $n \in \{10, 20, 40, 60\}$ in problem set 2. Problem set 1 consists of 420 instances: there are two values for α, three values for β, seven values for n, and ten instances for each combination. Problem set 2 consists of 240 instances due to a smaller set of n values.

Problem Set 1 Figure 4.11 shows the number of instances for which the optimal solution was found and proved within one hour by each of the four methods for problem set 1 (i.e., with $p_{ij} \in U[1, 10]$). It can be seen that the *timeIndexed* model is the best performer for all values of n. The *KolischModel* comes in at second place for instances with less than 80 orders, but is outperformed by the *CPModel* when n reaches 80. The *positionalVariables* model is not as good as the others, and cannot prove optimality for any instances with 60 or more orders.

Figure 4.12 presents the number of instances for which at least one feasible solution was found by each model within the one hour time limit. In this respect, the *CPModel* is the best – it finds a feasible solution in all problem instances. The performance of the *timeIndexed* model deteriorates after n of 60. The greatest gap in performance between the MIP models and the *CPModel* occurs at $n = 100$ and $n = 120$.

Problem Set 2 Figure 4.13 shows that the *CPModel* performs better than the *timeIndexed* model for n of 10 and 60. For n of 20 and 40, the *timeIndexed* model is slightly better. Figure 4.14 demonstrates the superiority of the *CPModel* in terms of finding feasible solutions. The *positionalVariables* model is the best of the MIP models for finding feasible solutions for this problem set.

For both proving optimality and finding feasible solutions, the performance of the MIP models deteriorates with n, and the rate of deterioration is greater with larger processing times. The number of times optimality is proved by the *CPModel* decreases from $n = 10$ to $n = 40$, but remains fairly constant thereafter. The *CPModel* also consistently finds at least one feasible

[6] Thank you to Tony T. Tran and Dr. Michael Trick for pointing out this property and these suggestions.

solution for all problem instances, regardless of n.

4.5.3 Experiment 2

Experimental setup is essentially the same as in experiment 1, with $n \in \{10, 20, 40, 60\}$ (i.e., 240 instances) for each of the two problem sets and with one machine-specific component per machine and two replenishments.

Problem Set 1 The results of experiments with instances where $p_{ij} \in [1, 10]$ are presented in Figures 4.15 and 4.16. As in experiment 1, the *timeIndexed* model dominates the others in terms of proving optimality. The *CPModel* and the *timeIndexed* model are the best in terms of finding feasible solutions.

Problem Set 2 Figures 4.17 and 4.18 summarize the results of this experiment. The *CPModel* is the best in terms of proving optimality when n is 10 or 20. None of the models are able to prove optimality for higher n. The *CPModel* is also best for finding feasible solutions.

The *timeIndexed* model should be the method of choice for this variation of the problem when the range of processing times is small, while the *CPModel* is recommended when the range of processing times is large.

4.5.4 Experiment 3

In experiment 3 we vary the number of machine-specific components while keeping the number of orders at 20 for problems with release dates but without shared components. Specifically, there are one, three, five or seven components shared on each machine ($A_i \in \{1, 3, 5, 7\} \ \ \forall i$). The number of replenishments is set to two. For fixed α and β, there are 40 instances, where every four instances with one, three, five and seven components are correlated: a problem with three components has the same parameters as a problem with one, but with requirements for two more component types; a problem with five components per machine is exactly the same as a problem with three, but with requirements for two more component types; etc.

Problem Set 1 The number of times optimality is proved is presented in Figure 4.19. The *timeIndexed* model is the best performer. All models find a feasible solution in all problem instances, so no graph is presented for this statistic.

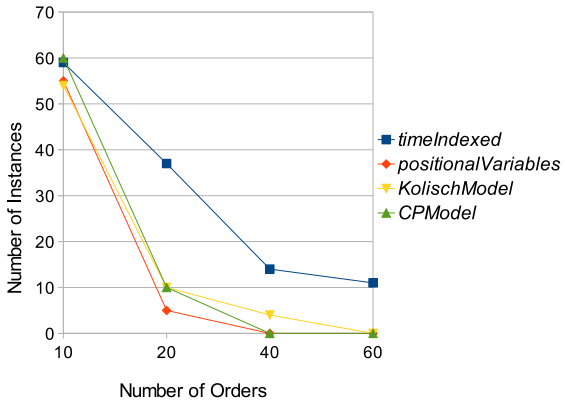

Figure 4.15: Number of Instances Solved to Optimality Within One Hour for Experiment 2 Problem Set 1.

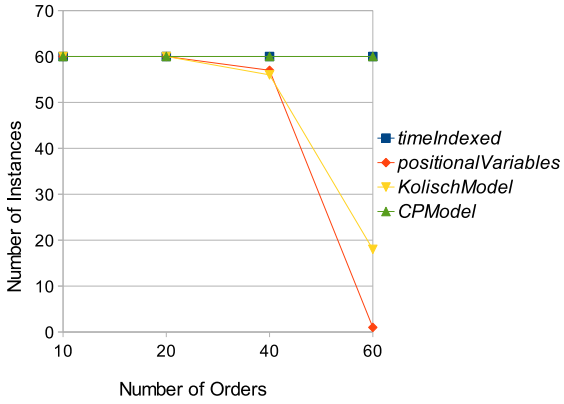

Figure 4.16: Number of Times At Least One Feasible Solution Was Found Within One Hour for Experiment 2 Problem Set 1.

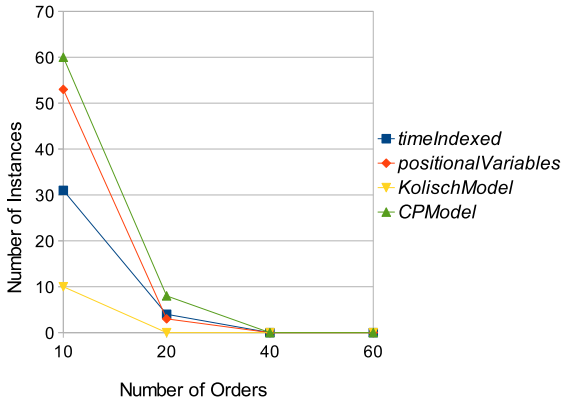

Figure 4.17: Number of Instances Solved to Optimality Within One Hour for Experiment 2 Problem Set 2.

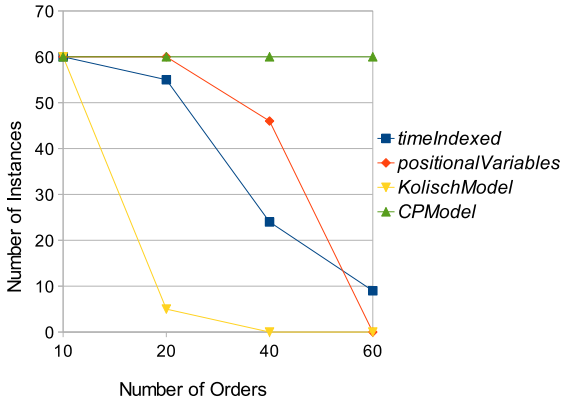

Figure 4.18: Number of Times At Least One Feasible Solution Was Found Within One Hour for Experiment 2 Problem Set 2.

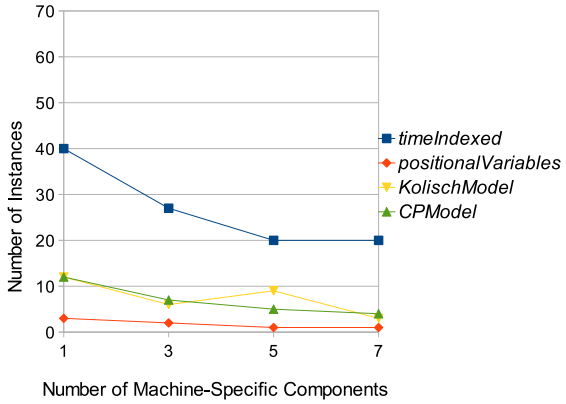

Figure 4.19: Number of Instances Solved to Optimality Within One Hour for Experiment 3 Problem Set 1.

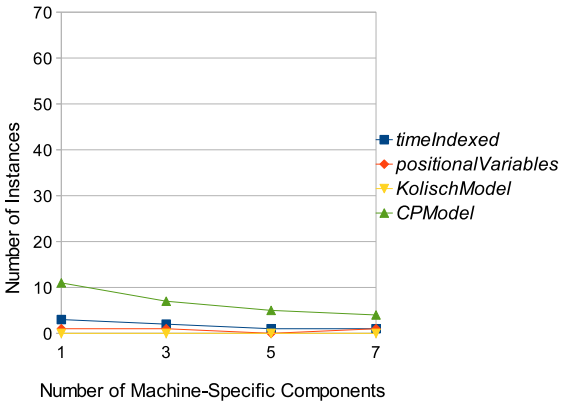

Figure 4.20: Number of Instances Solved to Optimality Within One Hour for Experiment 3 Problem Set 2.

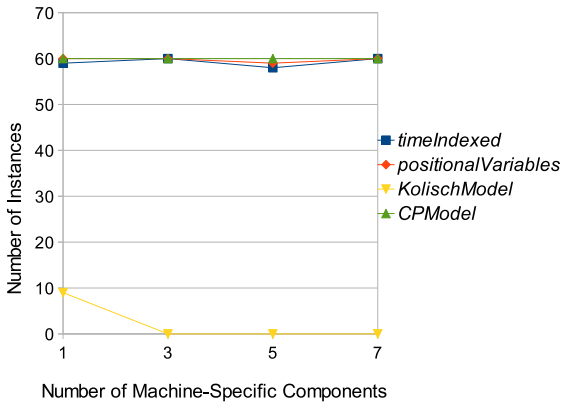

Figure 4.21: Number of Times At Least One Feasible Solution Was Found Within One Hour for Experiment 3 Problem Set 2.

Problem Set 2 The graphs for this experiment are presented as Figures 4.20 and 4.21. The *CPModel* appears to be the best performer. However, none of the models achieve satisfactory results for proving optimality.

4.5.5 Experiment 4

In experiment 4 we vary the number of shared components while keeping the number of orders at 20 and the number of machine-specific components on each machine at one. The number of replenishments equals two. The number of components shared between the two machines, $|B|$, is one, three, five or seven. For each combination of α, β and $|B|$, 10 instances are generated. Thus, there are 240 instances for problem set 1 and 240 for problem set 2.

Problem Set 1 All four models find at least one feasible solution in all problem instances. Figure 4.22 shows the number of times that optimality was proved by each of the models. The *timeIndexed* model seems to be the best performer overall, as in all other experiments with $p_{ij} \in U[1, 10]$ and a small number of orders.

Problem Set 2 The results for problem set 2 are presented in Figures 4.23 and 4.24. For proving optimality, even the best model, *timeIndexed*, does not perform well. For feasibility, all models except the *KolischModel* achieve consistent performance, finding at least one solution in all cases. For *KolischModel*, increasing the number of shared components increases the number of constraints and makes the model too large to be solved efficiently.

4.5.6 Experiment 5

This experiment looks at the effect of increasing the number of replenishments that occur within the time horizon for a problem with both shared and machine-specific components. The problem instances have 20 orders, one component shared among the two machines, one component shared among the sub-assemblies on machine 1 and one component shared among the sub-assemblies on machine 2. We evaluate instances with two, six, eight and ten replenishments within the scheduling horizon, considering 60 instances for each of these values (10 for each α and β). There are 240 problems in each of the two problem sets.

Problem Set 1 In this experiment, all models find at least one feasible solution for all instances within the one-hour time limit. Figure 4.25 shows the number of times when the optimal solution was found and proved by the models. The figure shows the superiority of the *timeIndexed* model over the other ones. The *CPModel* is the second-best performer. As the number of

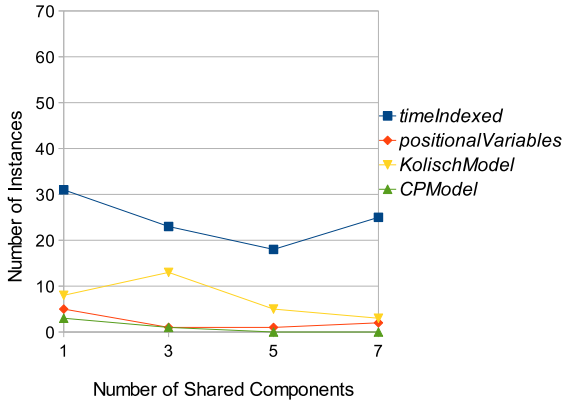

Figure 4.22: Number of Instances Solved to Optimality Within One Hour for Experiment 4 Problem Set 1.

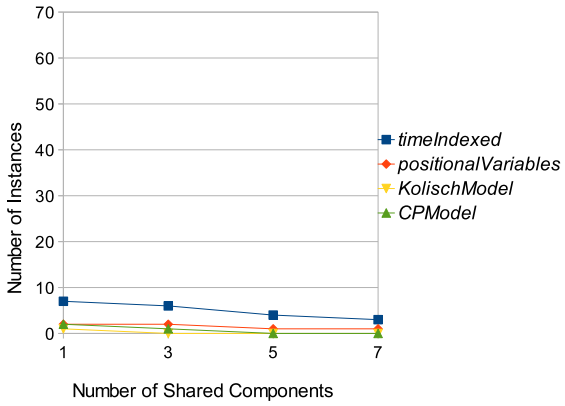

Figure 4.23: Number of Instances Solved to Optimality Within One Hour for Experiment 4 Problem Set 2.

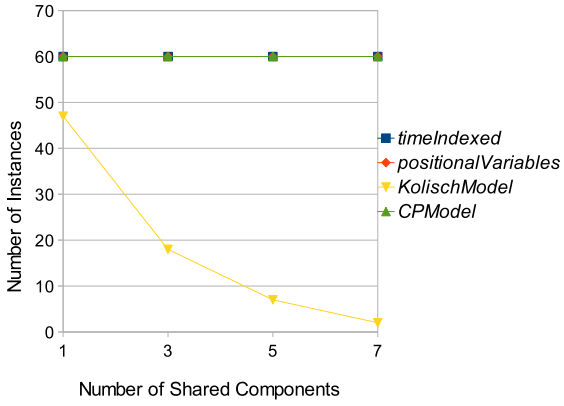

Figure 4.24: Number of Times At Least One Feasible Solution Was Found Within One Hour for Experiment 4 Problem Set 2.

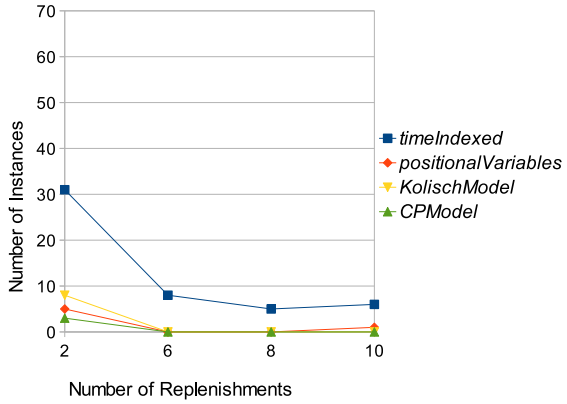

Figure 4.25: Number of Instances Solved to Optimality Within One Hour for Experiment 5 Problem Set 1.

replenishments increases from two to six, the number of instances in which the model proves optimality decreases. Further increases in the number of replenishments do not change the performance of the models by much.

Problem Set 2 The results of this experiment are presented in Figures 4.26 and 4.27. None of the models perform adequately in terms of proving optimality. All models except the *KolischModel* find at least one optimal solution.

4.5.7 Memory Consumption

For problem set 1, the *KolischModel* runs out of memory three times in experiment 2. Figure 4.28 shows the number of times each MIP model runs out of memory for problem set 2 of each experiment. Recall that the *MemoryEmphasis* parameter was turned on and that a tree memory limit of 900 megabytes was used. The statistics presented were calculated as the total number of instances minus the number of times optimality was proved minus the number of times the time limit was reached, and include instances where memory problems were due to the size of the model as well as due to the size of the search tree.

4.6 Conclusion

In this chapter, we considered a supply chain scheduling problem with the objective of minimizing the weighted tardiness of customer orders. Each customer order is composed of two sub-assemblies processed on dedicated machines, but joined by a common due date. Com-

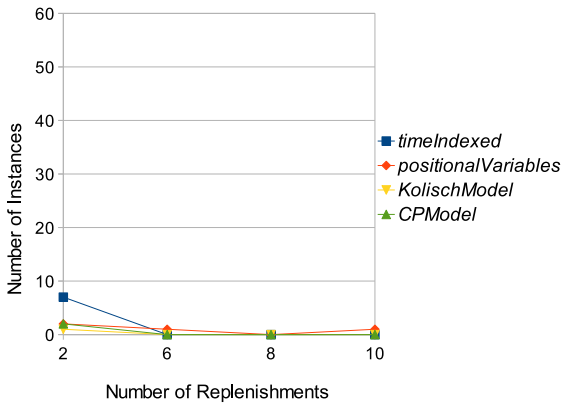

Figure 4.26: Number of Instances Solved to Optimality Within One Hour for Experiment 5 Problem Set 2.

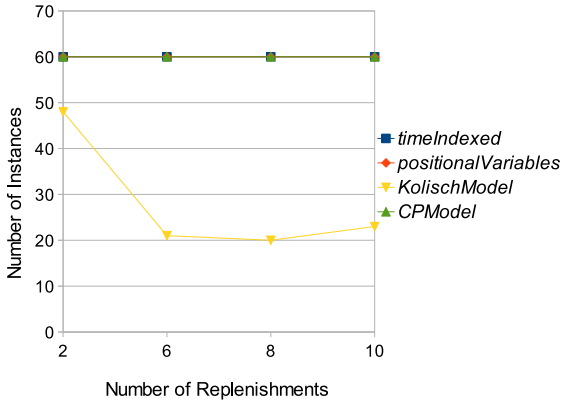

Figure 4.27: Number of Times At Least One Feasible Solution Was Found Within One Hour for Experiment 5 Problem Set 2.

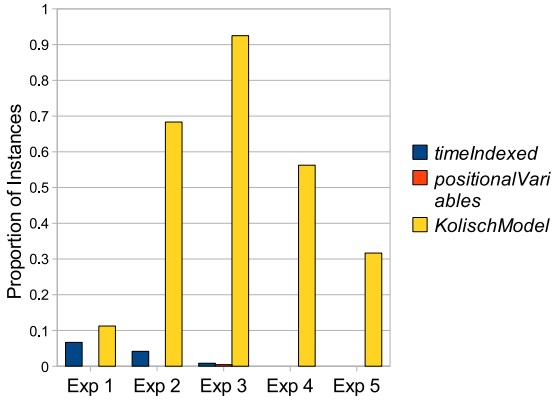

Figure 4.28: Proportion of Instances For Which the MIP Models Ran Out Of Memory for Problem Set 2.

ponent parts which are replenished periodically are necessary for sub-assembly fabrication. Three MIP models and a CP model for this problem were studied. We conducted extensive numerical experiments, which allowed us to compare the performance of the models with respect to finding an optimal or a feasible solution within a given time. The results show that a MIP model based on time-indexed binary variables is the best approach when the processing times are short, while the CP model performs better for problems in which the range of processing times is large. However, for problems with components shared between the two machines, none of the methods perform adequately, emphasizing the importance of developing more sophisticated algorithms. The main contributions of this chapter are the development and extensive empirical evaluation of complete methods for the assembly scheduling problem with inventory constraints, which is relevant for both supply chain and manufacturing settings.

Chapter 5

Scheduling and Inventory Management Future Work

Future work on the assembly scheduling problem with inventory constraints includes the development of better solution approaches as well as the exploration of various extensions. This problem also serves as a step toward a general framework to address joint inventory and scheduling decision-making.[1]

5.1 Better Solution Techniques

The experiments of Section 4.5 show that there is no one method that performs best over all variations of the assembly scheduling problem with inventory. For problems involving components shared between the machines, none of the methods are satisfactory. These results highlight the need for the development of more effective complete approaches and the investigation of heuristics for the $PD2|r_{ij}, components|\sum w_j T_j$ problem.

5.1.1 Complete Approaches

One way to develop better complete methods for the assembly scheduling problem with inventory constraints is to use decompositions, such as logic-based Benders decomposition (LBBD) (Hooker and Ottosson, 2003) and Lagrangian decomposition (Pirkwieser, 2006). In designing a decomposition approach, several issues have to be acknowledged. Firstly, we would ideally like to separate the problem of assigning components from each replenishment to activities on

[1] Parts of the work presented in this chapter have been published in a journal paper (Terekhov et al., 2012a). This material is used with permission of the copyright holder, Elsevier ©, and the paper's co-authors.

each machine from the problem of finding the start times of these activities. Secondly, minimizing the total weighted tardiness on one of the machines may not necessarily minimize the total weighted order tardiness, due to the common due date of sub-assemblies belonging to the same order. Additionally, separability of the sub-problem into smaller problems with strong intra-relationships and weak inter-relationships (Cadoli and Patrizi, 2009), which can lead to more effective approaches, is not trivial. For example, assigning activities to replenishment time periods and then independently solving scheduling problems for each such period is not possible: an activity should be allowed to start in one period and end in another. The idea of having sub-problems for independent time buckets seems more appropriate if the techniques proposed by Bock and Pinedo (2010) are employed. The approach taken by Coban and Hooker (2010) should also be investigated. A good starting point for the development of a Lagrangian decomposition method is the paper by Ahmadi et al. (2005), which addresses the total weighted completion time problem with no release dates.

Another line of future research is to consider alternative CP models. For example, the CP model used for the RCPSP by Berthold et al. (2010) may be adopted for our problem setting. One could also experiment with the use of constraint integer programming (Berthold et al., 2010).

5.1.2 Heuristic Approaches

It may be possible to extend the heuristics developed in previous assembly scheduling work (Lee et al., 1993; Potts et al., 1995) to the problem with components. Development of such methods would be especially attractive if worst-case performance guarantees, such as those in the paper by Potts et al. (1995), could be derived. The heuristics could also be used to find initial feasible solutions for the constraint programming model presented in Section 4.4. Moreover, the performance of tabu-search algorithms developed by Kolisch and Hess (2000) can be investigated.

5.2 Extensions

Numerous extensions to the assembly scheduling problem are possible, each of which would result in a more realistic, but also more complex, problem. These can be based on settings with more facilities, conceptual questions, and stochastic and dynamic variations of the problem.

5.2.1 More Complex Facility Structure

The methods presented in the previous chapter can be extended relatively easily to the case with more than two manufacturers, and to the case when processing at the MIT facility also requires sequencing. Similarly, one can combine scheduling at the manufacturing facilities with scheduling of a distribution centre, such as in the paper by Fanti et al. (2010). Given our experimental results, it is unlikely that the models proposed in this dissertation would be effective for these more complex problems; the use of decomposition methods and/or heuristics would be required instead.

As outlined in Chapter 2, in the real problem that inspired the study of Chapter 4, the manufacturing facilities are focused factories in which the production area is divided into sub-areas corresponding to different product families. Each sub-area is composed of several assembly lines. Therefore, instead of representing each manufacturer as a single machine, one may develop a more detailed model that takes into account the sequencing problems on each assembly line. Decomposition methods are likely to be needed in this case as well.

5.2.2 Conceptual Questions

The motivation for our study of the assembly scheduling problem came from a supply chain in which the manufacturing facilities act independently and there is no synchronization of production of sub-assemblies belonging to the same order. Therefore, on a conceptual level, it would be interesting to investigate the effect of the company switching from allowing the factories to schedule their operations independently to solving the centralized scheduling problem and imposing schedules on each of the facilities. Specifically, the following questions are of interest:

1. How significant would the improvement in whole order delivery rates (as represented, for example, by total weighted order tardiness) be?

2. Would the gains be significant enough to convince the company to switch from one structure to the other?

3. Would the gain from centrally-imposed schedules be similarly significant in other supply chain configurations?

Answers to these questions would also be helpful for evaluating the benefits of information sharing if the facilities are owned by different companies.

5.2.3 Stochastic Extensions

One can argue that our model of the supply chain scheduling problem is unrealistic because it is static and deterministic. Therefore, another direction for future work is to investigate dynamic and stochastic models.

With a fixed set of jobs, we can investigate models with stochastic processing times or models in which the quantities or timing of component replenishments are stochastic. Stochastic processing times can model variations in the processing requirements of different configurations of the same product type. The starting point for studying such problems could be the papers by Gourgand et al. (2005), Cai and Zhou (2004) and Righter (1997). Stochastic replenishment quantities and arrival times account for uncertainties in the delivery of components. The simplest problem with such assumptions is one which has stochastic activity release dates modelling the arrival time of unique components. To study this problem, one needs to consider the literature on static problems with stochastic release dates, discussed, for example, by Pinedo (2003). The dynamic version of the assembly scheduling problem is also of interest in future work, as mentioned in Section 11.2.3.

5.3 Integration of Scheduling and Inventory Management

In a supply chain where component parts and the sub-assemblies using these parts are produced by facilities belonging to the same company, or in an environment where components and subassemblies are processed at the same plant, an approach is necessary that takes into account the interdependence of inventory and scheduling decisions.

In the preceding chapter, we take an initial step toward developing a framework for integration of these decisions by studying a problem in which scheduling has to be optimized subject to constraints from a fixed periodic inventory policy. In that problem, replenishment timing and quantities are independent of the component inventory position at the manufacturing facilities. This independence allows us to decouple inventory and scheduling decisions, which would not have been possible if more common policies such (R, Q) and (s, S) were employed. For example, under an (R, Q) policy, we can make an order for Q items only when the inventory position reaches a level of R or lower; this implies that the schedule of jobs up to the replenishment time point would impact when the replenishment order would be placed.

One direction for future work is to study the more general optimization problem where at the higher (tactical) level of decision-making, one determines the timing and quantity of replenishments, whereas at the lower (operational) level, the jobs are scheduled. If the inventory policy leads to an infeasible scheduling problem at the lower level, then this information would

be communicated back to the higher level so that the inventory decisions could be adjusted. We note that determining the theoretically optimal inventory policy type is likely be intractable due to the complexity of the interactions between inventory and scheduling decisions; thus, we would instead like to focus on problems in which some assumptions about a policy type are made. In the full-cost approach,[2] the objective would be to optimize a combination of procurement, holding and weighted tardiness costs. Three variations of this problem could be considered:

- *Variation 1:* the replenishment policy type and its parameters are fixed (e.g., assume an (R, Q) policy with specific values for R and Q);
- *Variation 2:* the replenishment policy type is fixed, but the parameters of the policy (e.g., the values of R and Q) are decision variables of the problem;
- *Variation 3:* the replenishment policy type is not fixed, but some assumptions about it are made. For example, we could assume a periodic review structure, and the problem would be to optimize the timing and quantity of replenishments without being restricted by reorder point and order-up-to level parameters such as R and S.

Alternatively, we could consider the partial cost optimization problem, whose goal would be to minimize the inventory costs subject to a constraint ensuring that all jobs are completed on time. As above, different types of assumptions can be made about the specification of the replenishment policy. Grigoriev et al. (2005) propose a similar direction for future work: to address the case where customer orders and raw material replenishments can be renegotiated, with each order having a tardiness cost and each replenishment having an earliness cost.

When ownership of the facilities is different, a multi-agent view of the problem, such as the one presented by Duan et al. (2012), has to be adopted. In fact, we can assume that suppliers and manufacturers negotiate regarding the timing and size of component replenishments. When the supplier proposes a replenishment policy, the manufacturers need to evaluate the quality of the proposal by solving their scheduling problem, which is exactly the problem we solve in the previous chapter.

5.4 Conclusion

The motivation for the work presented in Part II of this dissertation is the supply chain of Alcatel-Lucent. In reality, the scheduling problem arising in this supply chain is combinatorial, stochastic and dynamic, and involves both scheduling and inventory decisions. Due to the

[2] See the literature review of Section 3.2.

problem's complexity, we have, so far, addressed its static and deterministic version, focusing on the combinatorics and the impact of inventory constraints on scheduling. Various directions for future work, which include stochastic extensions and a framework for integration of inventory management and scheduling, have been proposed.

Part III of this dissertation deals with solving scheduling problems that are combinatorial and dynamic in nature, and uses developments of queueing theory and deterministic scheduling. In the final part of this dissertation, we discuss future work on more complex problems that would require the developments of both Part II and Part III.

Part III

Integrating Scheduling and Queueing Theory for Dynamic Scheduling Problems

Chapter 6

Combining Scheduling and Queueing Theory: A Literature Review

In Chapter 2, we stated that real scheduling problems are combinatorial, dynamic and uncertain, and related to other processes of their environment. In Part II of this dissertation, we addressed two of these characteristics, combinatorial complexity and dependence of scheduling on other decision-making processes, within a specific realistic supply chain problem. Now, we focus on problems that are both combinatorial and dynamic. As in Part II, addressing both of these characteristics requires integration of methodologies from two fields, in this case queueing theory and classical scheduling.

In the scheduling community, there has been an increasing interest in modelling and solving of scheduling problems in which new jobs arrive over time, machines break down, and some or all characteristics of jobs that have to be scheduled are not known with certainty at the time when decisions have to be made (Bidot et al., 2009; Ouelhadj and Petrovic, 2009; Aytug et al., 2005; Davenport and Beck, 2000; Suresh and Chaudhuri, 1993). The problem of scheduling in such a dynamic and stochastic environment consists of determining an approach that dictates, at every decision time point, how the available machine processing time is to be allocated among competing job requests with the goal of optimizing the performance of the system.[1] The techniques that have been developed by the scheduling community for problems of this kind are generally based on various combinations of predictive approaches for constructing a baseline schedule and reactive approaches for changing the schedule online when a disruption occurs. In operations research, it has long been recognized that queueing theory can provide a basis for examination of such problems as well (Conway et al., 1967). In addition, queueing theory frequently considers the same application areas as scheduling. For example, both have extensively studied manufacturing environments, as is demonstrated by the complementary

[1] This chapter assumes familiarity with fundamental scheduling notions presented in Section 3.1.1.

Scheduling	Queueing
A survey of dynamic scheduling in manufacturing systems (Ouelhadj and Petrovic, 2009)	*Queueing theory in manufacturing: A survey* (Govil and Fu, 1999)
Scheduling and control of flexible manufacturing systems: a critical review (Basnet and Mize, 1994)	*Flexible manufacturing systems: a review of analytical models* (Buzacott and Yao, 1986)
Job shop scheduling techniques in semiconductor manufacturing (Gupta and Sivakumar, 2006)	*Queueing theory for semiconductor manufacturing systems: A survey and open problems* (Shanthikumar et al., 2007)

Table 6.1: Pairs of complementary scheduling and queueing theory papers that focus on manufacturing.

papers listed in Table 6.1.

However, to our knowledge, only a few papers (e.g., those by Nazarathy and Weiss (2010), Bertsimas and Sethuraman (2002) and Bertsimas et al. (2003)) combine queueing and scheduling ideas to address static scheduling problems, and only one recent dissertation (Tran, 2011) proposes methods for dynamic scheduling that take advantage of developments in both of these areas. It is true that scheduling books such as the one by Leung (2004) and Chrétienne et al. (1995) have chapters on both deterministic scheduling and queueing approaches, but they usually make no link between queueing theory and predictive-reactive scheduling.[2] Additionally, except for the work of Suresh and Chaudhuri (1993), literature surveys on scheduling in dynamic environments make no mention of queueing approaches. This characteristic of the scheduling literature may be partially due to the fact that a significant proportion of queueing research has dealt with the development of descriptive models for evaluation of the long-run expected behaviour of a system, while scheduling is prescriptive in nature and frequently considers only short-run performance measures. Nevertheless, there has also been a substantial amount of work on prescriptive queueing models that aim to provide a policy for stating which job or job class should be processed next.

In this chapter, we provide a foundation for the investigation of the integration of queueing theory and scheduling by surveying the relevant literature. Firstly, we provide an overview of queueing theory fundamentals and review the work done by researchers in queueing theory that is relevant to scheduling. The literature on scheduling in the context of queueing systems

[2]It is important to note that in early research on resource allocation and sequencing, queueing theory and scheduling were more unified. For example, the book *Theory of Scheduling* by Conway et al. (1967) contains many fundamental results for both deterministic scheduling problems and queueing problems.

is extensive, and so our review is by no means exhaustive: we omit many mathematical details and proofs, but attempt to provide a high-level view of the relevant material. Throughout the review, we make connections between queueing and scheduling. Secondly, we discuss methods for dynamic environments developed by the scheduling community. Finally, we describe our view on the relationship between queueing theory and scheduling.

6.1 Queueing Theory

We start by providing a brief introduction to queueing theory and general queueing models. We then survey the queueing literature on making scheduling decisions.

6.1.1 Queueing Theory Fundamentals

Queueing theory can be defined as the mathematical study of waiting lines (Gross and Harris, 1998). It therefore models systems in which one or more servers (machines) at one or more service stations process arriving customer requests (jobs).[3] As stated by Gross and Harris (1998), a mathematical model of a queue is composed of six main features.[4]

Arrival Pattern The main characteristic of the arrival pattern is the probability distribution of times between successive customer arrivals. This distribution may or may not be stationary (independent of time) and may or may not depend on the number of customers in the queue. Customers may arrive one-by-one or in a group (batch) whose size may be determined by a probability distribution (which could be deterministic). A customer may decide to join the queue and stay there until receiving service, join the queue but leave without service (renege) due to a long wait, or not join the queue at all (balk). If the system consists of more than one waiting line, a customer may decide which queue to join and/or to switch (jockey) from one queue to another.

Service Pattern Similarly, the service pattern is characterized by a service time distribution, which may be stationary or non-stationary and which may or may not depend on the number

[3]The terms *server* and *customer* are more common in the queueing literature, while the terms *machine* and *job* are more common in scheduling. In the subsequent description of the main features of a queue, we use the queueing terminology. However, for the rest of the review, we employ scheduling terms, except when the application clearly demands otherwise or when it is important to recognize that a server may be composed of multiple machines.

[4]Throughout the chapter, we take many of the fundamental queueing theory concepts and results from the book by Gross and Harris (1998), an excellent introduction to queueing theory. The reader is also referred to the textbook by Kleinrock (1976).

of customers waiting for service. Customers may be served singly or in groups whose size is determined by a probability distribution (which could be deterministic). The service time of a customer may become known with certainty upon arrival or at the end of service.

Queue Discipline The queue discipline is a rule that determines the order in which customers receive service. The simplest and most common queue discipline is first-come, first-served (*FCFS*). Other examples of queue disciplines include last-come, first-served (*LCFS*), random selection for service (*RSS*) or shortest processing time (*SPT*) first. The queue discipline is, essentially, a scheduling policy, and the reader may note the strong similarity between these examples of policies and dispatching rules in the scheduling literature (Pinedo, 2009).

System Capacity A queueing system may have finite or infinite capacity. In a finite capacity queue, there is a limit on the number of customers that can be present in the system at any point in time, and customers who arrive at times when this limit has been reached are not allowed to enter the system.

Number of Service Stations and Servers A queueing system may be composed of one or more stations (stages), each of which has one or more servers (channels). A multi-station system may be thought of as a network of interconnected nodes, with each node consisting of one or more queues with one or more servers. When there are several servers at a station, there may be a queue for each server, as in a supermarket, or one queue for all servers, as in the case of a bank. In a multi-station system, the order in which customers visit the stations (the routing) may be deterministic or stochastic. The distinction between single-station and multi-station models is discussed in more detail below.

Additional characteristics of queueing models include the server types (see, for example, the work on flexible servers by Ahn et al. (2002), Andradóttir et al. (2003), Gurvich and Whitt (2009)) and the customer types served by a system, as discussed in Section 6.1.1.2. Such characteristics affect the kinds of scheduling problems that need to be solved.

6.1.1.1 Single-Station vs. Multi-Station Models

In this section, we give some additional characteristics of single-station and multi-station queueing models and their relation to single-machine and multi-machine scheduling problems.

6.1.1.1.1 Single-Station Queueing Models Similarly to the $\alpha|\beta|\gamma$ system used to describe scheduling problems (Graham et al., 1979), a single-station queueing model is usually speci-

fied by a combination of symbols $A/B/X/Y/Z$, where A and B specify the inter-arrival and service time distributions, respectively, X is the number of parallel servers, Y indicates the capacity of the system and Z describes the queue discipline.[5] Typical entries for A and B are: D, which stands for deterministic; M, which stands for Markovian or exponential distribution; GI, which stands for a general independent distribution; or G, which corresponds to a general[6] distribution. X and Y are represented by positive integers or ∞. Examples for Z include *FCFS*, *LCFS*, *RSS*, priority (*PR*) or general discipline (*GD*). When the Y or Z fields are empty, infinite capacity and *FCFS* discipline are assumed, respectively. For example, an $M/G/3/5/LCFS$ queue is a queue with Markovian arrivals (M), a general service time distribution (G), three servers, a system capacity of five jobs, and last-come, first-served order of service.

Assuming equivalence between a server in queueing and a machine in scheduling, a single-station model corresponds either to a single-machine or a parallel-machine scheduling environment, depending on the number of servers. If there are multiple machines, having one queue in the system implies that any job can be processed on any machine, while having a distinct queue attached to each server models the situation when each job has to be processed on one specific machine.

6.1.1.1.2 Multi-station Queueing Models A queueing system with more than one service stage is typically referred to as a queueing network or a network of queues. It consists of a set of nodes, each of which is a (single-stage) queue. Jobs typically require service at several nodes (stations) of the network in some order (routing).

Queueing networks in which jobs enter the system "from the outside" and leave the system upon receiving service are referred to as open queueing networks. Models in which a finite population of jobs circulates within the system, and there are no arrivals from or departures to the outside, are called closed queueing networks. In semi-open networks, jobs are allowed to enter the network when the total number in the system is less than some threshold value (Chen and Yao, 2001). Open networks in which jobs can arrive at a node only from the outside or from an upstream node are referred to as acyclic or feed-forward networks. This kind of a network can therefore be analyzed in a recursive manner, starting from the upstream stations. In networks with feedback, on the contrary, jobs may visit the same station more than once (Chen and Yao, 2001). More complex networks may involve fork-join queues, in which an arriving job is split into sub-jobs that have to be processed in parallel and then reassembled (Bose, 2002; Baccelli et al., 1989). The work of Baccelli and Liu (1990) introduces synchronized

[5] See Section 3.1.1 for a description of the $\alpha|\beta|\gamma$ notation for scheduling problems.

[6] For a general distribution, there are no assumptions regarding its precise form. Thus, results that apply to the general distribution apply to any specific distribution.

queueing networks, which can model multiprocessor systems that process programs consisting of multiple tasks related by precedence constraints.

It has long been recognized that queueing networks are good models for dynamic job shop environments (Jackson, 1963; Wein and Chevalier, 1992; Buzacott and Shanthikumar, 1985, 1993). Semi-open and open network models provide the most intuitive representations of dynamic job shops since they allow one to explicitly model the process of job arrivals. A semi-open network can be used to represent a job shop with a strict limit on the number of jobs in the shop, while an open network can model a system with no such limit. Closed queueing networks can be used to model job shops in which the total number of jobs remains constant, or, in other words, for which it is valid to assume that a new job arrives at the same instant as another job is completed and leaves. Examples of environments that can be modelled as closed queueing networks include production systems that have a constant work-in-process inventory or follow a one-for-one replenishment policy (base-stock control rule) (Chen and Yao, 2001). Alternatively, one can imagine that there is a fixed number of palettes circulating through the service stations of the system, that raw materials are placed on one such palette and that these raw materials are gradually transformed into finished products as they receive processing (Williams, 1996). Acyclic networks can represent flow shops while networks with fork-join queues can be used to model shops with assembly operations.[7] Synchronized queueing networks can be useful for machine scheduling with complex precedences among the activities, for the task management problem described by Myers et al. (2007) and for project scheduling.

A generalization of a queueing network is referred to as a stochastic network (Kelly et al., 1996) or a stochastic processing network (Harrison, 1996). A stochastic processing network is defined by a set of buffers, a set of activities and a set of processors. Buffers hold jobs that have arrived to the system and are awaiting processing. Each activity uses one or several processors in order to process one or several jobs (which may belong to different buffers). Scheduling in stochastic processing networks corresponds to the determination of the order of executing the different activities on the various available processors. Stochastic processing networks can be used to model more complex scheduling environments than those that can be modelled by multi-class queueing networks. For example, they can allow one to represent material handling and machine-operator interaction (Dai and Lin, 2005) or input-queued switches (Dai and Prabhakar, 2000). In the remainder of this literature review, we focus on queueing systems rather than stochastic processing networks.

[7] In Chapter 11 we mention the investigation of fork-join queues for modelling the stochastic and dynamic versions of the assembly setting studied in Chapter 4.

6.1.1.2 Single-class vs. Multi-class Systems

In the literature, a distinction is usually made between single-class and multi-class queueing networks. In a single-class network, jobs being processed or waiting for processing at any given station are assumed to be indistinguishable. A multi-class system is one in which several classes of jobs are served at each station (Harrison and Nguyen, 1993). A class is usually defined as a combination of job type and processing stage. An instance of a job belonging to a particular class is equivalent to an operation in scheduling terminology. The reader should note a slight subtlety arising from these definitions. Specifically, in a single-class network, according to the definition of a class, there are, in total, as many classes as there are stations (Harrison and Nguyen, 1993). The term *single*-class refers to the fact that *a single class is served at each station*.

It is important to note that the individual jobs belonging to the same class are different: they have unique arrival times, unique processing times, and, in the case of multiple machines, may have a unique routing. However, these differences between the jobs are simply stochastic variations – all jobs in a class are governed by the same stochastic processes, and are, except for these stochastic variations, indistinguishable.

In a single-class system, scheduling corresponds to determining the order in which the jobs should be processed at each node of the network. Since the jobs are stochastically indistinguishable, scheduling decisions have to be based on realizations of job characteristics. In the context of single-class systems, the queueing literature mostly focuses on the performance evaluation of scheduling policies such as *FCFS*, *LCFS*, shortest remaining processing time first, processor-sharing, etc. (Shanthikumar, 1982; Wolff, 1989; Harchol-Balter, 2011). In some cases, such analysis yields very strong results, such as the optimality of the shortest remaining processing time policy in an $M/G/1$ queue (Schrage, 1968; Smith, 1978).[8]

Scheduling in multi-class queueing networks involves the determination of a policy that specifies which class of jobs should be processed next on each machine, and this decision may be based on the state of the system (i.e., the total number of jobs present in the system or in each of the classes), or the characteristics of a particular job class. Scheduling of multi-class queueing networks falls under the category of models for control of multi-class queueing networks, which are known to be mathematically challenging (Bertsimas et al., 1994). For example, the

[8]Interestingly, for the single-machine case the scheduling literature has parallel results: for a static deterministic problem with n jobs, the shortest processing time first rule minimizes the total flow time; the weighted shortest processing time rule minimizes the total weighted flow time (equivalently, the sum of weighted completion times) (Pinedo, 2003; Baker and Trietsch, 2009). Similarly, Baker and Trietsch (2009) and Pinedo (2003) show that, for a problem with n jobs and stochastic processing times (i.e., the realization of the processing time for a given operation is not known until the activity is finished), the shortest (weighted) expected processing time rule is optimal for the (weighted) expected flow time objective.

choice of job to be processed next at every station of the network at every decision time point may depend not only on the number and characteristics of jobs present in the station's queue, but also on the state of the other nodes in the network.

6.1.1.3 Descriptive vs. Prescriptive Models

Queueing theory is composed of work on descriptive and prescriptive (control) models (Gross and Harris, 1998; Stidham Jr., 2002). The goal of descriptive queueing theory is to evaluate the performance of a queueing system based on some assumptions about its characteristics. Typical performance measures include (Adan and Resing, 2002):

- the distributions of the amount of time spent by jobs in the system (sojourn time) and of the jobs' waiting time in the queue prior to receiving service (queueing time),
- the distributions of the number of jobs in the system and in the queue,
- the distribution of the amount of work in the queue, with work being defined as the sum of the service times of the jobs waiting in the queue and the remaining (or residual) service time of the job(s) currently receiving service,
- the distribution of the length of the busy period of a server, which is the time period during which the server is continuously busy,
- cost-based measures such as expected holding costs (Reiman and Wein, 1998) or net present value of the difference between rewards and costs over an infinite time horizon (Harrison, 1975).

Most of descriptive queueing theory focuses on steady-state analysis of the system, that is, its performance over a long period of time during which the system behaviour should stabilize. Specifically, steady-state expected values of the above-mentioned distributions are essential for understanding the performance of the system. Transient analysis (Grassmann, 1977; Kaczynski et al., 2011) and evaluation of time-dependent probability distributions (Massey and Whitt, 1998) provide additional insights into system performance but are, in general, more difficult to derive and hence rarer.

The area of queueing theory that deals with prescriptive models is frequently called the *design and control of queueing systems* (Tadj and Choudhury, 2005; Gross and Harris, 1998). In both queueing design and control problems, the goal is to find optimal values for the *controllable* parameters of the queue. These parameters include: the number of machines (channels) available for processing arriving jobs, the limit on the length of the queue(s) (system capacity), the arrival rate of jobs to the queue(s), the service rates of the machine(s), as well as any

combination of these. Queueing design problems are static – once the optimal value of a controllable parameter is determined, it becomes a fixed characteristic of the queue. Queueing control problems, on the contrary, are dynamic – the goal in such problems is usually to determine an optimal action to take when the system is in a particular state. For example, consider a retail facility with workers who have to serve stochastically-arriving customers and also perform back room tasks which are independent of the customer arrival process (Terekhov et al., 2009). In order to optimize the performance of such a facility, one has to solve the queueing design problem of finding the optimal number of cross-trained servers to employ as well as the related queueing control problem of determining when to dynamically switch these workers between the two task types. We refer the reader to the papers of Tadj and Choudhury (2005) and Crabill et al. (1977), and the books by Kitaev and Rykov (1995) and Stidham (2009), for overviews of design and control problems involving queues, and to the papers on the subject that are cited in the beginning of Section 6.1.2.

The queueing discipline of each buffer determines the order in which arriving jobs are processed. The queueing discipline can, in fact, be seen as a scheduling policy. Consequently, both descriptive and prescriptive queueing models can be of use in scheduling. Descriptive models can be helpful for analyzing the performance and deriving theoretical properties of particular scheduling policies. Prescriptive models, on the contrary, allow one to determine good or optimal scheduling rules. Some authors (Crabill et al., 1977) classify such models as part of the queueing control literature.

6.1.2 Methodologies for Scheduling

Of main interest to us are models in queueing theory that can help with sequencing decisions. We survey these models in this section. We do not review the related issues of queueing control and design (Tadj and Choudhury, 2005; Crabill et al., 1977; Govil and Fu, 1999) such as admission control (control of arrival rates, decision to accept/reject an arriving job, appointment scheduling) (Stidham, 1985; Fan-Orzechowski and Feinberg, 2007; Hajek, 1984; Pegden and Rosenshine, 1990), routing control (Ephremides et al., 1980; Veatch and Wein, 1992; Gurvich and Whitt, 2009), server assignment (Ahn et al., 2002; Andradóttir et al., 2003) and control of service rates (Weber and Stidham Jr., 1987; Grassmann et al., 2001).

The reader should note, however, that both routing control and server assignment problems discussed in the queueing literature can be seen as representations of scheduling problems in dynamic parallel machine environments.[9] In server assignment models, a rule for assigning

[9]There has also been work on establishing a duality relationship between scheduling and routing in parallel queues (Sparaggis et al., 1993).

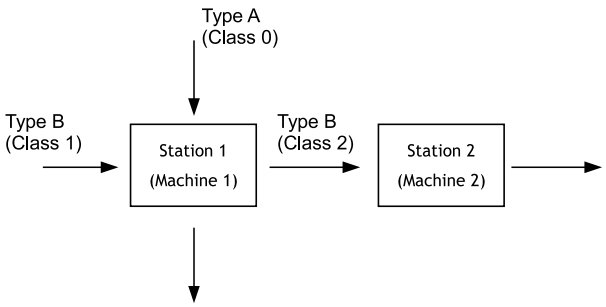

Figure 6.1: Illustrative example.

machines to job types needs to be determined and decisions are typically made when a machine becomes free; in routing models, on the contrary, machines have their own queues and every arriving job needs to be assigned to a machine (Righter, 1994). Investigating the relationship between such queueing design and control models and scheduling is worthwhile and interesting, but is outside the scope of this dissertation and thus is left for future work.

This section is aimed at providing the reader with an understanding of the main methodological streams in queueing theory that address scheduling problems. To achieve this goal, we classify queueing methodologies related to scheduling into three categories. Firstly, we discuss Markov Decision Processes (MDPs), which can provide the basis for proving theoretical properties of policies as well as for computation of policy parameters. Theoretically, the MDP approach does not need to place a-priori restrictions on the space of scheduling policies that is considered. However, when the complexity of the problem grows, MDP approaches fail due to the size of this space. In general, there are two ways to deal with such complexity: optimize within a restricted policy space or solve abstractions or approximations of the problem to obtain guidance for scheduling decisions in the original problem. These are the remaining two categories we discuss. Within each of the three parts of this section, we make connections with single-machine and multiple-machine scheduling problems and provide ideas for future work on the integration of queueing and scheduling. We conclude by describing our global view of the relationship between queueing theory and scheduling in Section 6.3.

For illustration purposes, we use a system that serves two types of jobs, as shown in Figure 6.1. Jobs of type A require processing at station 1 while jobs of type B need to be processed at station 1 and then at station 2. Both stations consist of exactly one machine. There are, therefore, three classes: class 0 corresponds to jobs of type A, class 1 to jobs of type B at machine 1 and class 2 to jobs of type B at machine 2. We assume that class 0 jobs arrive to the system with rate λ_0 and a general inter-arrival distribution; class 1 inter-arrival times also follow a general inter-arrival distribution, with rate λ_1. The arrival rate to class 2 depends on

the arrival and processing rates of class 0 and 1, and the scheduling policy employed at machine 1. Processing times for class i are generally-distributed with rate μ_i. This problem setting has been extensively studied in the queueing literature as it captures the key difficulties arising in dynamic scheduling and control (Chen et al., 1994). The reader is encouraged to think about the scheduling and resource allocation questions that may arise in this environment, and about how to address them using a predictive-reactive method or other dynamic scheduling methods surveyed in Section 6.2. In this section, we focus on the problem of sequencing type A and B jobs at machine 1 in order to minimize the discounted total cost over an infinite time horizon, assuming that per time unit holding costs of c_A and c_B are incurred for type A and B jobs, respectively.

6.1.2.1 Markov Decision Processes

An MDP model consists of states, actions, decision points, costs, transition probability distributions and an optimization criteria (Sennott, 1999).[10] In the scheduling literature, it is typical to assume that jobs belonging to the same class are distinguished from each other by their weights, processing times and due dates; schedules are constructed based on these individual characteristics. The MDP approach is general enough to represent scheduling problems under such assumptions, with actions corresponding to the choice of the specific job to process next, or the choice to remain idle. However, the state representation in the corresponding MDP model would need to include all of the known job characteristics and to keep track of the processing sequence, leading to an enormous state space that would make the problem intractable to solve.

The queueing representation of scheduling problems is much more amenable to analysis via MDP methods, since it assumes that jobs belonging to a particular class are stochastically indistinguishable and scheduling decisions amount to choosing the class of jobs to be processed next, rather than the individual job. It is usually implicit in queueing models that the job that arrived to the chosen class first is the job that will be processed first. Thus, a state can be defined as the number of jobs of each class present in the system; decision points can be job completion epochs or time points when there is a job arrival and the machine is idle; an action may be the choice to process a particular job class on a given machine or for this machine to

[10] Variants of MDP models are defined similarly. For example, semi-Markov decision processes (SMDPs) generalize MDPs by representing the evolution of the system in continuous time, with the time spent by the process in a particular state following an arbitrary probability distribution. The decision-maker is allowed or required to choose actions whenever the system state changes (Puterman, 1994). Additional examples include partially-observable MDPs, in which there is uncertainty about the state of the process but state information can be acquired (Monahan, 1982), and constrained MDPs, which have constraints on the cumulative costs incurred at any time (Yeow et al., 2006). See also the paper by Glasserman and Yao (1994), which discusses generalized SMDPs.

remain idle (Harrison, 1975). The goal is to find a policy that specifies the action to be taken in every state of the system so that an objective is optimized.

The policies may or may not depend on the history (past states) of the system, or on the actual time point when a decision is made. Derman (1970) classifies MDP policies in a hierarchical fashion. The most general is the class C of all possible policies, that is, policies which may be dependent on the complete history of the system. An important subclass is C_M, which consists of all memoryless, or Markovian, policies. In such policies, it is assumed that the probability of taking action a is a function of only the current state and the current time. C_S is a subclass of C_M which consists of time invariant policies. In other words, under these policies, the probabilities of taking a particular action are dependent only on the system state. A special subclass of C_S, C_D, consists of all deterministic policies. A deterministic policy is one in which the probability of taking a particular action a in a state i is either 0 or 1.

We now give a formal definition of an MDP with a countable state space $\mathbb{X}$ and action space $\mathcal{A}$ based on the paper by Chen and Meyn (1999). For each state $x \in \mathbb{X}$, there is a non-empty subset $\mathcal{A}(x) \subseteq \mathcal{A}$, which consists of actions that are admissible when the state at time t, denoted $\Phi(t)$, is x. Transitions in the state process Φ occur according to conditional probability distributions $\{P_a(x, y)\}$, which define the probability that the next state is $y \in \mathbb{X}$ given that the current state is $x \in \mathbb{X}$ and action $a \in \mathcal{A}$ is taken. A policy w can then be formally defined as a sequence of actions $\{a(t) : t \in \mathbb{Z}^+\}$, where $a(t)$ can depend only on the history of the process $\{\Phi(0), \Phi(1), \ldots, \Phi(t)\}$ and $\mathbb{Z}^+$ is the set of non-negative integers. A Markov policy is of the form $\mathbf{w} = \{w^0(\Phi(0)), w^1(\Phi(1)), w^2(\Phi(2)), \ldots\}$ where w^i, for each i, is a mapping from $\mathbb{X}$ to $\mathcal{A}$ and $w^i(x) \in \mathcal{A}(x)$ for each state x. A stationary policy is then a Markov policy with $w^i = w$ for all i and for some fixed w. Given a one-step cost function $c(\Phi(t), w(\Phi(t)))$ which states the cost associated with taking action $w(\Phi(t))$ in state $\Phi(t)$, one goal of the MDP may be to find a stationary policy $\mathbf{w}$ which minimizes the average expected cost

$$J(\mathbf{w}, x) := \limsup_{n\to\infty} \frac{1}{n} \sum_{t=0}^{n-1} \mathbf{E}_x[c(\Phi(t), w(\Phi(t)))]. \tag{6.1}$$

After a problem is modelled as an MDP, stochastic dynamic programming methods may be applied to either obtain numerically-optimal solutions (Sennott, 1999) or to characterize the structure of optimal policies (Stidham and Weber, 1993). For example, one common approach to finding the optimal policy is value iteration (Chen and Meyn, 1999), which is based on a finite-time problem with value function

$$V_n(x) = \min \mathbf{E}_x\Big[\sum_{t=0}^{n-1} c\Big(\Phi(t), a(t)\Big) + V_0\Big(\Phi(n)\Big)\Big]. \tag{6.2}$$

The reader is referred to the books by Bertsekas (2005, 2007) for an extensive coverage of topics related to the use of dynamic programming in optimal control problems, to the book on MDPs by Puterman (1994) and to the book by Meyn (2008), which provides an in-depth treatment of various topics related to scheduling in queueing systems, and talks about MDPs in detail in Chapter 9.

Stidham and Weber (1993) state that using MDPs and dynamic programming to solve for optimal policies in large problems (e.g., communication networks of realistic size) is usually intractable. However, determining the structure of optimal policies for small problems can help in the development of heuristic policies for large systems. Similarly, Glasserman and Yao (1994) state that computation of optimal controls for MDPs is generally infeasible without special structure, which motivates investigation of the *form* of optimal policies. For example, in a *switching curve* policy type, one action is optimal in the states below the curve, while another is optimal for the states above the curve (Glasserman and Yao, 1994).

For the two-station problem of Figure 6.1, the state at time t is $\Phi(t) = (Q_0(t), Q_1(t), Q_2(t))$, where $Q_i(t)$ is the number of class i jobs in the system at time t. The actions are: to process class 0 or to process class 1 at station 1, or to idle station 1. Assuming machine 2 is non-idling, from the queueing perspective there are no scheduling decisions to be made for class 2, since it is the only class served by machine 2; from the scheduling perspective, there is of course the question of sequencing jobs within this class. Chen et al. (1994) use value iteration as part of their proof of the existence of a stationary policy that is optimal for determining the actions at machine 1. This policy switches machine 1 from processing class 1 to processing class 0 (if there are jobs available in class 0) or to idling (if there are none) when the congestion level at machine 2 exceeds a threshold that is a function of the state at machine 1. Thus, the policy has a switching-curve structure, and reflects the intuition that as the congestion at station 2 grows, it becomes less and less appealing to process class 1 jobs.

As discussed above, an MDP formulation of the problem of deciding how to allocate resources to job classes can lead to both theoretical and numerical results. These results are useful in a variety of applications, e.g., in healthcare (Patrick et al., 2008; Zonderland et al., 2010). However, in applications where the processing time of a job can be well-estimated upon arrival or where operational-level decisions need to be made (Zonderland et al. (2010) mention the development of operational-level policies as future work), it is interesting to see whether combining MDP results with detailed within-class scheduling methods or with the predictive-reactive framework of Section 6.2 is useful.[11] In addition, MDP models may provide a meta-framework for the construction of queueing/scheduling hybrids: if a queueing algorithm and a

[11] In addition, in some cases using a pure queueing or MDP approach maybe intractable, as stated, for example, in the paper by Vermeulen et al. (2009), which proposes an adaptive method for scheduling of CT-scans.

scheduling algorithm are chosen, and the state of the problem is compactly represented, then we can define an action as the choice to follow the queueing algorithm or the choice to follow the scheduling algorithm until the next decision time point.

However, there is a fundamental difficulty with applying the MDP approach even in the context of scheduling classes of jobs. Specifically, when buffers are infinite, the corresponding optimization problem is infinite-dimensional; when the buffers are finite, the complexity of the problem grows exponentially with the state space dimension (Meyn, 2001). As a result, the rest of the queueing theory approaches for scheduling are based either on models with a specific structure, or models which approximate or aggregate system characteristics.

6.1.2.2 Models with Specific Structure

One approach to addressing the complexity of scheduling problems is to restrict the policy types that are considered. In this section, we discuss four major classes of models that adopt this approach: priority queues, polling systems, vacation models and bandit models.

6.1.2.2.1 Priority Queues The simplest models in queueing theory assume a *FCFS* discipline: jobs are processed in the order in which they arrive. A priority queueing model is different from a regular *FCFS* queue in two respects. Firstly, in a priority queueing model, arriving jobs are divided into classes or types of different priority. This notion of priority is analogous to that of job weights in scheduling: both are a measure of a job's importance. Since queueing models do not, in general, distinguish jobs based on individual characteristics, it is natural for these models to group jobs into priority classes; in scheduling, each job may be assigned a unique weight. Secondly, the system operates according to a "priority discipline", which is similar to a dispatching rule. In fact, they are based on the same idea: at a particular decision time point, be it the completion time or the arrival time of a job, one assigns an index to all jobs that are waiting to be processed and chooses the job with the greatest or smallest index as the one to be served next.

The queueing literature, just like the scheduling literature on dispatching rules, makes a distinction between preemptive and non-preemptive rules, and static and dynamic rules (Jaiswal, 1968).[12] For a system in which processing times become known upon arrival, an example of a priority queueing model with a static discipline is one in which a job class with index x is composed of all jobs that require between x and $x + dx$ units of processing time (Adan and Resing, 2002), and jobs are sequenced in non-decreasing order of their priority index, which does not change throughout each job's time in the system. For multi-class systems in which

[12] Jaiswal (1968) actually uses the terms *exogenous* and *endogenous* instead of *static* and *dynamic*, respectively. However, the terms *static* and *dynamic* are also used in queueing. See, for example, the paper by Goldberg (1977).

each job class k has weight/cost c_k and an expected processing time of $\frac{1}{\mu_k}$, a classic discipline is the $c\mu$ rule (equivalently, weighted shortest expected processing time rule). This rule schedules classes in non-increasing order of their $c_k\mu_k$ values, and uses *FCFS* within each class. Jaiswal (1982) provides several examples of dynamic priority rules from the literature. In one of these, the instantaneous priority index of a job in class k is $I_k(t) = c_k - W_k(t)$, where $W_k(t)$ is amount of time a job has been waiting in the system up to time t. More generally, $I_k(t)$ can be a concave function of the waiting time, $I_k(t) = \phi_k\big[W_k(t)\big]$.

In scheduling, a dispatching rule is generally viewed as a heuristic approach that can be applied to a variety of scheduling problems, regardless of whether they are deterministic or stochastic, static or dynamic. In the literature, the performance of dispatching rules is usually evaluated experimentally, although in some cases a dispatching rule can be shown to be optimal. In addition, there is significant interest in finding rules with tight worst case bounds on performance and polynomial running time, referred to as polynomial-time approximation schemes (PTAS) (Schuurman and Woeginger, 1999). An example of such a dispatching rule is the list scheduling heuristic proposed by Graham et al. (1979), in which, at a time point when a machine completes processing, the first available job from a specified priority list is scheduled. Following Schuurman and Woeginger (1999), we refer the reader to the papers by Hall (1997) and Lenstra and Shmoys (1995) for overviews of PTAS.

In queueing theory, the focus has been on theoretical performance evaluation of queueing systems operating under a particular queueing discipline and particular assumptions regarding the inter-arrival and processing time distributions. For example, consider a single-class $M/G/1$ system with arrival rate λ, mean processing time $\mathbf{E}[X]$ and system load $\rho = \lambda\mathbf{E}[X]$ ($0 \leq \rho < 1$). Let $f(\cdot)$ denote the probability density function of the processing time distribution. It has been shown that the distribution of the number of jobs in the system is the same for all non-preemptive scheduling disciplines that do not use job size information (Conway et al., 1967). As a result, the expected response time (flow time or sojourn time) $\mathbf{E}[T]$, defined as the length of time between the arrival of a job and its completion, is the same for all such policies, including *RSS*, *LCFS* and *FCFS*:

$$\mathbf{E}[T^{RSS}] = \mathbf{E}[T^{LCFS}] = \mathbf{E}[T^{FCFS}] = \mathbf{E}[X] + \frac{\lambda\mathbf{E}[X^2]}{2(1-\rho)}. \tag{6.3}$$

Equation (6.3) is based on the famous Pollaczek-Khintchine formula for the expected number of jobs in the system (Pollaczek, 1932; Khintchine, 1932). The distribution of the response time is not the same, however, and, in particular, $\mathbf{var}(T^{FCFS}) < \mathbf{var}(T^{RSS}) < \mathbf{var}(T^{LCFS})$ (Conway et al., 1967). The expected response time in the same system operating under the *SPT*

policy, which is a common rule that does use processing time information, is:

$$\mathbf{E}[T^{SPT}] = \mathbf{E}[X] + \int_0^\infty \mathbf{E}[W^{SPT}(x)]f(x)dx, \tag{6.4}$$

where $\mathbf{E}[W^{SPT}(x)] = \frac{\lambda \mathbf{E}[X^2]}{2(1-\rho(x))^2}$ is the time a job of size x waits before its processing is started, and $\rho(x) = \lambda \int_0^x tf(t)dt$ is the system load consisting of jobs of size less than x only (Phipps Jr., 1956; Harchol-Balter, 2011).

Similar types of results can be obtained for multi-class systems. For instance, consider an $M/G/1$ queue with K classes and with p_k being the probability that an arriving job belongs to class k. Lower-numbered classes have higher priorities (so, class 1 has the highest priority and class K has the lowest), and preemptions are not allowed. The random variable X_k denotes the processing time of a class k job and has distribution G_k. Define X as the "overall" processing time, drawn from distribution $G = \sum_{k=1}^K p_k G_k$, and $\rho_k = \lambda p_k \mathbf{E}[X_k]$ as the system load corresponding to class k jobs only. The expected delay of class k jobs in this system has been shown to be (Wolff, 1989):

$$d_k = \frac{\lambda \mathbf{E}(X^2)}{2(1 - \sum_{k<l} \rho_k)(1 - \sum_{k \leq l} \rho_k)}. \tag{6.5}$$

Recently, there has been a strong interest in the performance evaluation of a class of policies that aims to prioritize shorter jobs in order to ensure "SMAll Response Times"; this class of "SMART" policies was introduced by Wierman et al. (2005). In their paper, Wierman et al. (2005) formalize the commonly used heuristic of giving priority to jobs which are short initially or have small remaining processing times; they derive simple bounds on the mean response time of *any* policy in the SMART class and tight bounds on the mean response time specifically for the preemptive-shortest-job-first policy and *SRPT*. Nuyens et al. (2008) further evaluate the performance of this policy class with respect to the tail of the processing time distribution.

In some cases, it can be shown that a queueing discipline is optimal under particular distributional assumptions. One of the most general results for a multi-class single-server system is the optimality of the $c\mu$ rule in the class of *all* scheduling rules (not just priority policies) in the $M/GI/1$ queue with the possibility of idling and machine breakdowns (Meilijson and Yechiali, 1977). For our example problem, the paper by Chen et al. (1994) shows that when $c_0\mu_0 \leq (c_1 - c_2)\mu_1$, it is optimal for class 1 (type B jobs at machine 1) to have preemptive priority over class 0 (type A jobs at machine 1).

Prescriptive models that determine the best service discipline have been developed in the queueing control literature (Crabill et al., 1977). For example, Robinson (1978) uses semi-Markov decision theory to determine the optimal priority policy for deciding which of two job

types a machine should process; Jaiswal (1968) describes a dynamic programming approach developed by Oliver and Pestalozzi (1965) to optimize processing time thresholds on which priorities are based under the assumption that processing times become known upon arrival. Hassin et al. (2009) address optimization of relative priorities using an achievable region approach, which is discussed in Section 6.1.2.3.1. Related work includes optimization of the service discipline in a setting where there is uncertainty in the input data (Pardo and de la Fuente, 2007) and determination of the optimal thresholds for switching from a preemptive to a non-preemptive discipline (Drekic and Stanford, 2000). A general class of priority policies called fluctuation smoothing policies is proposed by Lu et al. (1994). Queueing models with switchover, which deal with the control of the service process and the queue discipline in the presence of state-dependent switching costs, are reviewed by Rosa-Hatko and Gunn (1997).

Since the underlying idea of dispatching rules and priority queueing models is the same, integration of methodologies from these two fields may be valuable. Priority queueing models can be beneficial from the perspective of scheduling since it may be possible to cast a dispatching rule as a priority queueing discipline and apply queueing analysis in order to derive theoretical performance guarantees for this rule under particular assumptions regarding the processing time or inter-arrival time distributions. As mentioned previously, there has been a significant amount of work in this direction (Harchol-Balter, 2011; Nuyens et al., 2008; Wierman et al., 2005). Conversely, new priority queueing models can be developed based on dispatching rules that have been studied in scheduling but not in queueing theory. It would be interesting to determine whether the steady state performance of composite dispatch rules, such as the Apparent Tardiness Cost heuristic (Pinedo, 2009), could be analyzed using queueing theory.

In addition, it has been noted that *any* scheduling algorithm can be represented as a function of M, P and A, where M is the decision mode, P is a priority function and A is an arbitration rule (Jaiswal, 1982; Ruschitzka and Fabry, 1977). M defines the time points at which the priority function P is evaluated for all jobs in the system. The job with the highest priority value from the function P is chosen for processing. A is used to break ties between jobs with equal priorities. Representing scheduling approaches as priority scheduling algorithms in this manner may provide insight into the properties of these approaches, as well as serve as one possible framework for integrating queueing and scheduling.

6.1.2.2.2 Polling Systems Another category of queueing-based models is known as "polling systems" (Takagi, 1988; Vishnevskii and Semenova, 2006; Boon et al., 2011). In a typical polling system, a single server has to visit and serve several queues of jobs in some order. Jobs belonging to different queues may vary in some of their characteristics, such as their inter-arrival time distributions. The server switching between the queues is equivalent to a machine

or a set of machines being switched between processing of different job types. The polling system can be controlled by deciding (i) the order in which the different queues are served (polling order), (ii) the jobs served on each visit to a queue (queue service discipline), and (iii) the sequencing of jobs within each queue (queue service order) (Takagi, 1988; Wierman et al., 2007). Together, these decisions prescribe the order in which the jobs should be processed, and, thus, form a global scheduling policy.

Polling occurs in a variety of real-world applications (Boon et al., 2011), such as flexible manufacturing (Sharafali et al., 2004). However, for problems that do not naturally possess a polling structure, using a polling model may still be useful. For example, a frequent assumption in polling models is the presence of switching times/costs. Therefore, such models are relevant for single machine scheduling problems with setup times/costs and, as noted by Wierman et al. (2007), for the stochastic economic lot scheduling problem. More generally, we can think of the polling structure, together with decisions (i)–(iii), as a particular scheduling policy type, which can, theoretically, be applied to a wide range of scheduling problems. Thus, in our example problem, which does not require a polling structure, we can still apply a polling-type scheduling policy: (i) "visit" the queues corresponding to job types A and B in a cyclic order (A, B, A, etc.); (ii) during each visit, process jobs of the corresponding type until the queue is empty (exhaustive queue discipline); (iii) sequence jobs within each queue in *FCFS* order. We can then gain an understanding of the performance of this policy by using descriptive results for polling models.

Consider a polling system with N queues where for each queue i, job arrivals follow a Poisson process with rate λ_i, and processing times are distributed according to a distribution $B_i(\cdot)$ with first moment β_i and second moment $\beta_i^{(2)}$. The load of queue i is $\rho_i = \lambda_i \beta_i$, and the total system load is $\rho = \sum_{i=1}^{N} \rho_i$. The server visits the queues in a cyclic order. If no switching time is incurred when the server switches between different queues, then this system is equivalent to an $M/G/1$ queue with arrival rate $\Lambda = \sum_{i=1}^{N} \lambda_i$ and with the service time distribution being defined as $\sum(\lambda_i/\Lambda)B_i(\cdot)$. Since no work is created or destroyed in such a system, it has to obey a conservation law (Yechiali, 1993). Thus, if $E[W_i]$ denotes the mean waiting time for type i jobs, then, regardless of the queueing discipline, the expected amount of work in the system is (Schrage, 1970; Boxma and Groenendijk, 1987):

$$\sum_{i=1}^{N} \rho_i E[W_i] = \rho \frac{\sum_{i=1}^{N} \lambda_i \beta_i^{(2)}}{2(1-\rho)}. \tag{6.6}$$

With non-zero switching times, the above conservation law does not hold, because during the switch-over periods the server is idle while there may be work present in the system. However, useful pseudo-conservation laws (which do depend on the service discipline) have been

developed. Suppose the switching times from queue i, $i = 1, \ldots, N$, to queue $i + 1$, are independent and identically-distributed random variables with first moment s_i and second moment $s_i^{(2)}$. Given that the polling order is cyclic, the mean of the total switching time during one cycle (length of time between visits to the same queue) is $s = \sum_{i=1}^{N} s_i$. For a system with the cyclic polling order and the exhaustive queue discipline, it has been shown that (Boxma and Groenendijk, 1987):

$$\sum_{i=1}^{N} \rho_i E[W_i] = \rho \frac{\sum_{i=1}^{N} \lambda_i \beta_i^{(2)}}{2(1-\rho)} + \rho \frac{s^{(2)}}{2s} + \frac{s}{2(1-\rho)}[\rho^2 - \sum_{i=1}^{N} \rho_i^2]. \tag{6.7}$$

The first term in this expression corresponds to the total amount of work in the system without switch-over times, while the second and third terms represent the amount of work present at an arbitrary epoch during a switch-over period (Levy and Sidi, 1990).

In fact, there has been an extensive amount of descriptive work on polling systems under particular assumptions regarding decisions (i)–(iii). We refer the reader to the work of Takagi (1986), Takagi (1988) and Levy and Sidi (1990) for overviews of such work. Below, we focus on optimization models for the three different types of decisions that need to be made in polling models.

(i) Polling Order Similarly to dispatching rules, the polling order can be static or dynamic. For example, the cyclic policy, which states that the server should visit the N queues in order $1, 2, \ldots, N, 1, 2, \ldots$, is a static order since it is chosen prior to system operation and is independent of the system state (Levy and Sidi, 1990). An example of a dynamic polling order is one in which the server is assigned to the queue that contains the greatest amount of work at a decision epoch (Levy and Sidi, 1990). Optimization of a static polling order amounts to optimizing a pattern which repeats itself M times (referred to as a *polling table*) and has the form $I(1), I(2), \ldots, I(M), I(1), I(2), \ldots$, with $I(i)$ being the identity of the queue in the ith position of the pattern, $1 \leq I(i) \leq N$ (Levy and Sidi, 1990). Boxma et al. (1989; 1991) employ a three-step heuristic approach for this problem:

Step 1: Determine the relative visit frequencies f_i for queues $i = 1, 2, \ldots, N$,

Step 2: Determine the size of the polling table, M, and the number of visits, m_i, that should be made to queue i, $i = 1, \ldots, N$, within a cycle,

Step 3: Given the values of M, m_i and f_i for all i, determine the order of visits to the queues within a cycle.

Each of the three steps is itself an optimization problem. For step 1, Boxma et al. (1989) propose to use the optimal proportions obtained from solving a related non-linear optimization problem in a Markovian polling system. For step 2, the table size M can be chosen in such a way that ensures that $Mf_1, Mf_2, \ldots, Mf_N$ are integers, or within ϵ of an integer, and such that the sum of these integers equals M. For step 3, the goal is to find an order in which, for every i, the number of visits to other queues between two visits to queue i is approximately the same; the authors use the Golden Ratio policy proposed by Hofri and Rosberg (1987). If we fix the table size, M, steps 2 and 3 become closely related to the notion of fair sequences that has received attention in the scheduling literature (e.g., see the book chapter by Kubiak (2004)): given "desired" frequencies f_i, we may optimize some function of the differences between the actual number of visits, m_i, and f_i so as to achieve a "fair" polling order.

Now, suppose that the system state, $(n_1, n_2, \ldots, n_N)$, where n_i is the number of jobs currently present in queue i, can be observed at the beginning of each cycle. Browne and Yechiali (1989a; 1989b) show that visiting the queues in increasing order of n_i/λ_i minimizes the expected length of the cycle. This result holds for the two most common queue service disciplines and is independent of the service time requirements of the queues. Therefore, we can combine this rule with policies other than *FCFS* to solve more complex scheduling problems. For example, a manufacturing facility may want to minimize the length of a production cycle but also minimize the total job tardiness. In this case, we can use the above result to construct the production cycle with the minimum expected length; if we can observe processing times and due dates upon the server's arrival to a queue, then a static, deterministic scheduling problem can be solved to optimize the total tardiness of the current set of jobs. To our knowledge, such an approach has not been investigated in the literature.

The reader is referred to the papers by Levy and Sidi (1990) and Yechiali (1993) for overviews of approaches to optimization of both static and dynamic polling orders, and to the papers by Altman and Yechiali (1993), Yechiali (1991), Borst et al. (1994) and Gaujal et al. (2007) for additional examples. Khamisy et al. (1992) address the equivalent problem of positioning the queues on the cyclic path of the server (referred to as optimization of the network topology) in a polling system with precedence constraints.

(ii) Queue Service Discipline The rule that is used to decide the number of jobs served during a visit to a particular queue is the queue service discipline (Vishnevskii and Semenova, 2006). Two common queue service disciplines are the exhaustive and the gated. Under the exhaustive discipline, a queue is served until it becomes empty. Under the gated service discipline, all jobs that are present in the queue at the start of the visit are served (all jobs that arrive during service have to be processed in the next visit). The exhaustive policy tends to

optimize the system's efficiency, while the gated policy is known to be fairer in its allocation of the processing resource among different queues (Levy and Sidi, 1990; Levy, 1991).

The general idea for controlling the system via the queue service discipline is to employ disciplines with controllable parameters that limit the amount of service each queue receives. One option is to employ a deterministic limited policy, which has a parameter L_i that represents the maximum number of jobs that should be served during a visit to queue i.[13] Since even performance evaluation of this policy is difficult, there has been work on a related alternative: the binomial-gated policy (Levy, 1991). In such a policy, a parameter, p_i, represents the proportion of jobs present in queue i at the start of the cycle that should be served during this cycle. Assuming that X_i is the number of jobs in queue i when the cycle begins, the number of jobs served is a binomial random variable with parameters X_i and p_i; thus, on average, a fraction p_i of the jobs present in queue i at the beginning of the cycle will be served during the cycle. Optimization of the system therefore amounts to finding the optimal p_i values. Since one of the main reasons this policy is considered is its analytical tractability, there is some doubt regarding its applicability in practice.

Another way to control the service discipline is to use a Bernoulli limited policy: upon completion of a job in queue i, the probability that the server will process a job from the same queue is given by a parameter q_i. Blanc and van der Mei (1995) study the problem of finding q_i values for all queues with the objective of minimizing the sum of steady-state mean waiting times weighted by arbitrary strictly positive values.

van Wijk et al. (2010) propose an approach to find service disciplines that balance efficiency and fairness. Their measure of efficiency is $\sum_{i=1}^{N} \rho_i \mathbf{E}[W_i]$. Fairness is expressed in terms of the differences in mean waiting times between the queues, $\max_{i,j}(\mathbf{E}[W_i] - \mathbf{E}[W_j])$. Their goal is to optimize a weighted combination of these objectives:

$$\tilde{\gamma}(\alpha) := \max_{i,j}(\mathbf{E}[W_i] - \mathbf{E}[W_j]) + \alpha \sum_{i=1}^{N} \rho_i \mathbf{E}[W_i], \tag{6.8}$$

for some $\alpha \in [0, \infty)$, by choosing the values κ_i for every queue i, and implementing the corresponding κ-gated service discipline. This discipline allocates at most κ_i gated service phases to each queue i once the server arrives there. In the first phase at queue i, the server processes all jobs present at the time of its arrival to this queue. If there are jobs present in queue i once phase 1 is finished, the server stays at this queue and processes all jobs that have arrived since the start of the first phase (this corresponds to phase 2). If there are jobs in queue i when phase 2 is completed, phase 3 is started, etc., until at most κ_i phases are completed. At

[13] See the paper by Levy and Sidi (1990) for a summary of variations of the limited policy.

that point, the server switches to the next queue j and serves it using at most κ_j phases. This multi-phase approach is very similar to the notion of periodic scheduling, since in both cases the system is periodically reviewed and only those jobs that have arrived by the review time point are considered for scheduling.

(iii) Queue Service Order Work on scheduling within a given queue in polling systems has received little attention compared to optimization of the polling order and the queue service discipline. Fournier and Rosberg (1991) consider systems with various priority disciplines within each queue. Wierman et al. (2007) demonstrate that the order of service within the queue can have a significant impact on the performance of the system. They analyze policies that use processing time information, such as *SPT*, as well as those that do not. For instance, they consider a symmetric two-queue polling system with processing times and setup times being exponentially distributed with mean 1, and with a cyclic polling order and gated service discipline. They experimentally show that the mean waiting time under *SPT* is, on average over a variety of loads, 15% lower than that of *FCFS*. For the same system with a more-variable Weibull distribution, the improvement of *SPT* over *FCFS* is even greater (Wierman et al., 2007). Boon and Adan (2009) study a polling system in which jobs within each queue are divided into low- and high-priority classes.

Optimization of the queue service order is a promising area for scheduling techniques. When a gated or a limited-type queue service discipline is employed, one needs to solve a single-machine static scheduling problem at the beginning of each queue visit. If processing times become known upon arrival, deterministic single-machine approaches may be applied; if only expected processing times are known, then a stochastic scheduling problem needs to be solved. These problems may be of varying complexity: minimizing flow time is polynomial (employing *SPT* or expected *SPT* policies is optimal) while minimizing total tardiness is in general NP-complete (Baker and Trietsch, 2009) and therefore would require a more sophisticated optimization method.

Although most of the polling literature focuses on single-server and single-station models, there have been extensions of these models to multiple-server systems (Borst, 1995; Down, 1998; Antunes et al., 2011) and networks of polling systems (Reiman and Wein, 1999; Beekhuizen et al., 2008). Scheduling decisions in such systems are the same as in single-station, single-server ones (the order of visiting the queues, the set of jobs served on each visit and the sequence in which the jobs are to be served), but they have to be made for every server and every station, and are dependent on the locations of all servers, making the overall problem more complex. As a consequence, there has been very little work on optimization of scheduling

decisions in such systems. An important example is the work by Browne and Weiss (1992), who consider a polling system in which the server is composed of c parallel machines which switch between queues as one unit. They address optimization of the polling order at the start of each cycle with the goal of minimizing the expected cycle length (as do Browne and Yechiali (1989a; 1989b) for a single-server system). In Chapter 7 of this dissertation, we study a polling system in which the server is composed of two machines in tandem. We investigate the performance of periodic scheduling methods in which a static deterministic scheduling problem is solved at the beginning of each queue visit under cyclic, gated assumptions.

In summary, there are numerous avenues for integration of polling research and work in the scheduling literature. Firstly, applying a polling-type scheduling policy to a problem implies that a vast number of descriptive results become directly applicable. Such results may be used as bounds on optimal schedules or within scheduling algorithms. Secondly, the division of the scheduling policy into three levels offers a natural framework for modelling systems with different objectives at different decision-making levels. Scheduling approaches could also prove useful for improving the performance of polling systems. Specifically, if we make the assumption that processing times of jobs become known upon arrival, then the problem of optimizing the sequence in which jobs are served in each queue can be turned into a static, deterministic scheduling problem, making an abundance of scheduling work directly relevant. If the total workload within each queue can be observed at the beginning of each cycle, then the problem of optimizing the polling order becomes that of optimizing a system with N (batch) jobs. It would be interesting to determine whether work on lot-sizing or batch scheduling methods would be applicable in this context. The interested reader is referred to the work by Winands (2007) to see the connection between lot-sizing and polling systems, and is encouraged to contrast the idea of using a batch scheduling model for the problem of optimizing the polling order with the work on batch polling systems (Van Der Wal and Yechiali, 2003; Boxma et al., 2008). Additionally, future work should consider the use of polling models with precedence constraints (Khamisy et al., 1992) for modelling scheduling problems with precedences and the task management problem described by Myers et al. (2007).

6.1.2.2.3 Vacation Models In a queueing system with vacations, the server is assumed to take "vacations" from serving a particular queue of customers. Vacations can correspond to breakdowns of a machine, maintenance operations, or service of other job classes (Stidham Jr., 2002; Doshi, 1986). Consider the example of Figure 6.1. From the perspective of a class 0 job, service of class 1 can be viewed as a machine vacation (Stidham Jr., 2002). Adopting this point of view, one can see that determining how to allocate machine capacity among job classes is equivalent to determining the timing and duration of vacations. In fact, a vacation model can

be seen as a special case of a polling system. Thus, we do not review vacation models in as much detail as the work on polling systems. We note, however, that while the methodology employed in the study of vacation models is similar to that of polling systems, some more general results have been obtained (Stidham Jr., 2002). One of the most important results is that the waiting time in an $M/GI/1$ queue with vacations follows the distribution of the sum of two independent components: the waiting time in an $M/GI/1$ queue without vacations and the equilibrium residual vacation time (Stidham Jr., 2002; Fuhrmann, 1984). Vacation models could be useful for scheduling in the same way as work on optimization of the polling order and queue service discipline in polling systems. In particular, it can provide high-level guidance regarding how much time should be spent on specific class prior to taking a vacation. A review of descriptive vacation models can be found in the paper by Doshi (1986), while various prescriptive vacation models are discussed by Tadj and Choudhury (2005). Extensions of vacation models to the case of multiple servers have been analyzed by, for example, Kao and Narayanan (1991) and Chao and Zhao (1998). The book by Tian and Zhang (2006) provides a comprehensive review of both descriptive and prescriptive vacation models.

6.1.2.2.4 Bandit Models A multi-armed bandit (MAB) problem (Gittins, 1979; Whittle, 1988; Bertsimas, 1995; Bertsimas and Niño-Mora, 1996) consists of a set of projects $1, \ldots, K$, only one of which can be processed at each discrete point in time. For every time point t when a project k in state $j_k(t)$ is being worked on, a reward, $R^k_{j_k(t)}$, is obtained. The rewards are assumed to be additive and are discounted in time by a factor β, $0 < \beta < 1$. The project that is being executed changes state according to a homogeneous Markov transition rule, while the states of the projects that are not being worked on do not change. The goal of the problem is to determine a scheduling policy: a rule that, at each point in time, prescribes which project should be executed and maximizes the total expected discounted reward over an infinite horizon (Bertsimas and Niño-Mora, 1996). Interestingly, in computer science, there has been a significant interest (Streeter, 2007; Gagliolo and Schmidhuber, 2007; Cicirello and Smith, 2005; Radlinski et al., 2005; Vermorel and Mohri, 2005) in a variation of the MAB problem in which the distribution of rewards is not known in advance and the goal is to choose, at each iteration, the project to execute in order to maximize the sum of collected rewards.

The initial inspiration for the study of MABs is the problem faced by a gambler in a casino deciding which slot machine should be played (Puterman, 1994), or, equivalently, which lever or arm of a slot machine should be pulled (Weber and Weiss, 1990). Naturally, the MAB problem is also a representation of the situation where a decision-maker needs to choose which project to execute at a point in time, with states defined by the projects' levels of completion and a reward being obtained only when a project is finished (Puterman, 1994). From the perspective

of scheduling, the MAB problem is a representation of a single machine scheduling problem with preemptions in which the set of jobs stays fixed over time (Pinedo, 2009). The state of the job can be the remaining processing time, which changes only when this job is the one being processed by the machine. Bertsimas and Niño-Mora (1996) state that the MAB problem is a special case of a dynamic and stochastic job scheduling problem. An additional application of MABs is in sequential clinical trials where a new medical treatment is compared to an existing one or to a placebo (Puterman, 1994).

The arm-acquiring bandit problem is an extension of the MAB problem in which a set of new projects, $A(t)$ arrives at time t. These jobs can be executed starting at time $t + 1$ and are assumed to be independent of each other and of all the previous jobs. As in the MAB problem, the goal is to determine a policy that specifies the project that should be executed at each point in time (Mahajan and Teneketzis, 2007). Since each project can be viewed as a job and each state of a job k, $j_k(t)$, can correspond to the amount of processing received by the job up to time t, the problem can be used as a representation of a single machine dynamic scheduling problem with preemptions. The example of Figure 6.1 can be viewed as an arm-acquiring bandit problem if we allow preemption and if we consider jobs as being equivalent to projects. Within the queueing literature, the MAB problem is related to polling models, since the decision to execute a particular project can be equivalent to choosing a particular queue.

There are many other variations of MAB models, most of which can be linked to problems from the scheduling literature. These include:

- the MAB problem with switching penalties (Mahajan and Teneketzis, 2007), corresponding to a static single machine preemptive scheduling problem with switching costs.

- the MAB problem with deadlines (Niño-Mora, 2007), corresponding to a static single machine preemptive scheduling problem with deadlines.

- the branching bandit problem (Varaiya et al., 1985; Weiss, 1988), in which projects are classified into types according to their state and where a project is replaced by some number of new projects of each type upon its completion. This problem allows modelling of job arrival processes that are more general than Poisson and can be used to represent a machine in a job shop that processes a variety of parts (Bertsimas et al., 1995).

- the restless bandit problem (Weber and Weiss, 1990; Niño-Mora, 2007), in which m projects have to be operated at any time point; rewards may be incurred and states may change even for projects that are not being worked on. In the literature, it is stated that these models can represent situations where m out of K available workers always need to be active, and their states represent their physical condition; states change regardless

of whether the worker is busy or idle (e.g., if the workers are getting rest, their physical condition improves) (Weber and Weiss, 1990). Similarly, we can use the restless bandit problem to model a manufacturing environment with m parallel machines; this model would be general enough to represent deterioration and improvement in the condition of machines. It would be interesting to determine the relationship between the restless bandit problem and stochastic variations of resource constrained project scheduling discussed by Mercier and Van Hentenryck (2008). The restless bandit model can also be applied in the context of the system shown in Figure 6.1, since we can think of type A and B jobs as two projects which change state depending on the amount of processing received and on the amount of work that arrives.

- the MAB problem with a goal state (Dumitriu et al., 2003; Katta and Sethuraman, 2005), which is related to planning problems in artificial intelligence (Ghallab et al., 2004).

One of the biggest advantages of the MAB problem representation and its variants comes from a powerful class of policies that have been developed for solving them. These policies are based on a priority *index* that is defined for each project as a function of its state. At each time point, the set of projects with the currently greatest priority index values is chosen for processing (Niño-Mora, 2007).[14]

Interestingly, the earliest result on the optimality of priority index rules arose in the context of a *deterministic* problem (Niño-Mora, 2007). Specifically, Smith (1956) showed that an index rule is optimal for the problem of minimizing the sum of completion times of a set of jobs with known processing times and linear holding costs (weights). In this setting, the index of a job can be defined as the ratio of the holding cost rate per unit of time to its processing time, which represents the cost reduction per unit of effort, or the average productivity of work on the job (Niño-Mora, 2007). Subsequently, Rothkopf (1966) extended Smith's result to the problem with stochastic job durations. Cox and Smith (1961) showed the optimality of an equivalent index rule in the context of a multi-class single-server queue with linear holding costs. The seminal work by Klimov (1975) develops an optimal index rule for a more complex version of this problem, one with Bernoulli feedback between job classes (Niño-Mora, 2007). From the perspective of this dissertation, it is interesting to note that these results, currently considered fundamental in queueing theory, have their roots in deterministic scheduling.

Both the MAB problem and the arm-acquiring bandit problem are solvable by a Gittins index policy (Gittins, 1979; Bertsimas and Niño-Mora, 1996; Mahajan and Teneketzis, 2007), which is a generalization of the well-known $c\mu$-rule (also discussed in the context of priority

[14]These policies are of the same nature as those discussed in the priority queues section above, and they can be similarly classified into static and dynamic policies.

queues above, and in the context of fluid models and the achievable region method in Section 6.1.2.3). According to such a policy, at each time point t, one should process the job with the highest value of the Gittins index, which represents the maximum expected discounted reward per unit of expected discounted time due to the processing of a job (Mahajan and Teneketzis, 2007). Heuristics based on the Gittins index have also been developed for more complex problems, such as research planning (Glazebrook and Owen, 1995). For the restless bandit problem, Whittle (1988) provides an index policy based on the solution of its relaxation, while Weber and Weiss (1990) show that this policy is asymptotically optimal. The paper by Niño-Mora (2007) provides a unifying approach for developing and computing index-based priority policies, illustrating a number of application areas, including scheduling in multi-class queues and dynamic priority allocation to multiple stochastic projects. We note that MDP methods (discussed in Section 6.1.2.1) and achievable region methods (discussed below in Section 6.1.2.3) have both been used to examine bandit problems (Puterman, 1994; Bertsimas, 1995).

The simplicity of index policies, together with their theoretical optimality in some settings, implies that it may be useful to model part or all of a dynamic scheduling problem as a bandit problem. For instance, a multi-armed or an arm-acquiring bandit problem could be used as a relaxation of a non-preemptive scheduling problem that is solved at each scheduling point in a predictive-reactive method. Alternatively, the Gittins indices of the jobs could be used to create effective scheduling heuristics or to guide a predictive-reactive approach to more globally optimal decisions, since these indices can be incorporated into the constraints or the objective function of the models used to construct predictive schedules.[15]

6.1.2.3 Methods Based on Approximations and Abstractions

In the previous section, we discussed methods that deal with the complexity of scheduling in queueing systems by restricting the type of policies considered. In this section, we discuss an alternative methodology that uses approximations or abstractions of the original system. This methodology consists of four steps:

1. modelling and development of the approximation/abstraction model for the network control problem of interest;

2. solution of the approximation/abstraction;

3. derivation of an implementable scheduling policy for the original scheduling problem from the solution to the approximation/abstraction;

4. asymptotic performance analysis of the solution.

[15] Predictive-reactive methods are discussed in Section 6.2.

We discuss the approximation/abstraction methods of steps 1 and 2 in the following section. General approaches for step 3 are discussed in Section 6.1.2.3.2. Step 4 is outside the scope of this dissertation, and the reader is encouraged to consult the papers referenced throughout this section.

6.1.2.3.1 Approximations and Abstractions While the idea of approximating the problem of interest by a simpler one has also been used in scheduling, the characteristics that are relaxed to develop the approximation are different than those used in queueing. In scheduling, approximation techniques are based on relaxing precedence or no-preemption constraints; in queueing, approximations are based on scaling time and space in order to view high level patterns in the system's behaviour. We discuss two such approximations here: Brownian models and fluid models.

An abstraction approach that fits within the above framework is the achievable region method. Instead of directly considering the problem of finding the optimal control policy, it aims to find the optimal achievable objective value. To our knowledge, this methodology has no direct equivalent in the scheduling literature, but is closely related to the idea of finding lower and upper bounds on the objective function that are used to prune the search space in scheduling.

Brownian Models In heavy-traffic conditions, when the utilization of a system approaches its capacity, costs are amplified and optimal control is even more important than in underutilized systems (Stidham Jr., 2002). It has been shown that the queue length or the workload process of a queueing network with balanced heavy loading[16] can be approximated by a diffusion process called the reflecting Brownian motion (Williams, 1996). The resulting Brownian models have the advantages of requiring a minimum amount of data and of being based on a compact mathematical representation (Harrison, 2003). Moreover, they can be used for networks with multiple classes of jobs, both feedforward and feedbackward routings and machines subject to various types of disruptions (Chen and Yao, 2001).

Specifically, Brownian models are constructed by compressing time by a factor of n and compressing the spatial dimensions by a factor of $\sqrt{n}$ (Harrison, 1996). When looking at the system on this scale, it is impossible to pay attention to detailed scheduling decisions, but *trends* in the behaviour of the system become apparent. Therefore, solving the Brownian scheduling problem amounts to determining high level properties such as conditions for idling

[16]For a general open network, this term refers to the situation when the load imposed on each station by some exogenous input process is approximately equal to the capacity of that station. In a closed queueing network, the term implies that the total population within the network is large and the relative intensities for the different stations are approximately the same (Harrison, 1988).

the machine(s). These properties can then be translated into implementable scheduling policies. This approach is best illustrated using the two-station example presented at the beginning of Section 6.1.2 and the paper of Harrison and Wein (1989).

Assume that the utilization of each station is close to 1 (Kelly and Laws, 1993). Let $Q_k = \{Q_k(t), t \geq 0\}$ be the queue length process of job class k, $k = 1, 2, 3$, and let $I_i(t) = \{I_i(t), t \geq 0\}$ be the total amount of time that machine i, $i = 1, 2$, is idle in $[0, t]$. The scaled versions of these processes are $Z_k = \{Z_k(t), t \geq 0\}$ and $U_i = \{U_i(t), t \geq 0\}$ where:

$$Z_k(t) = \frac{Q_k(nt)}{\sqrt{n}}, \quad t \geq 0, \text{ and } k = 1, 2, 3, \tag{6.9}$$

$$U_i(t) = \frac{I_i(nt)}{\sqrt{n}}, \quad t \geq 0, \text{ and } i = 1, 2. \tag{6.10}$$

In order to define the Brownian model in terms of workload present in the system, let M_{ik} be the expected amount of time that should be given by machine i to a class k job before it leaves the system (Harrison and Wein, 1989); denote the workload profile matrix made up of the M_{ik} values by $\mathbf{M}$. Let $\{W_i(t), t \geq 0\}$ be the scaled workload process defined as $W_i(t) = \sum_{k=1}^{3} M_{ik} Z_k(t)$, $t \geq 0$ and $i = 1, 2$. $W_i(t)$ is therefore the total expected amount of scaled work left for machine i at time t anywhere in the system. It has been shown by Harrison (1988) that the sequencing problem at machine 1 is well-approximated by the Brownian control problem of choosing processes $\mathbf{Z}$ and $\mathbf{U}$ which are right continuous with left limits and are a solution to the following:

$$\text{minimize} \qquad \lim_{T \to \infty} \sup \frac{1}{T} E\left[\int_0^T \sum_{k=1}^{3} Z_k(t) dt\right] \tag{6.11}$$

$$\text{subject to} \quad \frac{1}{2} Z_1(t) + \frac{1}{2} Z_2(t) = B_1(t) + U_1(t) \text{ for all } t \geq 0, \tag{6.12}$$

$$Z_2(t) + Z_3(t) = B_2(t) + U_2(t) \text{ for all } t \geq 0, \tag{6.13}$$

$$\mathbf{U} \text{ is non-decreasing with } U(0) = 0, \tag{6.14}$$

$$Z(t) \geq 0 \text{ for all } t \geq 0, \tag{6.15}$$

$$\mathbf{Z} \text{ and } \mathbf{U} \quad \text{are non-anticipating with respect to } \mathbf{X}, \tag{6.16}$$

where $\mathbf{X}$ is a three-dimensional Brownian motion[17] with drift vector θ and covariance matrix $\mathbf{\Gamma}$, $B(t) = \mathbf{M}X(t)$, so that $\mathbf{B} = (B_1, B_2)$ is a two-dimensional Brownian motion with drift $\mathbf{M}\theta$ and covariance matrix $\mathbf{M\Gamma M}^T$. The solution to this model can be obtained by solving a linear program that is embedded in the above at every time t. In fact, there exists a solution that, with

[17] See Chapter 10 of the book by Ross (2003) for an introduction to Brownian motion.

probability 1, simultaneously minimizes $\sum_{k=1}^{3} Z_k(t)$ for all t. This solution is $(\mathbf{U}^*, \mathbf{Z}^*)$ where

$$U_i^*(t) = -\inf_{0\le s\le t} B_i(s) \quad \text{for} \quad i = 1, 2, \tag{6.17}$$

$$Z_1^*(t) = [2b_1^*(t) - b_2^*(t)]^+, \tag{6.18}$$

$$Z_2^*(t) = [2b_1^*(t) \wedge b_2^*(t)], \tag{6.19}$$

$$Z_3^*(t) = [b_2^*(t) - 2b_1^*(t)]^+, \tag{6.20}$$

and $\sum_{k=1}^{3} Z_k^*(t) = [2b_1^*(t) \vee b_2^*(t)]$, $t \ge 0$.

Based on the definition $W_i(t)$ and Equations (6.12), (6.13) and (6.17), this solution can be interpreted as follows: $U_i^*(t)$ increases only at times t when $W_i^*(t) = 0$ for $i = 1, 2$. Therefore, in this solution, machine 2 incurs scaled idleness only when there are no scaled jobs of type B *anywhere in the network* (Harrison and Wein, 1989). The reader is referred to the papers by Harrison and Wein (1989) and Kelly and Laws (1993) for additional details. The interpretation of the above solution suggests that the policy for the original network should attempt to avoid machine 2 idleness when there are type B jobs in the system. However, finding a policy that implements this notion in a way that is optimal for the original system is non-trivial. For example, one intuitive implementation is a policy that always gives priority to type B jobs at station 1. This policy, however, will tend to keep a greater total number of jobs in the system than necessary since it will delay the processing of type A jobs, which would have otherwise left the system faster. Thus, a better approach is a policy that gives priority to type A if the number of jobs at station 2 is greater than some value c and gives priority to type B otherwise. This policy balances the short-run objective of reducing the number of jobs in the system (due to giving priority to A jobs some of the time) and the longer-run objective of avoiding idleness at station 2. The best value of parameter c and the asymptotic behaviour of the proposed policy (Step 4 of the above framework) are discussed by Harrison and Wein (1989). The paper by Martins et al. (1996) provides a rigorous proof of the connection between the problem of scheduling in this network and the diffusion process that is its heavy traffic limit.

Fluid Models Another way to approximate the problem of scheduling in a multi-class queueing system is to relax the assumption that the system serves discrete jobs. If we consider the behaviour of the system over a long time horizon, this makes intuitive sense – viewing the evolution of the system at a high level would give one the impression of continuous fluids moving through the system, rather than discrete entities. Thus, a multiclass fluid network is different from the network it approximates because it serves continuous fluid flows rather than discrete jobs. Scheduling of this network corresponds to real-time allocation of the available processing capacity of each station to the various fluid classes (Chen and Yao, 1993). The fluid

solution can then be translated into scheduling decisions for the original system in both static and dynamic conditions.

Fluid model work on classical, deterministic scheduling problems is based on the fundamental result that the optimal fluid makespan provides a lower bound on the optimal makespan in a high-multiplicity[18] job shop (Bertsimas and Gamarnik, 1999). The optimal fluid makespan is, simply, equal to the workload on the bottleneck machine (i.e., the machine lower bound). Bertsimas and Gamarnik (1999), Boudoukh et al. (2001) and Bertsimas and Sethuraman (2002) use these results to develop heuristic algorithms that produce asymptotically optimal schedules as the number of jobs of each class (the multiplicity) grows. Bertsimas et al. (2003) apply the same ideas to the harder problem of minimizing inventory holding costs in a job shop. Dai and Weiss (2002) address minimization of makespan in a more general setting: the processing times of jobs in a class are different but follow the same distribution. Nazarathy and Weiss (2009, 2010) study high-volume job shops, where the number of jobs is large in comparison to the number of machines and the maximum number of activities per job, with weighted flow time and makespan objectives, respectively.

The paper of Nazarathy and Weiss (2010), in addition, compares a typical job shop scheduling problem with a multi-class queueing network problem and provides a step toward the same goal of integration of queueing and scheduling as this dissertation. Specifically, consider the job shop scheduling problem with N jobs and M machines. Each job j consists of r_j activities. Activity i, $i = 1, \ldots, r_j$, of job j is processed[19] on machine $\sigma(i, j)$ and has duration $p_{i,j}$. In a high-volume job shop, N is large, but M is fixed and r_j is bounded. This problem can be modelled as a multi-class queueing network as follows. The activities are classified into a set of classes $\{1, \ldots, K\}$. An activity of class k is processed by machine $\sigma(k)$, and the set of classes processed by machine m is denoted K_m. Considering the processing times of all activities in class k leads to a processing time distribution G_k with rate μ_k. Similarly, by considering the next routing step possible after the completion of any class k activity, we can define $P_{k,l}$ as the fraction of class k activities that, upon completion, turn into class l, and $1 - \sum_l P_{k,l}$ as the fraction of class k activities that, upon completion, leave the system. Thus, the corresponding multi-class queueing network consists of M machines and K classes, with jobs of class k served according to distribution G_k and routed according to the probability matrix $\mathbf{P}$ defined by the $P_{k,l}$ values. The initial number of jobs in each class is $Q_k(0)$, $k = 1, 2, \ldots, K$, with $N = \sum_k Q_k(0)$. There are no exogenous arrivals and so this system is called a *finite-horizon* multi-class queueing network (Nazarathy and Weiss, 2010).

The dynamics of the fluid model corresponding to this queueing network are based on the

[18] The term *high-multiplicity* refers to the fact that more than one job is present in each class.

[19] A job is allowed to visit the same machine more than once along its route.

following equation:

$$q_k(t) = q_k(0) - \mu_k \int_0^t u_k(s)ds + \sum_l P_{l,k}\mu_l \int_0^t u_l(s)ds, \tag{6.21}$$

where $q_k(t)$ is the amount of fluid present in class k at time t and $u_k(s)$ is the instantaneous allocation of the processing capacity of machine $\sigma(k)$ to class k. This equation states that the total amount of fluid in class k at time t is equal to the initial amount of fluid minus the amount that has been processed up to time t, plus the amount that has arrived from other classes by time t. Let $q_k^+(t)$ be the total amount of class k fluid that still needs to flow through class k (including fluid currently in other classes that will turn into class k eventually), and let $\mathbf{T}^*$ be the machine lower bound for the job shop. The solution to the fluid makespan minimization problem is shown by Nazarathy and Weiss (2010) to be

$$u_k(t) = \frac{q_k^+(0)}{\mu_k \mathbf{T}^*} \tag{6.22}$$

$$q_k(t) = q_k(0)(1 - \frac{t}{\mathbf{T}^*}) \tag{6.23}$$

$$q_k^+(t) = q_k^+(0)(1 - \frac{t}{\mathbf{T}^*}). \tag{6.24}$$

Let $Q_k(t)$ be the number of activities that are present in the queue of machine $\sigma(k)$ at time t, and $Q_k^+(t)$ be the total number of class k activities that still need to be completed at time t (including activities of class k that have not yet arrived at $\sigma(k)$). The jobs can then be scheduled via an online fluid tracking policy:

for each time t and each machine i that is free
  define $\mathcal{K}_m(t) = \{k \in \mathcal{K}_m : Q_k(t) > 0\}$, the set of classes
  that currently have activities available for processing at machine m
  if $\mathcal{K}_m(t) = \emptyset$
    idle machine m
  else
    process an available activity of class $k^* \in \arg\max_{l \in \mathcal{K}_m(t)}(\frac{Q_l^+(t) - q_l^+(t)}{q_l^+(t)} + 1)/(1 - \frac{t}{\mathbf{T}^*})$.

Ties for k^* as well as the choice of activity within k^* can be decided using arbitrary rules. We refer the reader to the paper by Nazarathy and Weiss (2010) for additional details as well as an overview of other job-shop scheduling rules based on the solution of the fluid model. We also refer the reader to the thesis by Raviv (2003) which demonstrates that fluid models can be used as approximations of large instances of other hard combinatorial problems.

The fluid policy described above can be applied in a dynamic setting if the fluid model on

which it is based is adjusted to include exogenous arrival information. In this case, Equation (6.21) becomes

$$q_k(t) = q_k(0) + \lambda_k(t) - \mu_k T_k(t) + \sum_l P_{l,k}\mu_l T_l(t), \tag{6.25}$$

where $T_k(t) = \int_0^t u_k(s)ds$ is the cumulative amount of time that station $s(k)$ devotes to class k in $[0, t]$ (Chen and Yao, 1993). Suppose we would like to minimize holding costs in a network with K classes and M machines, $M \leq K$, arrival rate vector $\boldsymbol{\lambda} = (\lambda_1, \ldots, \lambda_K)$, service rate vector $\boldsymbol{\mu} = (\mu_1, \ldots, \mu_K)$ and routing matrix $\mathbf{P}$. Define two additional matrices: $\mathbf{R} = (\mathbf{I} - \mathbf{P}^T)diag(\mu)$, where $diag(\mu)$ is a diagonal matrix of processing rates; and the constituency matrix $\mathbf{C}$ with each entry $c_{m,k} = 1$ if $k \in K_m$ and 0 otherwise. Let $\mathbf{1}$ be a vector of 1s of the appropriate dimension, and let $1_{\mathbf{q}(t)\neq 0}$ be an indicator function which equals 1 whenever there is at least one non-empty queue. Define the vector $\mathbf{q}(t) = (q_1(t), \ldots, q_K(t))$. Let h_k be the per time unit cost for holding one unit of class k fluid, and $\mathbf{h} = (h_1, \ldots, h_k)$ (Chen and Yao, 1993). The fluid model for the problem of optimally controlling a network is then:

$$\begin{aligned}
\min_{\mathbf{T}(t)} \quad & \mathbf{h}\mathbf{q}(t) && (6.26)\\
\text{subject to } \mathbf{q}(t) &= \mathbf{q}(0) + \boldsymbol{\lambda}t - \mathbf{R}\mathbf{T}(t) \geq 0, && (6.27)\\
\mathbf{T}(t) & \quad \text{is non-decreasing with } \mathbf{T}(0) = 0, && (6.28)\\
\mathbf{U}(t) &= \mathbf{1}t - \mathbf{C}\mathbf{T}(t) \text{ is non-decreasing.} && (6.29)
\end{aligned}$$

When $h_k = 1$ for all k, the above objective can be interpreted as the minimization of expected throughput time (Atkins and Chen, 1995). For the example problem of Figure 6.1, $\mathbf{q}(t) = (q_0(t), q_1(t), q_2(t))$, $\boldsymbol{\lambda} = (\lambda_0, \lambda_1, \lambda_2)$, $h = (c_A, c_B, c_B)$,

$$\mathbf{P} = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{pmatrix},$$

$$\mathbf{R} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & -1 & 1 \end{pmatrix},$$

and

$$\mathbf{C} = \begin{pmatrix} 1 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}.$$

One approach for obtaining a solution to the above fluid control problem is based on the

idea that for some (possibly small) period of time, say $[0, t_1]$, the allocation of capacity remains a constant proportion, i.e., $T(t) = \mathbf{x}t$ with a fixed $\mathbf{x}$ for all $t \in [0, t_1]$ (Chen and Yao, 1993). Each component of the vector $\mathbf{x}$, x_k, $k = 1, \ldots, K$, represents the proportion of capacity that station $\sigma(k)$ should devote to processing class k fluids (Atkins and Chen, 1995) and forms the solution to the following linear program (Atkins and Chen, 1995; Chen and Yao, 1993):

$$\begin{aligned} \max \quad & \mathbf{cx} && && (6.30)\\ \text{s.t.} \quad & (\mathbf{Rx} - \boldsymbol{\lambda})_k \leq 0, \quad k \in \Pi && && (6.31)\\ & \mathbf{Cx} \leq \mathbf{1} && && (6.32)\\ & \mathbf{x} \geq 0. && && (6.33) \end{aligned}$$

The set Π is a subset of $\{1, \ldots, K\}$ corresponding to classes with 0 fluid level, and $\mathbf{c} = \mathbf{hR}$. Constraint (6.31) ensures that the fluid level for classes in the set Π does not go below 0, while constraint (6.32) states that any chosen allocation $\mathbf{x}$ has to satisfy the unary capacity of each machine. The linear program is re-solved at different decision epochs and, when the fluid level in a particular class becomes 0, that class is added to the set Π. Atkins and Chen (1995) create state-dependent priority rules based on the solution of the linear program and compare their performance to traditional policies such as *FCFS* and *SPT* in four environments.[20] They conclude that the performance of fluid policies is at least comparable to classical scheduling heuristics regardless of the distributions of processing or inter-arrival times and the traffic intensities. They state that more research is necessary to highlight the advantages of these policies, which may occur in non-stationary situations or in cases when there are machine breakdowns.

In some cases, it is possible to solve the above linear program for all possible states of the system a-priori and use the resulting information to develop an online scheduling policy. For example, consider a single station serving four classes, with h_1 being the largest of the holding costs. Solving the linear program for any state when there is fluid present in class 1 results in an optimal solution $x_1^* > 0$, $x_2^* = x_3^* = x_4^* = 0$. This solution can be translated into a policy in a straightforward manner: process class 1 jobs while there are such jobs in the system (i.e., give priority to class 1) (Atkins and Chen, 1995). In fact, examining the remaining states of the system and solving the corresponding linear programs leads to the (known to be optimal) $c\mu$ rule. Other ways of using the fluid model to determine a scheduling policy are discussed in Section 6.1.2.3.2.

In many cases, it can be shown that the fluid model is not just an approximation of the system of interest, but rather a formal limit of a sequence of scaled systems as the initial

[20] In Section 10.3, we propose to use one of the problems examined by Atkins and Chen (1995) for investigating the algorithmic integration of queueing and scheduling.

number of jobs in the system goes to infinity. Formal fluid models are of particular importance in stability analysis of queueing systems (Dai, 1995; Dai and Meyn, 1995). Stability analysis consists of identifying conditions under which the number of jobs in the system is guaranteed to remain bounded over time.[21] In the queueing literature, understanding of the stability of a system is considered to be a precursor to more detailed performance questions (Kumar and Meyn, 1995). Moreover, it has been shown that a system may be stable under one scheduling discipline, but not another (Kumar and Meyn, 1995; Bramson, 1994). In spite of its importance as a measure of long-run performance, stability of periodic scheduling approaches has not been addressed within the scheduling community prior to our work. Specifically, we introduce stability into the dynamic scheduling literature in Chapter 8, showing stability of a method based on periodic makespan optimization in two flow shop systems.[22] Concurrently, Tran et al. (2013) and Terekhov et al. (2012d) show stability of periodic scheduling methods in dynamic parallel machine settings.

While several fluid-model heuristics have been developed in the literature, their performance has not been empirically compared to predictive-reactive scheduling methods. For the parallel machine scheduling problem, Tran (2011) and Tran et al. (2013) compare a round-robin policy, which is derived from the solution of a fluid-model linear program, with a scheduling approach based on periodic makespan minimization. Studies of this type for other scheduling problems are necessary in order to understand the strengths and weaknesses of queueing and scheduling approaches, and to create effective hybrids. We discuss this future research direction in Section 10.3.

The Achievable Region Method The achievable region method is a mathematical programming approach for solving stochastic control optimization problems (Federgruen and Groenevelt, 1988; Bertsimas and Niño-Mora, 1996; Stidham Jr., 2002). In traditional mathematical programming formulations of scheduling problems, the goal is typically to determine the start times of the jobs or their positions in the processing sequence on a machine, with constraints being expressed in terms of these variables. In the achievable region method, on the contrary, the decision variables are the performance measures for each class, and constraints are written in terms of these measures. Therefore, while the traditional methodology in scheduling is to find the schedule that optimizes the performance of the system, the achievable region approach is based on finding the optimal performance measure first, and then determining the corresponding scheduling policy. The idea of posting constraints on the performance that is

[21] In contrast, in the predictive-reactive literature, a predictive schedule is called *stable* if it does not change much as uncertainty is realized (Bidot et al., 2009).

[22] The work of Chapter 8 also appears as a paper (Terekhov et al., 2012c) and a technical report (Terekhov et al., 2012b).

achievable by a policy is similar to that of adding lower bounds on the quality of the schedule used in the scheduling literature.

More formally, consider a queueing system with J job types. Let u be a *control rule* that determines how the servers' time is allocated among the arriving job requests. Denote the set of all *admissible* control rules by $\mathcal{U}$. Although the precise definition of admissibility depends on the specific problem being addressed (Stidham Jr., 2002), it is generally required that all control policies in $\mathcal{U}$ are *non-anticipative* (so that a decision can be based only on the current state of the problem and its history) and *non-idling* (so that a machine is not allowed to be idle if there are jobs waiting to be processed on this machine) (Dacre et al., 1999).

Define a J-dimensional *system performance vector* $\mathbf{x}^u = (x_1^u, x_2^u, \ldots, x_J^u)$ associated with every control policy u, where each x_j^u is the expected value of the performance measure for class j (Dacre et al., 1999). For example, if the goal of the problem is to minimize a weighted combination of the expected waiting times for each class, then the performance vector will consist of J expected waiting times, one for each class (Federgruen and Groenevelt, 1988). The set of all admissible performance vectors, $X = \{\mathbf{x}^u, u \in \mathcal{U}\}$, is called the *performance space* or the *achievable region* of the problem. Frequently, the achievable region can be (at least partially) characterized by constraints derived from conservation laws, which state that the amount of work in the system due to a job class i under any policy is at least as much as the amount of class i work under the policy that gives this class priority over all other classes processed by a particular machine.

Suppose that the cost of running the system under the control u is denoted $c(\mathbf{x}^u)$. The scheduling problem of interest is therefore to find a rule, u^{OPT}, that would state how job classes should be assigned to machines in order to optimize the cost of running the system. More formally, this problem is stated by Dacre et al. (1999) as

$$Z^{OPT} = \inf_{u \in \mathcal{U}} \{c(x^u)\}. \tag{6.34}$$

Alternatively, given X, u^{OPT} can be determined by solving the problem

$$Z^{OPT} = \inf_{x \in X} \{c(x)\}. \tag{6.35}$$

In other words, instead of finding the control rule that achieves the smallest cost by solving problem (6.34), we can first find the best possible performance vector by solving (6.35) and then determine the optimal control associated with this performance vector. Thus, the *achievable region approach* is composed of three steps (Dacre et al., 1999):

1. Identification of the performance space X,

2. Solution of the mathematical programming problem (6.35),

3. Derivation of the optimal control rule u from the solution of problem (6.35).

To illustrate this approach, we consider a simplification of our example problem (Figure 6.1) which occurs if we ignore station 2. Specifically, suppose we want to find a non-anticipative and non-idling scheduling policy u for minimizing the long-run holding costs in a two-class $M/M/1$ system. Jobs of class k arrive to the system according to Poisson processes with rate λ_k and are processed by the machine with exponential rate μ_k. For stability, the rate at which work arrives to the system, $\rho_1 + \rho_2 = \lambda_1/\mu_1 + \lambda_2/\mu_2$, is assumed to be strictly less than 1. The problem can be stated as

$$Z^{OPT} = \inf_{u \in \mathcal{U}} \{c_1 E_u(N_1) + c_2 E_u(N_2)\}, \tag{6.36}$$

where $E_u(N_i)$ is the expected steady-state number of class i jobs present in the system operating under policy u, and c_i is the holding cost per job of class i.

In steady state, the amount of work present in this system is not dependent on the control policy u. Thus, the following constraints may be derived:

$$\frac{E_u(N_1)}{\mu_1} + \frac{E_u(N_2)}{\mu_2} = \frac{\rho_1 \mu_1^{-1} + \rho_2 \mu_2^{-1}}{1 - \rho_1 - \rho_2}, \tag{6.37}$$

$$\frac{E_u(N_1)}{\mu_1} \geq \frac{\rho_1 \mu_1^{-1}}{1 - \rho_1}, \tag{6.38}$$

$$\frac{E_u(N_2)}{\mu_2} \geq \frac{\rho_2 \mu_2^{-1}}{1 - \rho_2}, \tag{6.39}$$

for $u \in \mathcal{U}$. The first equation is an expression for the expected amount of work in the system in steady state. The second and third equations follow from the observation that the steady state amount of class 1 work is minimized by giving priority to class 1 over class 2, and vice versa. Defining $x_k^u = E_u(N_k)/\mu_k$, the problem can be written as

$$Z^{OPT} = \inf_{x \in \mathcal{P}} \{c_1 \mu_1 x_1 + c_2 \mu_2 x_2\}, \tag{6.40}$$

where

$$P = \{(x_1, x_2); x_1 \geq \frac{\rho_1 \mu_1^{-1}}{1 - \rho_1}, x_2 \geq \frac{\rho_2 \mu_2^{-1}}{1 - \rho_2}, x_1 + x_2 = \frac{\rho_1 \mu_1^{-1} + \rho_2 \mu_2^{-1}}{1 - \rho_1 - \rho_2}\}. \tag{6.41}$$

The minimum of this problem is attained at the point where class 1 has absolute priority over class 2 if $c_1 \mu_1 \geq c_2 \mu_2$ and at the point where class 2 has priority over class 1 if $c_1 \mu_1 <$

$c_2\mu_2$. Since it can be easily shown that the performance space $X = \{(x_1^u, x_2^u), u \in \mathcal{U}\}$ is equal to the line segment given by P, it follows that the solution to the original control problem is the well-known $c\mu$-rule: serve the job class with the largest $c_k\mu_k$ value first (Dacre et al., 1999). The reader is referred to Section 3 of the paper by Bertsimas et al. (1994) for an achievable region analysis of our example problem with two stations.

The idea of characterizing the region of achievable performance and writing the problem in terms of constraints on performance vectors does not appear to have been used in scheduling, and should be explored in the future. Alternatively, the constraints developed for the achievable region approach in queueing theory may be profitably integrated into dynamic scheduling models.

Comparison of Approximation/Abstraction Models Given the description of the approximations and abstractions used in the queueing literature, a natural question is to determine when one of them is more applicable or more effective than another. To our knowledge, this question has not received much attention in the queueing literature. We can only make the observation that the Brownian model captures variability better than the fluid model. This difference is due to the fact that the Brownian model is based on the Central Limit Theorem, taking into account the first two moments[23] of the given distribution, whereas the fluid model is based on the Law of Large Numbers and thus focuses on the first moment only. Thus, the fluid model can be thought of as a coarser approximation method, but one that is usually easier to work with. In some cases, it is necessary to establish some specific characteristic of the system via the fluid model before a Brownian model can be built, as can be seen in the paper by Kang et al. (2004). For the achievable region to be effective, it should be possible to derive constraints on the optimal performance, such as those shown by Equations (6.37)–(6.39).

6.1.2.3.2 Translation Techniques In order for the approximation/abstraction methods to be useful in practice, their solutions need to be translated into implementable policies. One approach, demonstrated above, is to derive policies that mimic the intuition provided by the solutions to Brownian, fluid or achievable region models. However, there are two main issues with this approach: it is problem-specific, and there are multiple ways of implementing the same intuition, some of which may in reality perform better than the others (see the paper by Maglaras (2000) for a discussion). Such issues have motivated the study of general translation mechanisms and their performance guarantees. We note, however, that the study of such mechanisms has mostly been linked to Brownian and fluid models, and that application of

[23] It is theoretically possible to develop Brownian models of higher order, but results for such models are extremely difficult to obtain, except the cases where there is state-space collapse (see, e.g., the paper by Williams (1998)).

similar principles to derive translation techniques from the achievable region method requires additional investigation.

One approach for translation, presented in the paper by Chen and Meyn (1999), is to initialize the MDP value iteration algorithm (see Section 6.1.2.1) from the solution of the fluid model. The rest of the general translation mechanisms can be classified into discrete-review and continuous-review ones. Both types of methods are similar in nature to periodic scheduling approaches (see Section 6.2) from the scheduling literature: they periodically review the status of the system and solve a resource allocation problem that prescribes how much time should be devoted to each class of jobs until the next review point. Thus, unlike periodic methods from the scheduling literature, these approaches concern the allocation of capacity among job classes rather than individual jobs. Another difference from scheduling methods is that this resource allocation problem is usually modelled as a linear program (LP). In discrete-review methods, the time between two review points is typically large, while in continuous-review ones, it is made as small as possible. Discrete-review methods have the advantage of requiring fewer optimizations, while continuous-review methods should perform better because they take the most up-to-date information into account (Teh, 2000). A similar idea, referred to as a sequential open-loop strategy, has also been used in optimal control theory for preemptive scheduling problems (Nash and Weber, 1982). As mentioned in Section 3.2, the inventory management literature makes a similar distinction between periodic-review and continuous-review methods.

One general concept that is extensively used within both discrete-review and continuous-review paradigms is that of safety stocks. Safety stocks state the amount of material that should be present in the queues in order to avoid starvation of resources. Thus, they prevent idleness of resources, which should be avoided to maintain stability (Meyn, 2008). Since, as discussed in Section 6.1.2.3.1, stability has not received much attention in the scheduling literature, neither has the use of safety stocks. Interestingly though, in the scheduling literature, Branke and Mattfeld (2005) demonstrate that avoiding early idleness is essential for achieving good long-run performance. Also, since the notion of safety stocks is used in inventory management, it could be useful for the dynamic version of the assembly scheduling problem that we propose to study in future work (see Chapter 11).

Discrete Review We discuss several discrete-review approaches: BIGSTEP, reward-based policies and trajectory-tracking policies. According to Meyn (2008), discrete-review policies provide "the most natural technique to translate a policy based on the fluid model or some other idealized model for application in a physical network" (p. 129).

BIGSTEP The first discrete-review approach developed in the queueing literature is called

BIGSTEP, and was proposed by Harrison (1996) as a translation mechanism from solutions of Brownian approximation models, discussed in Section 6.1.2.3.1, to tactical allocations of server time. The BIGSTEP approach is defined by two parameters: l, the length of time between two review time points, and $\boldsymbol{\theta} = (\theta_k)$, the threshold (planned safety stock) parameter vector (Teh, 2000). At every review time point ($\tau = 0, l, 2l, \dots$), the current queue lengths $\boldsymbol{q} = (q_k)$ are observed, and an LP is formulated. This LP is called the BIGSTEP planning problem. Its solution specifies the amount of time that should be spent on processing each job class in the period $[\tau, \tau + l]$, subject to the constraint that the number of jobs in each queue k cannot fall below θ_k.

If the corresponding Brownian approximation model allows for a pathwise solution,[24] then the BIGSTEP planning problem is equivalent to the fluid model presented in Equations (6.26)–(6.29) with time 0 corresponding to the current review point τ, and t replaced by $\tau + l$. Since this model requires first-moment data only, and since most dynamic control problems allow for a pathwise solution, the BIGSTEP approach can be applied to a wide variety of problems (Teh, 2000). If the Brownian approximation model does not have a pathwise solution, then a second term, representing the future expected cost of server idleness, is added to the objective in Equation (6.26). This additional term is $\boldsymbol{p}\ \boldsymbol{U}(\tau + l)$, where $\boldsymbol{p}$ is the penalty rate vector obtained by solving the appropriate Brownian control problem.

The idea behind the BIGSTEP approach is to ensure that the number of jobs present in each of the buffers is large enough relative to the length of the planning horizon so that the actual sequence in which the jobs are processed within the next l time units is not important from a tactical viewpoint (Harrison, 1996). In order to implement these tactical allocations at the operational level, one can choose an arbitrary sequence in which to consider classes, and process each class for the amount of time specified by the LP solution; a simple policy such as *FCFS* can be used to sequence jobs within each class (Harrison, 1996). The assumption that the details of the sequencing of individual jobs are "irrelevant" may seem unintuitive from a scheduling perspective. However, one needs to keep in mind that the goal of BIGSTEP is to optimize the long-run performance of the system. Naturally, in realistic applications, both short-run and long-run performance measures are of interest, which motivates the study of combining the BIGSTEP approach with detailed scheduling models of each time period. We refer the reader to the paper by Harrison (1998) for an example of the application of the BIGSTEP approach to a parallel machine system.

Reward-based Policies Maglaras (1999) extends the BIGSTEP approach to a family of

[24] A pathwise solution is a dynamic scheduling policy that, in the heavy traffic limit, minimizes the instantaneous cost rate at every time point with probability one (Teh, 2000).

discrete-review methods derived from dynamic reward functions, which associate a positive value $r_k(\boldsymbol{q})$ with every queue length vector $\boldsymbol{q}$ and each job class k. Given strictly positive real-valued functions $r(\cdot)$ and $l(\cdot)$ and a K-dimensional vector $\boldsymbol{\beta}$, at each review point t_j, the controller calculates the length of the review period and the target safety stock level to be achieved at the end of the period. A linear program which is based on the fluid representation of the system is then solved to determine the amount of time that should be allocated to each class k until the next review time point. The objective function is defined in terms of the dynamic reward function $r(\cdot)$, which represents the reward rate for spending time on processing each class. For additional details, the reader is referred to Section 3 of the paper by Maglaras (1999). Note that, unlike in the BIGSTEP method, the length of the review time period in the reward-based discrete-review methods grows as a function of the state size. As this happens, the approximation described by the equations of the linear program becomes more accurate (Maglaras, 1999).

Trajectory-tracking Policies The trajectory-tracking family of policies is also an extension and a generalization of the BIGSTEP method (Maglaras, 2000). It is defined by the trajectory mapping Ψ, a function l and K-dimensional vector $\boldsymbol{\beta}$. The parameters l and $\boldsymbol{\beta}$ have the same meaning as above. The trajectory map Ψ describes the desired behaviour to be tracked. One option for the trajectory map is the solution of the corresponding fluid model (Maglaras, 2000). The approach consists of three stages: initialization, planning and execution.

The initialization stage involves setting the length of the review periods and the safety stock levels: $l = l(|\boldsymbol{Q}(0)|)$ and $\theta = \beta l$, where $\boldsymbol{Q}(0)$ is the state of the system at time 0. At the planning stage, the status of the system is reviewed at times $0 = t_0 < t_1 < t_2 < \ldots$; the queue state observed at each review period is $\boldsymbol{q} = \boldsymbol{Q}(t_j)$. In step 1 of the planning stage, a target state z for the end of the review period is chosen as $z = \theta + |Q(0)|\Psi(\bar{l}; (\bar{q} - \bar{\theta})^+)$, where $\bar{l} = \frac{l}{|Q(0)|}$ and $(\bar{q} - \theta)^+ = \max\{0, \max\{0, (\boldsymbol{q} - \boldsymbol{\theta})^+\}/|\boldsymbol{Q}(0)|\}$. In step 2, the time allocations that will steer the state from q to z are computed. The reader is referred to the paper by Maglaras (2000) for details of this step.

In the execution stage, the following quantities are defined:

$$p_k = \left\lfloor \frac{x_k}{m_k} \right\rfloor, \quad k = 1, \ldots, K, \tag{6.42}$$

$$u_i = \max\{0, (l + t_\theta - (Cx)_i)^+\}, \quad i = 1, \ldots, S. \tag{6.43}$$

Firstly, p_k jobs are processed sequentially from each class $k \in C_i$. As in the BIGSTEP method above, it is implied that the order of processing of these jobs does not matter. Secondly, each machine i idles for $\min\{u_i, \max\{0, (l + t_\theta - d_i)\}\}$, where d_i is the time taken to complete

all jobs at machine i. The start of the next processing period is then defined as $t_{j+1} = t_j + \max\{l + t_\theta, d_1, \ldots, d_S\}$. The resulting policies are asymptotically optimal under fluid scaling, and guaranteed to be stable if the traffic intensity of each station is less than one (Maglaras, 2000). A similar idea of trajectory-tracking policies is explored by Meyn (2001).

The idea of trajectory tracking has not been explored in the scheduling literature. It would be interesting to examine it from two perspectives: firstly, predictive-reactive methods could be developed which aim to track the solution of the fluid model as do the trajectory-tracking policies in queueing, and, secondly, for static problems, there may be problem relaxations, such as preemptive versions of non-preemptive problems, that can be tracked.

Continuous Review We discuss several continuous-review approaches: discrete-review derived, trajectory tracking and maximum pressure policies.

Discrete-review Derived Policies The work of Teh (2000) extends the idea of the BIGSTEP method to discrete-review derived (DRD) continuous-review policies. As in the BIGSTEP method, the goal of a DRD policy is to determine the amount of time that should be devoted to processing a particular class of jobs in the coming time period of length l. However, in the BIGSTEP approach, the value of l needs to be large, while Teh (2000) proposes to make l as small as possible without forcing the time allocation variables to take on negative values. Moreover, the value of l may change when the allocation specified by the LP solved at each review point changes. The desired queue state at the end of the period, used in the LP formulation, is obtained by solving a Brownian control problem (Teh, 2000).

Trajectory Tracking Policies Continuous-review tracking policies are proposed by Bäuerle (2000) and by Maglaras (2003). Maglaras (2003) states that such policies consist of observing, at each time point t, the vector of current queue lengths, $\boldsymbol{q}$, and choosing $z(\boldsymbol{q})$, a target position of the system at some time point $t + l$, and $v(\boldsymbol{q})$, an allocation vector describing how to divide the available processing capacity among the job classes so as to guide the evolution of the system from $\boldsymbol{q}$ to $z(\boldsymbol{q})$. A Brownian model is used in order to define $z(\boldsymbol{q})$, while reasoning based on fluid models (discussed in Section 6.1.2.3.1) is employed for computing the decisions $v(\boldsymbol{q})$ that would lead the system to $z(\boldsymbol{q})$. The idea behind this policy type is similar to that of state-dependent sequencing rules, except instead of priorities being switched after every state transition, capacity allocations to different jobs classes are changed.

Paschalidis et al. (2003; 2004) define target-pursuing policies, which are similar to trajectory-tracking policies, but utilize the solution of an achievable region problem to set one of its parameters. Let $\mathbf{n}(t)$ be the vector representing the number of jobs in each class of the sys-

tem. At each time t and for a finite review interval Δt, a target-pursuing policy minimizes $\mathbf{E}[||\mathbf{n}(t+\Delta t) - \theta|| \; |\mathbf{n}(t)]$ for some norm $||\cdot||$ and target θ; that is, given the number of jobs in each class at time t, $\mathbf{n}(t)$, the policy minimizes the expectation (with respect to the probability distribution of $\mathbf{n}(t+\Delta t)$) of the norm of the difference between the number of jobs present in each class at the next review point and the target (Paschalidis et al., 2004). Paschalidis et al. (2004) show that setting $\theta = \mathbf{w}^*$, where $\mathbf{w}^*$ is the lower bound on the optimal performance obtained via an achievable region approach, often results in good performance. For example, in the context of the problem of Figure 6.1, the best-performing target-pursuing policy is numerically shown to be within 0% to 2.3% of the best-performing policy for various loads for the weighted sum of mean queue lengths objective.

Maximum Pressure Policies Maximum pressure policies (Dai and Lin, 2005) are generalizations of the well-studied MaxWeight policy (Tassiulas and Ephremides, 1992; Tassiulas and Bhattacharya, 2000; Andrews et al., 2004; Stolyar, 2004). A maximum pressure policy looks at the system state at activity completion and job arrival epochs, and determines the allocation vector $a = (a_j)$ that maximizes the total network pressure (Dai and Lin, 2005). Each entry a_j specifies the proportion of time that should be spent on processing activity j (by the appropriate server(s)) that remains valid until the next review point. More formally, one has to choose $a^* \in \arg\max_{a \in E(t)} p(a, Z(t)) = Z(t) \cdot Ra$, where $Z(t)$ is the vector of buffer levels at time t, $R = (R_{ij})$ is the input-output matrix in which each entry R_{ij} is interpreted as the average amount of buffer i material consumed per unit of activity j, $E(t)$ is the set of extreme points of the feasible region of all possible maximum pressure policies, and $\cdot$ denotes the inner product between the two vectors. The set $E(t)$ depends on whether resources can process one or several jobs at a time (i.e., whether processor splitting is allowed or not) and on whether preemptions are allowed or not. Maximum pressure policies are different from the above-mentioned policies since they directly specify which job should receive processing next. They are semi-local since the sequencing decisions for each server are based on both the state of the buffers for which the server is responsible and the state of the downstream buffers. One of their advantages is that they do not use any arrival rate information, which can be hard to estimate in practice (Dai and Lin, 2005).

The work discussed above is aimed at deriving policies which are asymptotically optimal and stable, and, given the long-run nature of these objectives, they prescribe the proportion of each server's capacity that should be spent on processing a particular class. In fact, it can be shown that the detailed sequencing decisions are irrelevant for achieving these long-run objectives. However, in real scheduling problems, one typically needs to address short-run

performance measures in addition to long-run ones. Therefore, it is worthwhile to investigate different operational-level scheduling policies that can also respect the tactical allocations provided by methods of this section.

The solutions to approximations/abstractions may also be used in the form of bounds when operational-level schedules need to be constructed. Moreover, a model that specifies high-level allocations could be used as part of a problem decomposition method such as the logic-based Benders method of Hooker (2005); this idea has not been explored in the literature.

6.1.2.4 Summary

We surveyed the queueing theory literature related to scheduling, dividing it into three categories. The first is based on MDP models, which are, theoretically, powerful enough to represent any scheduling problem. However, even when focusing on classes rather than individual jobs, MDP models become intractable as the size of the problem (i.e., the number of classes) increases. To overcome the difficulties of large MDP models, the queueing literature has chosen either to restrict the types of policies that are considered or to use approximations or abstractions.

Section 6.1.2.2 reviewed the second category of models, those possessing a special structure. These include priority queues, polling systems, vacation models and bandit models. Section 6.1.2.3 presented the general framework for the use of approximations and abstractions which consists of four steps: modelling and development of the approximation/abstraction, solution of this approximation/abstraction, derivation of an implementable scheduling policy based on this solution, and asymptotic analysis of the optimality of the resulting policy. The approximations/abstractions used in the first two steps include Brownian models, fluid models and the achievable region approach. Methods for the derivation of implementable scheduling policies are presented in Section 6.1.2.3.2.

One common theme throughout the literature reviewed above is that, since queueing theory cannot deal with individual job characteristics, scheduling can be used to sequence jobs within each class of jobs. Doing so is feasible in applications where the processing times of jobs can be accurately estimated upon arrival (e.g., in manufacturing). We briefly investigate the improvements in performance due to within-class sequencing in Chapter 7 and propose further examination of this research direction in Chapter 10.

We refer the reader who is interested in a more mathematically-rigorous coverage of scheduling models developed within queueing theory to the comprehensive book by Meyn (2008).

6.2 Dynamic Scheduling

Recall that in Part II of this dissertation, we investigated the notion that scheduling problems are related to other decision-making problems in their environment, focusing specifically on the integration of inventory management decisions with scheduling. In the first chapter of that part of the dissertation, Chapter 3, we therefore reviewed fundamental combinatorial scheduling notions as well as the literature on scheduling in assembly environments and scheduling with inventory constraints. In the current part of the dissertation, our focus is on addressing the dynamism of real scheduling problems by integrating queueing theory and scheduling. Above, we reviewed the queueing theory work related to scheduling. Now, we survey the work on dynamic scheduling that has been developed by the scheduling community.

According to Thomas and Szczerbicka (2007), dynamic scheduling methods can be divided into three branches.[25] The first is based on dispatching rules, which, as we discussed in Section 6.1.2.2.1, are equivalent to priority queueing disciplines. In spite of this equivalence, queueing and scheduling have very different views of these rules. In the scheduling literature, the focus is on experimental evaluation of dispatching rules under various conditions (Panwalkar and Iskander, 1977; Haupt, 1989). Thomas and Szczerbicka (2007) state, in addition, that such schedulers "lack the ability to plan into the future" (p. 2). Queueing theory, in contrast, usually studies long-run theoretical performance of priority disciplines given distributional information about the system. Integrating the work done on priority queueing disciplines and dispatching rules is therefore a promising direction for future work.

The second category of dynamic scheduling methods views the problem as a collection of linked static sub-problems and utilizes the developments of combinatorial optimization (Thomas and Szczerbicka, 2007). Examples of work on such methods include the papers by Bierwirth and Mattfeld (1999) and Fromherz (2001). The most general framework within this category, however, is the predictive-reactive framework of Bidot (2005) and Bidot et al. (2009). The predictive component of a predictive-reactive approach constructs a baseline schedule for a short period of time into the future based on known job characteristics. Frequently, the objective in creating a predictive schedule is robustness. A solution-robust schedule is one in which the start times of jobs do not change significantly due to changes in the environment, while in a quality-robust schedule, performance is insensitive to disruptions (Herroelen and Leus, 2005). In order to create predictive schedules, Leon et al. (1994) incorporate a robustness term into the objective function, Davenport et al. (2001) extend activity durations, while Dubois et al. (1996) and Beck and Wilson (2007) minimize the possibility and the probability, respectively,

[25] For an alternative and more detailed classification, see the paper by Suresh and Chaudhuri (1993). For dynamic scheduling in process industries, we refer the reader to the paper by Li and Ierapetritou (2008).

that the schedule performs worse than a certain threshold.

The reactive component states how the schedule should be modified in the case of disruptions. The simplest reactive methods are rules that can be used to repair a disrupted schedule, such as the right shift rule, which shifts the start times of all jobs forward after a change in the environment occurs (Smith, 1994). More complex reactive approaches include rescheduling of all or a subset of the jobs that are present but whose processing has not yet started (Sabucuoglu and Bayiz, 2000). When rescheduling is performed, the typical goal is to ensure that the newly generated schedule differs as little as possible from the original schedule. Thus, El Sakkout and Wallace (2000) develop methods for minimizing the total absolute deviation in the start times of the jobs in the two schedules, while Bean et al. (1991) reschedule in a way that, at some point in the future, matches the previous schedule with the new schedule exactly.

Generally, scheduling methods based on periodic optimization of static problems do not consider long-run performance measures. Thus, Branke and Mattfeld (2002, 2005) propose anticipatory scheduling, which is based on a rolling horizon approach that anticipates future needs by including a secondary objective within each static scheduling sub-problem. This secondary objective, called the *flexibility term*, penalizes early idle times of the machines, thus preserving machine capacity for future jobs. They show that anticipatory scheduling can improve the system's performance with respect to the total tardiness objective.

The third category, according to Thomas and Szczerbicka (2007), utilizes stochastic optimization and sampling. A general framework within this category is online stochastic combinatorial optimization (OSCO) (Van Hentenryck and Bent, 2006). In OSCO, problems are solved by online anticipatory algorithms, which are composed of a method for sampling the distribution of future events and a method for solving deterministic problems which correspond to possible realizations of the future evolution of the system. Thomas and Szczerbicka (2007) also include the paper by Chang et al. (2000) in this third category. Chang et al. (2000) address an oversubscribed dynamic scheduling problem with tasks of unit duration by using a discrete-time partially-observable MDP (POMDP) model and sampling. From our perspective, this paper is interesting because it combines a model that can be classified as being part of the queueing literature (POMDP is a variation of the MDP model discussed in Section 6.1.2.1) with an approach from the dynamic scheduling literature, i.e., sampling; it is also one of the rare papers in the scheduling literature that mentions queueing work and divides jobs into multiple classes.

A review of these three types of dynamic scheduling methods leads to the natural question of how they compare to each other. Thus, Montana (2005) compares dispatching rules and combinatorial optimization methods (though note that this comparison was developed prior to the formalization of the predictive-reactive framework by Bidot et al. (2009)). In particular,

he evaluates the trade-off between the time required for the construction of a schedule and the resulting schedule's quality. Thomas and Szczerbicka (2007) further investigate this trade-off, including the sampling-based algorithm of Bent and Van Hentenryck (2004) in their analysis. More studies of this type are necessary to obtain a full understanding of dynamic scheduling. In particular, future work needs to compare methods from the predictive-reactive framework, OSCO, dispatching rules, various queueing policies and queueing/scheduling hybrids.

Two additional fields of study that are closely related to dynamic scheduling are: online scheduling and real-time scheduling. While online scheduling addresses the same problems as dynamic scheduling, it is different since, as stated by Pruhs et al. (2004), it considers a relatively small set of algorithms and focuses on obtaining performance guarantees. The algorithms listed by Pruhs et al. (2004) that have received the most attention in the online scheduling literature can be seen as priority disciplines or dispatching rules discussed in Section 6.1.2.2.1. An example of a method that is widely used to analyze online algorithms is competitive analysis, which aims to determine whether a particular algorithm A is c-competitive for objective function f, i.e., whether $f(A, I) \leq cf(OPT, I) + b$ for any instance I, a fixed constant b and OPT being the optimal offline algorithm for the problem (Pruhs et al., 2004). In Section 11 of his survey paper, Pruhs (2007) briefly compares competitive analysis with stochastic analysis: one of the main differences between the two is that competitive analysis does not require distributional knowledge, making it complementary to work in queueing theory. We see value in unifying the results in the online scheduling and priority queues literatures. In future work, we may also use online scheduling analysis for determining the value of distributional knowledge in dynamic problems. Real-time scheduling is different from both dynamic scheduling and online scheduling because it focuses on systems that have explicit timing (i.e., deadline) requirements, which may be deterministic or probabilistic (Sha et al., 2004). The reader is referred to Part IV of the *Handbook of Scheduling* (Leung, 2004) and the papers by Sha et al. (2004) and Burns (1991). See the paper by Lehoczky (1996) for the application of queueing theory to the real-time scheduling paradigm.

6.3 Summary of Our View on Queueing and Scheduling

In a static and deterministic environment, scheduling corresponds to determining the start times of a given set of jobs that has to be processed on one or several machines. This problem is combinatorial in nature, and involves a finite set of jobs with known characteristics and a finite time period whose length is equivalent to the schedule's horizon. In reality, such a problem corresponds to only one possible scenario that the scheduler may be faced with at a particular point in time, and the schedule created based on this scenario is likely to be valid for only a short

horizon. In fact, the scenario represents a realization of many stochastic processes that govern the evolution of the system in question (e.g., inter-arrival times, processing times, etc.). From the point of view of scheduling in dynamic and stochastic settings, static and deterministic scheduling deals with short-run decisions based on a local, myopic view of the environment, its combinatorial structure and realized uncertainty. The majority of the classical scheduling literature deals exactly with such problems.

A static and stochastic problem, in which the set of jobs to be processed is fixed but the characteristics (e.g., processing times) are stochastic also deals with a short time horizon and adopts a myopic view of the system. Such a problem represents a particular realization of the process of job arrivals at a specific point in time, but considers stochastic properties of the jobs. Thus, the problem models a variety of scenarios for a particular set of jobs, and the schedule has to be constructed in a way that not only deals with the combinatorics of the problem, but also with its stochastic nature: since it is impossible to create a schedule that would be of high quality in all scenarios, the goal in such problems is usually to achieve good (or optimal) performance in a probabilistic sense (e.g., in expectation). Work on these problems is known as stochastic scheduling (e.g., see the books by Pinedo (2003) and Baker and Trietsch (2009)), and methods for stochastic scheduling are based on approaches for deterministic problems as well as some probabilistic reasoning (Righter, 1994; Beck and Wilson, 2007).

The problem of scheduling in a dynamic environment involves a long time horizon and has to somehow deal with all the possible realizations of the job arrival process and of the job characteristics. The ultimate goal in solving this problem is to construct a schedule that would be optimal for the specific realization that actually occurs. The quality should be close to that of the schedule that could have been constructed if all of the uncertainty had been revealed *a priori*. Clearly, this is a difficult task, because to make a decision, one can use only the information that is known with certainty at that particular decision point and the stochastic properties of scenarios that can occur in the future. In addition, the effect of the decision on both short-run and long-run performance has to be considered. To deal with such problems, queueing theory and scheduling have adopted different approaches.

Queueing theory has taken the viewpoint that, since it is impossible to create an optimal schedule for every single sample path in the evolution of the system, one should aim to achieve optimal performance in some *probabilistic* sense (e.g., in expectation) over a long time horizon. This goal could be attained by construction of a policy based on the global stochastic properties of the system. For example, a policy could specify how start time decisions should be made each time a new job arrives or a job is completed. However, the schedule resulting from such a policy, while being of good quality in expectation, may be far from optimal for the particular realization of uncertainty that occurs. Moreover, queueing theory generally studies systems

with simple combinatorics, as such systems are more amenable to rigorous analysis of their stochastic properties.

In the scheduling community, a dynamic scheduling problem is generally viewed as a collection of linked static problems. This viewpoint implies that methods developed for static scheduling problems become directly applicable to dynamic ones. Such methods can effectively deal with complex combinatorics and can optimize the quality of the schedules for each static sub-problem. However, they tend to overlook the long-run performance and the stochastic properties of the system, with notable exceptions being the work on anticipatory scheduling (Branke and Mattfeld, 2002, 2005) and online stochastic combinatorial optimization (Van Hentenryck and Bent, 2006).

Queueing theory methods for scheduling are based on reasoning about the stochastic processes underlying the system, and aim to achieve good performance, in a probabilistic sense, over the variety of possible scenarios that may occur. However, they are not able to deal with individual job characteristics. They can be only approximate for problems with a complex combinatorial structure or asymptotically optimal when the number of jobs in the system becomes so large as to make the combinatorics irrelevant. Queueing methods may also result in policies that, at a particular point in time, may suggest a decision that may not be desirable from a realistic perspective: for example, a policy may state that jobs of priority 1 should always be scheduled ahead of jobs of priority 2, leading to a build-up of priority 2 jobs in the system and to the dissatisfaction of the customer corresponding to priority 2 jobs. In contrast, predictive-reactive methods are based on solving static problems and then using reactive approaches online in case there are disruptions that have not been anticipated by the baseline schedule. Their focus is on individual job characteristics, and their advantages come from their ability to optimize the short-run performance of the system.

From the dynamic scheduling perspective, the strengths of queueing theory and scheduling can be seen as being complementary. We believe that integrating the two may allow us to develop both a better understanding of, and more effective solution techniques for, scheduling in dynamic and stochastic environments, because we can more effectively reason about the long run using queueing theory and about the short run using scheduling methods. In this dissertation, we propose one possible framework for integration of the two areas.

6.4 Conclusion

As stated in Chapter 2, real scheduling problems are combinatorial, dynamic and uncertain, and related to other decision-making processes of an environment. In this part of the dissertation, we study scheduling problems that are both combinatorial and dynamic by combining queueing

theory and scheduling. Therefore, in the current chapter we surveyed the literature on queueing theory, starting with fundamental notions and then proceeding to discuss methods specifically developed for scheduling. We then briefly surveyed work on dynamic scheduling developed within the scheduling community. Furthermore, we provided our view on various types of scheduling problems and their relationship to queueing theory, which serves as a motivation for the rest of this part of the dissertation.

Our investigation of queueing and scheduling takes place on three levels, conceptual, theoretical and algorithmic. In the next chapter, we provide one example of work on the conceptual level. We describe two flow shop settings, and demonstrate how integration of concepts and ideas from queueing and scheduling leads to the development of new insights about dynamic scheduling. In Chapter 8, we demonstrate how queueing and scheduling can be integrated on a theoretical level. Specifically, we prove stability of periodic scheduling methods based on makespan minimization. In Chapter 9, we continue the investigation of the theoretical level by proposing to use fluid limit analysis as a tool for analyzing scheduling algorithms. Chapter 10, the final chapter of Part III, discusses future work on both the conceptual and theoretical levels. In addition, it provides one possible framework for integration of queueing and scheduling on the algorithmic level, and gives two examples of how this framework can be applied.

Chapter 7

Conceptual Integration of Scheduling and Queueing Theory

In this chapter, we analyze two dynamic flow shop environments: a novel polling system that is an extension of systems traditionally examined in queueing theory and a dynamic two-machine flow shop, which is important in scheduling research.[1] In both cases, the objective is to minimize the mean flow time of jobs (i.e., the time between the arrival of a job to the system and its completion time). Our experiments suggest that in the polling system, a scheduling method that optimizes the makespan at each decision point provides the best performance, while in the dynamic flow shop, an approach based on minimizing the mean flow time directly at each decision point works better. We provide a mathematical characterization of these performance differences in terms of the structural properties of the scheduling problems. We demonstrate that in the polling system there is a conflict between short-run and long-run objectives, while in the dynamic flow shop, optimizing the short-run objective periodically is consistent with the long-run performance goal.

This chapter demonstrates that combining problem settings, ideas and concepts from queueing theory and scheduling can lead to novel insights about scheduling in dynamic environments. Specifically, we obtain a new understanding of the trade-off between short-run and long-run objectives in dynamic scheduling. We also show that periodic scheduling methods can perform better than queueing-based dispatching rules for optimization of long-run performance, although the choice of objective function for each sub-problem may not be obvious.

We start this chapter with formal descriptions of the two systems of interest. In Section 7.2, scheduling approaches for these systems are presented, together with experimental results.

[1] The work presented in this chapter has been published as a workshop paper (Terekhov et al., 2010), has been included in a conference paper (Terekhov et al., 2012c), and is part of a working journal paper (Terekhov et al., 2012d). The material included in the conference paper (Terekhov et al., 2012c) is used with permission of the copyright holder, the Association for the Advancement of Artificial Intelligence ©, and the paper's co-authors.

A detailed analysis of the performance differences between two of these methods is given in Section 7.3. We compare the two systems of interest and discuss our assumptions in Section 7.4. Section 7.5 concludes the chapter.

7.1 Problem Settings

We study two related dynamic scheduling environments: a two-machine flow shop and a polling system with a flow shop-like server. The goal, in both systems, is to assign start times to all jobs in a way that minimizes the mean job flow time over a long time horizon. A job's flow time is defined as the difference between the job's completion time on the second machine and its arrival time to the system.

7.1.1 Dynamic Flow Shop

In a two-machine dynamic flow shop, jobs arrive according to some stochastic process over time and must be processed first on machine 1 and then on machine 2. We assume that the inter-arrival distribution is general with rate λ, while the processing time distributions for machine 1 and 2 are general with rates μ_1 and μ_2, respectively. Thus, the load for machine 1 is $\rho_1 = \frac{\lambda}{\mu_1}$ and the load for machine 2 is $\rho_2 = \frac{\lambda}{\mu_2}$. Both ρ_1 and ρ_2 are assumed to be less than 1, as this is a necessary and sufficient condition for ensuring that the number of jobs in the system remains bounded over time.[2]

Processing times of a job on both machines become known at the instant of its arrival to the system. Both machines are of unary capacity. Preemptions are not allowed. The queues in front of machine 1 and machine 2 are both assumed to be of infinite size. In the queueing literature, this system is known as a tandem queue (Towsley and Baccelli, 1991) or network of queues in series (Gross and Harris, 1998).

7.1.2 Polling System

Polling systems are also dynamic environments: jobs arrive at random points in time and, under one set of queueing assumptions, the processing times of these jobs become known upon their arrival. As already stated in Section 6.1.2.2.2, polling systems usually consist of a single server and multiple job types. Each arriving job has to wait for processing in the queue corresponding to its specific type. The server visits these queues in a particular order and serves a subset of the jobs during each visit. A variety of policies for the order in which the queues should be visited

[2]It is a necessary condition since each machine individually behaves as a single-server queue. A proof of its sufficiency is presented in Section 8.2.

and for deciding the number of jobs that should be served on each visit have been considered in the literature (Levy and Sidi, 1990; Takagi, 2000). We assume that the server visits the queues in a cyclic manner. During each visit, the server employs a gated discipline: it processes all jobs that are present at the time of the server's arrival to the given queue. Preemptions are not allowed. Unlike in standard queueing models, we assume that the server of the polling model consists of two machines in series (a two-machine flow shop).

We assume that the inter-arrival distribution for each queue b, $b = 1, 2, \ldots, B$, is general with rate λ_b. The processing time distributions are general with rates μ_{1b} and μ_{2b} for machines 1 and 2, respectively. We assume that $\rho_{1b} = \frac{\lambda_b}{\mu_{1b}} < 1$ and $\rho_{2b} = \frac{\lambda_b}{\mu_{2b}} < 1$ for all b, as this assumption is necessary to ensure that the number of jobs in the system stays bounded.[3] Due to the assumption of a gated policy, upon arrival to a queue, the server is faced with a static two-machine flow shop scheduling problem.

7.1.3 Discussion of Assumptions

In this section, we discuss the assumptions made in the above problem settings that are motivated by the queueing literature as well as those that are motivated by the scheduling literature. We also justify why we look at dynamic flow shops and polling systems.

7.1.3.1 Available Information

The above systems are based on a combination of assumptions from the queueing theory literature and from the scheduling literature. In queueing theory, it is typically assumed that jobs arrive according to some stochastic process, while in classical scheduling models, a set of jobs is present at time 0 and there are no arrivals. As in queueing, we assume that we know the distribution of the inter-arrival times, but our scheduling framework (described in the following section) takes advantage of the fact that if we look at the system at any one point in time, we will see a static set of jobs, i.e., a classical scheduling model.

Similarly, queueing models usually assume that the distribution of job durations is known, but the actual processing time of a job is not available until its completion time.[4] In the classical scheduling literature, exact processing times of jobs are assumed known prior to construction of a schedule, and durations of jobs processed on the same machine are, in general, not identical. The queueing assumptions can be justified by many application areas in which data collected

[3] It is a necessary condition since each machine individually can be viewed as a single-server queue. A sufficient condition for stability is presented and proven in Section 8.3.

[4] One exception is the *deterministic* distribution, under which the durations of all jobs are assumed to be identical.

over time can be used to determine the distributions. The scheduling assumptions are motivated by manufacturing and computer applications. In manufacturing, several configurations of the same product may be processed on a machine. There may be so little variance in the production time of a particular configuration as to make this time essentially deterministic; each configuration may require a different amount of processing time. In computer applications, different users may submit tasks with varying demands for the same resource. Characteristics of these tasks, such as file sizes, may be used to accurately estimate the amount of processing required.[5] To link the queueing and scheduling assumptions, we assume that the processing times that become known upon arrival are realizations drawn from the corresponding distributions.

7.1.3.2 Problem Settings

The polling model was inspired by the need to create an initial theoretical basis for studying the integration of ideas from queueing theory and scheduling, and it provides one of the simplest possible environments for doing so. However, there has been some previous work on polling models with coupled servers. Specifically, Borst (1995) and Browne and Weiss (1992) both study systems in which there are multiple parallel machines serving the queues as one unit. Such models are motivated by, for example, distributed systems in which jobs enter queues that correspond to "front-end" systems and are processed by a server composed of multiple computers connected by some communication medium ("back-end" systems) (Borst, 1995). Our model assumes that, instead, the computers are divided into clusters which process sub-tasks of jobs in a flow-shop manner; the server switches between queues as one unit since each queue may correspond to a user who gets exclusive rights to the resources once its service is started. Katayama (1992) also examines a tandem processing structure combined with polling, by looking at cyclic-service tandem queueing systems with multi-class customers. Their model, unlike ours, assumes that all jobs arrive at queue 0 first and then either return to queue 0 or move to the second stage of processing at one of $\mathcal{N}$ subsequent queues; there is a single server that attends queues 0 to $\mathcal{N}$ in a cyclic order.

Our model also possesses characteristics of manufacturing environments in which each queue corresponds to a unique product type or to a specific customer. Each product requires processing on the given set of machines as in a flow shop. In this application, the fact that the first machine may be idle while the second machine is finishing jobs from the current set can be due to a joint resource (e.g., space that is used to store products that have been processed

[5]The problem setting where jobs arrive to the system dynamically and job processing times become known with certainty upon arrival is known as the *online-time* setting in the online scheduling literature (Pruhs et al., 2004). One application mentioned by Pruhs et al. (2004) is a web server serving static documents.

on machine 1, or a conveyor belt that cannot be switched to a different queue until all of the current items have been processed) or the necessity to reprogram/reconfigure both machines at the same time before switching to the next queue. Extending our model to include setup times can naturally model systems with multi-stage production processes where the whole production area needs to be cleaned.

Unlike the polling model with a two-machine flow shop server, the dynamic flow shop environment is not novel. Both static (Hejazi and Saghafian, 2005; Della Croce et al., 1996; Xia et al., 2000; Park et al., 1984; Dudek et al., 1992; Gupta and Stafford Jr., 2006; Framinan et al., 2005) and dynamic (Sarper and Henry, 1996; Park, 1988; El-Bouri et al., 2008) flow shops have been extensively studied in the scheduling literature. Tandem queues (i.e., flow shops under typical queueing assumptions) have received significant attention in the queueing literature (Towsley and Baccelli, 1991; Gross and Harris, 1998; Johri and Katehakis, 1988; Andradóttir and Ayhan, 2005). It should be noted that for both the dynamic flow shop and the polling system, a static deterministic two-machine flow shop scheduling problem is embedded at every time point. However, the higher-level structure of multiple queues in the polling system makes the two systems different. We can also think of scheduling the dynamic flow shop as being the problem that the server would encounter if it was serving a particular queue of the polling system under an exhaustive[6] discipline.

7.2 Scheduling in Polling Systems and Dynamic Flow Shops

We can schedule jobs in the systems described above via periodic scheduling strategies: at a given point in time, we review the jobs present in the system or a particular queue of the system, create a schedule for these jobs, and once this schedule is executed, review the status of the system again. We examine two queueing-based and two scheduling-based methods for creating each sub-schedule in both polling systems and dynamic flow shops. The time at which scheduling happens, however, is different in the two environments. In the polling system, the server switches from one queue to the next only after all of the jobs present upon its arrival to the queue have been processed on *both* machines; the validity of this assumption is discussed in the previous section. The start time of every new sub-problem is equal to the completion time of the last job in the previous sub-problem, as shown in Figure 7.1. In a dynamic flow shop without a polling structure, such an assumption is unreasonable since it would create unnecessary idle time on machine 1. Thus, in a dynamic flow shop, scheduling is

[6]Under the *exhaustive* discipline, the server processes jobs from one queue until this queue becomes empty. Refer to Section 6.1.2.2.2 for a review of the literature related to different queue service disciplines in polling systems.

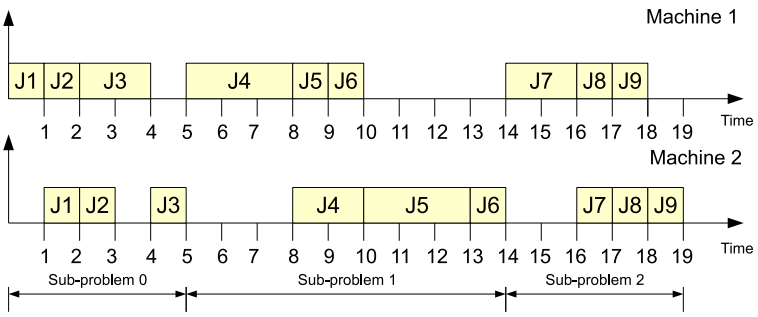

Figure 7.1: Polling system with three sub-problems (each corresponding to a queue visit) and three jobs per sub-problem.

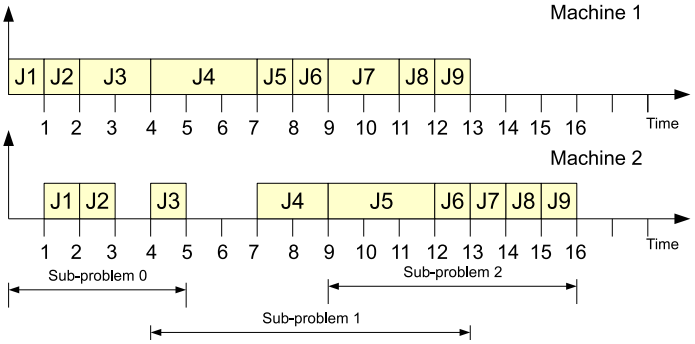

Figure 7.2: Dynamic flow shop with three sub-problems and three jobs per sub-problem. The start of a sub-problem is the start of a set of jobs on machine 1, the end of a sub-problem is the end of a set of jobs on machine 2.

performed once all jobs of the previous sub-problem have been processed on the *first* machine, as illustrated by Figure 7.2. This difference may seem minor, but as we show below it has a significant impact on the analysis of the two systems.

7.2.1 Methods for Solving Static Sub-problems

To our knowledge, policies for the polling system discussed in this chapter have not been examined in the queueing literature. We are also unaware of any queueing policy that has been proven to be optimal for the flow time objective, even in the expected sense, for a dynamic two-machine flow shop under our assumptions. Thus, we consider two queueing approaches for which theoretical results are available for systems related to ours. Specifically, the two queueing policies we use to solve each sub-problem are first-come, first-served (*FCFS*) and shortest total processing time first (SPT_{sum}).

Under *FCFS*, the jobs are processed in non-decreasing order of their arrival times to the queue. Towsley and Baccelli (1991) show that *FCFS* achieves the smallest expected flow time in a two-machine dynamic flow shop in the class of work-conserving, non-preemptive policies that *do not use* processing time information.

Employing SPT_{sum} means that all jobs present in the queue at the time of scheduling are processed in non-decreasing order of the *sum* of their durations on machine 1 and machine 2. This policy choice is motivated by the fact that, in the case when the server is a single unary resource, shortest processing time first minimizes the expected flow time in queueing systems with a single queue or with a polling structure under a cyclic, gated service discipline (Wierman et al., 2007). Wierman et al. show that such a policy can outperform *FCFS* by as much as 15% in a gated, cyclic polling system.

From the perspective of scheduling, each static queue sub-problem presents an opportunity for optimization. Since minimizing flow time under the above assumptions is equivalent to minimizing the total completion time, a natural choice of objective to be optimized in a sub-problem is the sum of completion times of activities on the second machine. We refer to this model as the *completionTime* model. Optimizing the total completion time will lead to the best *short-run* performance but, given the dynamics of the system, may not result in the best mean flow time in the *long run*. The *completionTime* model also has a computational disadvantage: minimizing the sum of completion times in a two-machine flow shop is NP-hard (Pinedo, 2003).

The fourth method we employ is motivated by a combination of queueing-based reasoning and the fact that scheduling methods can be used to optimize local queue performance. Specifically, suppose that we minimize the makespan for the set of jobs present in the queue, with makespan being defined as the difference between the end time of the job that is scheduled in the last position on machine 2 and the start time of the job that is scheduled in the first position on machine 1. Minimizing makespan may lead to a schedule with a sub-optimal mean flow time for the sub-problem, since minimizing mean flow time is not equivalent to minimizing makespan. However, the minimum makespan schedule may allow jobs in subsequent sub-problems to start earlier than under the *completionTime* approach. Earlier start times for all jobs would imply lower total completion times for all future sub-problems and, therefore, better *long-run* performance. Moreover, the optimal makespan schedule for a static two-machine flow shop can be found using a polynomial-time algorithm – Johnson's rule (Conway et al., 1967).

Johnson's rule divides jobs into two sets: set I consists of all jobs whose processing time on machine 1 is less than or equal to its processing time on machine 2, and set II consists of all the remaining jobs. Set I is processed before set II. Johnson's rule creates permutation

schedules, that is, schedules in which the order of jobs is the same on both machines. Within set I, jobs are sequenced in non-decreasing order of the processing time on machine 1, while within set II, jobs are sequenced in non-increasing order of the processing times on machine 2. The scheduling approach that uses Johnson's rule to solve each sub-problem will be referred to as the *makespan* approach.

Below we present experiments comparing the performance of *FCFS*, SPT_{sum}, *makespan* and *completionTime* models in our two problems of interest. The *completionTime* model was implemented via constraint programming in Ilog Scheduler 6.5 and uses the *completion* global constraint (Kovács and Beck, 2011). The remaining methods were implemented using C++. In these experiments, the *completionTime* model was run with a time limit of one second per sub-problem in order to ensure reasonable run-times for our experiments. With a time limit, this approach is not guaranteed to find the optimal sub-problem schedule. We do not take algorithm run-time into account in any of our results. Preliminary experiments with a time limit of five seconds showed identical performance.

7.2.2 Dynamic Flow Shop

To evaluate the performance of our four methods in a dynamic flow shop, we considered a system with exponentially distributed inter-arrival times and exponentially distributed processing times with the same means on both machines. We fixed the arrival rate, λ, to 10, and varied the load on the system by changing the rates of the processing time distributions from 100 to 10.53. The results of these experiments are shown in Figure 7.3. Each point in the figure represents the mean flow time over 100 instances of 55,000 jobs each. Preliminary experiments with uniformly distributed processing times showed similar results.

Figure 7.3 shows that there is no significant difference between the methods. The *completionTime* approach has a slight advantage over the others for loads of 0.7 and less, while SPT_{sum} is slightly better for loads of 0.8 and greater. The *makespan* model is marginally better than *FCFS* and *completionTime* at loads above 0.8.

SPT_{sum} is the best-performing model for loads above 0.8, when the static sub-problems become large. These observations are supported by the results of Xia et al. (2000), who have shown that SPT_{sum} is asymptotically optimal for the static average completion time objective as the number of jobs in a two-machine flow shop increases. It was not clear, a-priori, that applying this method to each sub-problem would result in the best long-run behaviour for high loads. In our study, we have not compared our periodic methods to the dispatching rules evaluated by Sarper and Henry (1996), who empirically show that ordering the jobs in non-decreasing order of processing times at *each machine* (i.e., *SPT* at each machine) results in

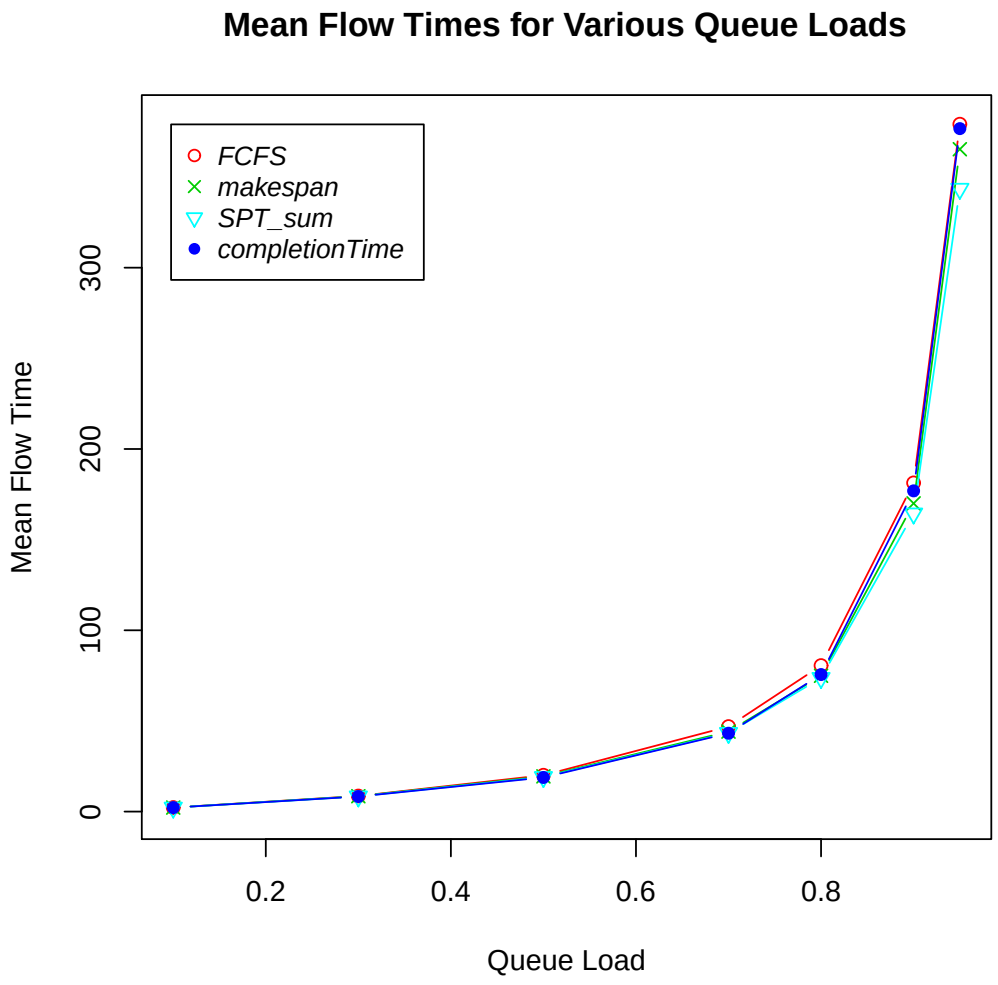

Figure 7.3: Mean flow times in a dynamic two-machine flow shop for *FCFS*, SPT_{sum}, *completionTime* and *makespan* models as the system load varies.

lowest mean flow times at loads of 0.8 and 0.9; comparison of our methods to such dispatching rules is left for future work. *FCFS* is the worst performer over all loads, due to the fact that it is the only method that does not use processing time information.

7.2.3 Polling System

In order to understand the performance of the four methods in a polling system with a flow-shop server, we simulated five symmetrical systems with $B = 5$ queues and with a fixed arrival rate of $\lambda_b = 1$ for all b. In each system, the processing times are exponentially distributed with the same means on both machines for all queues (i.e., $\mu_{1b} = \mu_{2b} = \mu$ for all b). As the mean processing times increase in the different experimental conditions, the load on the system,

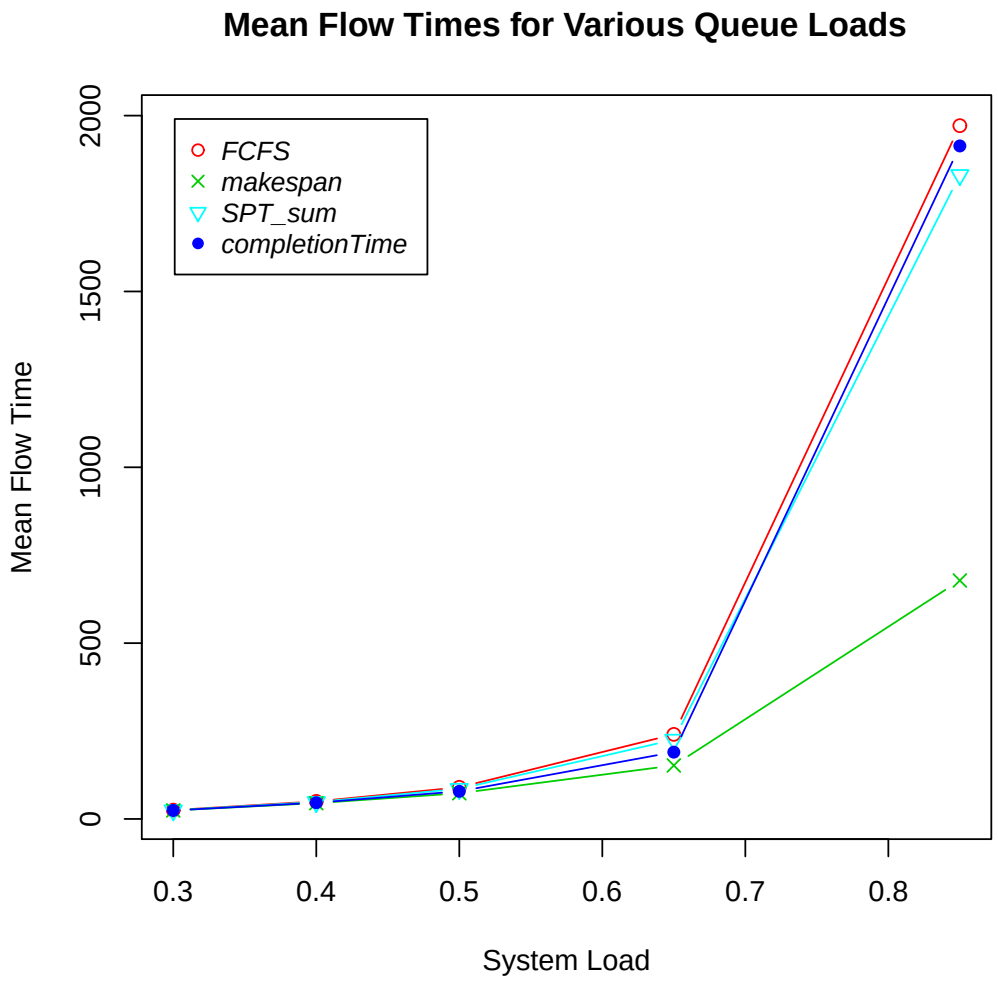

Figure 7.4: Mean flow times in a polling system with a two-machine flow shop server for *FCFS*, SPT_{sum}, *completionTime* and *makespan* models as the system load varies.

defined as $\sum_{b=1}^{B} \max\{\rho_{1b}, \rho_{2b}\} = 5\rho_{1b}$, increases. Thus, in order to observe the variation in performance as the load changes, we considered systems with $\mu \in \{16, 12, 10, 8, 6\}$.

Figure 7.4 shows the mean flow times for the *completionTime* model with a one-second time limit, *FCFS*, SPT_{sum} and the *makespan* model as the system load increases. Every point in this figure represents the mean flow time over 100 problem instances, each consisting of 25,000 jobs (5000 per queue). The figure shows that, for loads of 0.5 or less, the performance of the four methods is almost identical, although *makespan* has a slight advantage over the other three approaches. For loads greater than 0.5, *makespan* also achieves the lowest mean flow times. Moreover, the difference in performance between *makespan* and the other methods grows as the load increases. *FCFS* results in the highest flow times, and SPT_{sum} performs

better than *completionTime* when the load is 0.85. For an asymmetrical system consisting of five queues with different loads, the results match the pattern of Figure 7.4.

7.3 The *makespan* method vs. the *completionTime* method

In our analysis, we focus on understanding the performance differences between the *makespan* and *completionTime* models only. Since both approaches are based on periodic scheduling, we evaluate the overall impact of the differences between the two in each sub-problem solution.

7.3.1 Assumptions

To simplify our analysis, unless stated otherwise, we assume that the arrival processes are such that exactly n jobs are available at the time when a sub-schedule needs to be constructed. In the polling system, this assumption guarantees that both models solve exactly the same sub-problems (which is always true in the dynamic flow shop). Our results also hold if we assume that *at least* n jobs are available at the start of each sub-problem, the queue service discipline is limited-n rather than gated and the same set of jobs is selected by both *makespan* and *completionTime* at each polling instant.[7] In both the dynamic flow shop and the polling system, these assumptions imply that, as soon as a sub-problem ends, the next one starts immediately. In other words, in the dynamic flow shop, machine 1 never idles, and in the polling system, at least one of the two machines is always busy. We discuss these assumptions and their implications in Section 7.4.3. We assume that all processing times are finite, but do not make any other assumptions about their distributions.

7.3.2 Notation

Let the processing time of job j on machine m be denoted by p_{mj}. Let $\hat{C}_j$ be the completion time of job j in the schedule constructed by the *completionTime* model for the particular sub-problem to which j belongs, assuming that the *completionTime* model solves this sub-problem to *optimality*. Similarly, let C_j^M be the completion time of job j in the *minimum* makespan schedule provided by the *makespan* model for the same sub-problem. We denote the completion time of job j in an arbitrary schedule by C_j.

Given that there are n jobs per sub-problem, we assume that the jobs of sub-problem i, $i \geq 0$, are indexed from $in+1$ to $(i+1)n$.[8] Thus, for sub-problem i, $i \geq 0$, the sum of completion

[7] Under a limited-n discipline, at most n jobs can be processed during a queue visit (sub-problem). Refer to Section 6.1.2.2.2 for a review of the literature related to different queue service disciplines in polling systems.

[8] We relax this assumption in Section 7.4.3.

times for the *completionTime* model (i.e., the optimal sum of completion times) is $\sum_{j=in+1}^{(i+1)n} \hat{C}_j$, and the sum of completion times for the *makespan* model is $\sum_{j=in+1}^{(i+1)n} C_j^M$. Similarly, the makespan obtained from solving the ith problem using the *completionTime* model is denoted $\hat{C}_{max_i}$, while the makespan from the *makespan* model (i.e., the optimal makespan) is denoted $C_{max_i}^M$. The makespan of an arbitrary schedule is denoted by C_{max_i}.

Let $\Delta_i(\sum C_j) = \sum_{j=in+1}^{(i+1)n} C_j^M - \sum_{j=in+1}^{(i+1)n} \hat{C}_j$ be the difference in the total completion time between the *completionTime* and the *makespan* models for the ith sub-problem. Let $\Delta_i(C_{max}) = C_{max_i}^M - \hat{C}_{max_i}$ be the difference in the makespan values, for the ith sub-problem, between the two models. Consider a problem with T sub-problems, which are numbered $0, 1, \ldots, T-1$. The difference in the sum of completion times for T sub-problems is

$$\sum_{i=0}^{T-1} \Delta_i(\sum C_j) = \sum_{i=0}^{T-1} \sum_{j=in+1}^{(i+1)n} C_j^M - \sum_{i=0}^{T-1} \sum_{j=in+1}^{(i+1)n} \hat{C}_j. \tag{7.1}$$

Additionally, denote by C_j^0 the end time of job j under the assumption that the first job of the sub-problem to which j belongs starts at time 0, which can be viewed as a temporal mapping of the completion time. More formally, if s_i denotes the start of the sub-problem i to which job j belongs, then $C_j = s_i + C_j^0$. For example, in the polling system of Figure 7.1, $C_7 = 17$, $s_2 = 14$ and $C_7^0 = 3$. In the dynamic flow shop of Figure 7.2, $C_7 = 14$, $s_2 = 9$ and $C_7^0 = 5$. We refer to the values of C_j^0 as the *original* completion times and the values of C_j as the *actual* completion times. Thus, $\hat{C}_j^0$ and $C_j^{M,0}$ denote the original completion times of job j if the *completionTime* and the *makespan* policy are employed, respectively.

Since the *makespan* policy optimizes the makespan of each sub-problem and since the sub-problem's makespan is independent of the previous sub-problems, it follows that $\Delta_i(C_{max}) \leq 0$. Similarly, the *completionTime* model, theoretically, results in the optimal sum of completion times, and hence $\Delta_i(\sum C_j^0) \geq 0$. It is not obvious, however, that $\Delta_i(\sum C_j)$, which is based on the *actual* completion times, is greater than or equal to 0, since actual completion times are influenced by the previous sub-problems.

7.3.3 Dynamic Flow Shop

In the dynamic flow shop the sub-problems overlap: at a given point in time, machine 1 may be processing a job from sub-problem i while machine 2 is still processing jobs from sub-problem $i-1$. This overlap is taken into account when a static sub-problem is solved, and, therefore, is included in the calculation of C_j^0. This allows the completion time of job j belonging to sub-problem $i > 0$ to be written simply as $C_j = \sum_{l=1}^{in} p_{1l} + C_j^0$. That is, the actual completion time of job j is its original completion time *shifted* forward in time by the sum of the machine

1 processing times of all previous sub-problems. For example, in Figure 7.2 we see that the end time of every job belonging to sub-problem 1 is shifted by 4 time units, while the end times of jobs in sub-problem 2 are shifted by 9 units.

From the empirical results of Sections 7.2.2 and 7.2.3, it became clear that optimizing the total completion time of each sub-problem does not lead to the best long-run flow time performance in all environments due to the periodic nature of the methods. Moreover, Figure 7.3 does not provide a complete representation of how well the *completionTime* model performs compared to the other methods, since the sub-problem is not always solved to optimality within the given time limit. We establish the superiority of the theoretical *completionTime* model in the subsequent theorem.

Theorem 7.3.1. *In a dynamic two-machine flow shop under the assumptions of Section 7.3.1, if n is finite, then optimizing the total completion time of each sub-problem leads to a smaller sum of completion times over T time periods than minimizing each sub-problem's makespan. That is, $\sum_{i=0}^{T-1}\sum_{j=in+1}^{(i+1)n}\hat{C}_j \leq \sum_{i=0}^{T-1}\sum_{j=in+1}^{(i+1)n} C_j^M$, or, equivalently, $\sum_{i=0}^{T-1}\Delta_i(\sum C_j) \geq 0$.*

Proof. Equation (7.1) can be written in terms of C_j^0 and C_j^{M0} as follows:

$$\begin{aligned}
\sum_{i=0}^{T-1}\Delta_i(\sum C_j) &= \sum_{i=1}^{T-1}\Big[\sum_{j=in+1}^{(i+1)n}(C_j^M - \hat{C}_j)\Big] \\
&= \sum_{i=1}^{T-1}\Big[\sum_{j=in+1}^{(i+1)n}\big[(\sum_{l=1}^{in} p_{1l} + C_j^{M,0}) - (\sum_{l=1}^{in} p_{1l} + \hat{C}_j^0)\big]\Big] \\
&= \sum_{i=1}^{T-1}\Big[\sum_{j=in+1}^{(i+1)n}\big[C_j^{M,0} - \hat{C}_j^0\big]\Big] \\
&= \sum_{i=1}^{T-1}\Big[\Delta_i(\sum C_j^0)\Big], \qquad (7.2)
\end{aligned}$$

where the third equality follows from the fact that both the *completionTime* and the *makespan* approach solve exactly the same sub-problems, implying that the sum of machine 1 processing times is the same under both. The final expression shows that the total difference in the sum of *actual* completion times over T time periods is the same as the total difference in the sum of *original* completion times over T periods. We know that $\Delta_i(\sum C_j^0) \geq 0$ for all i by the fact that the theoretical *completionTime* model finds the optimal sum of completion times schedule. Therefore, $\sum_{i=0}^{T-1}\Delta_i(\sum C_j) \geq 0$, which means that optimizing the total completion time of each sub-problem will always lead to a better overall sum of completion times than minimizing each sub-problem's makespan. □

The results of Figure 7.3 are consistent with Theorem 7.3.1, since they show that the model that achieves the smallest mean flow times in the long run is the one that finds the best sum of completion times schedules for each sub-problem. For loads of 0.7 or less, this is the *completionTime* model. For all higher loads, the *completionTime* model run with a time limit is not always able to find the optimal solution. From our experimental results it is clear that SPT_{sum} finds better total completion time schedules when the value of n is not too small, leading to its overall best performance in terms of the long-run mean flow time.

A simple corollary of Theorem 7.3.1 concerns the limiting behaviour of the difference between the two models as T goes to ∞.

Corollary 7.3.1. $\lim_{T\to\infty} \sum_{i=0}^{T-1} \Delta_i(\sum C_j) \geq 0$.

Proof. Since $\Delta_i(\sum C_j^0) \geq 0$, it follows that

$$\lim_{T\to\infty} \sum_{i=0}^{T-1} \Delta_i(\sum C_j) = \lim_{T\to\infty} \sum_{i=1}^{T-1} \Delta_i(\sum C_j^0) \geq 0.$$

When $\Delta_i(\sum C_j^0) = 0$ for all i, $\lim_{T\to\infty} \sum_{i=0}^{T-1} \Delta_i(\sum C_j) = 0$. Whether the series converges or diverges depends on the properties of $\Delta_i(\sum C_j^0)$ and can be determined via standard convergence methods (see, e.g., the textbook of Stewart (1999)). □

Whether the series is convergent or divergent depends on whether the $\Delta_i(\sum C_j^0)$ are bounded for all i, which in turn depends on the distributions of the processing times. We leave further investigation of this question for future work.

The main message of the above analysis is that, under the assumptions of Section 7.3.1, but regardless of the processing time distributions and regardless of T and n, the *completionTime* model will perform better than *makespan* for the total completion time objective for the dynamic two-machine flow shop. We conjecture that the same is true for the M-machine dynamic flow shop. In such an environment, we can still write the completion time as $C_j = \sum_{l=1}^{in} p_{1l} + C_j^0$ where C_j^0 is the completion time of job j on the last machine given that the sub-problem starts at time 0 and overlapping jobs of sub-problem $i-1$ are taken into account. Thus, we can describe the M-machine dynamic flow shop with $M > 2$ using the differences in the sum of completion times and makespans of static sub-problems, similarly to the two-machine case.

7.3.4 Polling System

We view the global polling system schedule as a sequence of n-job sub-schedules, which allows us to disregard the information about the queue to which a particular sub-problem belongs and

makes our analysis simpler and independent of the polling order.

Given that the server switches between queues (and sub-problems) at the completion time of the last job on the second machine, every C_j in the dynamic problem can be represented in terms of C_j^0 and the sum of the makespans of the preceding sub-problems. That is, if job j belongs to sub-problem i, $i \geq 1$, then, under the assumptions of 7.3.1, $C_j = \sum_{k=0}^{i-1} C_{max_k} + C_j^0$: the actual completion time of job j is its original completion time *shifted* forward in time by the sum of the makespans of all previous sub-problems. For example, in Figure 7.1 we see that the end time of every job belonging to sub-problem 1 is shifted by 5 time units, while the end times of jobs in sub-problem 2 are shifted by 14 units. Equation (7.1) can be written in terms of C_j^0 and C_j^{M0} as follows:

$$\begin{aligned}
\sum_{i=0}^{T-1} \Delta_i(\sum C_j) &= n \sum_{i=1}^{T-1} \sum_{k=0}^{i-1} C_{max_k}^M + \sum_{i=0}^{T-1} \sum_{j=in+1}^{(i+1)n} C_j^{M0} \\
&- n \sum_{i=1}^{T-1} \sum_{k=0}^{i-1} \hat{C}_{max_k} - \sum_{i=0}^{T-1} \sum_{j=in+1}^{(i+1)n} \hat{C}_j^0 \qquad (7.3) \\
&= n[(T-1)\Delta_0(C_{max}) + (T-2)\Delta_1(C_{max}) + \dots \\
&+ \Delta_{(T-2)}(C_{max})] + \Delta_0(\sum C_j^0) + \cdots + \Delta_{T-1}(\sum C_j^0). \qquad (7.4)
\end{aligned}$$

Thus, we can calculate the difference in the sum of completion times for T periods by evaluating T static sub-problems under the assumption that each sub-schedule is started at time 0. When $\sum_{i=0}^{T-1} \Delta_i(\sum C_j) \leq 0$, we know that the *makespan* model is as good as or better than the *completionTime* model for the objective of minimizing the total completion time over T time periods. From Equation (7.4) we see that this condition depends on the values of n and T, and the magnitude of the differences in the sums of completion times and in the makespans of the sub-problems. Specifically, as n and T grow, $n[(T-1)\Delta_0(C_{max}) + \cdots + \Delta_{(T-2)}(C_{max})]$ decreases since $\Delta_i(C_{max}) \leq 0 \ \forall i$. However, an increase in T also leads to a greater number of $\Delta_i(\sum C_j^0)$'s in Equation (7.4), which in general could be much larger than the values of $|\Delta_i(C_{max})|$.

Our empirical results suggest that, in the polling system, minimizing the makespan of each sub-problem should always lead to better performance than minimizing the sum of completion times of the sub-problem (i.e., a result similar to the one of Theorem 7.3.1, with the roles of *completionTime* and *makespan* reversed). However, this turns out to not be the case in general. Specifically, for a finite T and n, it is possible to construct an example where minimizing each sub-problem's makespan results in a *worse* overall sum of completion times than minimizing each sub-problem's sum of completion times. One such example occurs when the *makespan*

and *completionTime* models produce schedules with equal makespans and equal sums of completion times for all but the last sub-problem of the scheduling horizon, as formally stated in Proposition 7.3.1.

Proposition 7.3.1. *If* $\Delta_i(C_{max}) = 0$ *and* $\Delta_i(\sum C_j^0) = 0$ *for* $i = 0, 1, \ldots, T-2$*, but* $\Delta_{T-1}(C_{max}) < 0$ *and* $\Delta_{T-1}(\sum C_j^0) > 0$*, then* $\sum_{i=0}^{T-1} \Delta_i(\sum C_j) > 0$*, implying that the* completionTime *model results in a smaller overall sum of completion times for any finite* T *and any finite number of jobs per sub-problem,* n*.*

Proof. The result follows simply from Equation (7.4):

$$\begin{aligned}
\sum_{i=0}^{T-1} \Delta_i(\sum C_j) &= n[(T-1)\Delta_0(C_{max}) + (T-2)\Delta_1(C_{max}) + \ldots \\
&+ \Delta_{(T-2)}(C_{max})] + \Delta_0(\sum C_j^0) + \cdots + \Delta_{T-1}(\sum C_j^0) \\
&= 0 + \Delta_{T-1}(\sum C_j^0) \\
&= \Delta_{T-1}(\sum C_j^0) > 0. \qquad (7.5)
\end{aligned}$$

□

The next question of interest is the limiting behaviour of the difference between the two models. Firstly, we present in Proposition 7.3.2 one set of conditions under which the difference between *makespan* and *completionTime* grows in favour of *makespan* as $T \to \infty$. Specifically, this case occurs if the schedules provided by the two models differ in their sum of completion times and makespans for sub-problem 0 only. This result occurs due to the fact that a shorter makespan in the initial sub-problem allows all subsequent jobs to start earlier under the *makespan* approach than under *completionTime*; the effect of the positive difference in the total completion time of sub-problem 0 does not propagate.

Proposition 7.3.2. *If* $\Delta_0(C_{max}) < 0$ *and* $\Delta_0(\sum C_j^0) > 0$*, but* $\Delta_i(C_{max}) = 0$ *and* $\Delta_i(\sum C_j^0) = 0$ *for all* $i \geq 1$*, then for any finite* n*,* $\lim_{T\to\infty} \sum_{i=0}^{T-1} \Delta_i(\sum C_j) = -\infty$*.*

Proof. The result follows simply from Equation (7.4):

$$\begin{aligned}
\lim_{T\to\infty} \sum_{i=0}^{T-1} \Delta_i(\sum C_j) &= \lim_{T\to\infty} [n[(T-1)\Delta_0(C_{max}) + (T-2)\Delta_1(C_{max}) + \ldots \\
&+ \Delta_{(T-2)}(C_{max})] + \Delta_0(\sum C_j^0) + \cdots + \Delta_{T-1}(\sum C_j^0)] \\
&= \lim_{T\to\infty} [n(T-1)\Delta_0(C_{max}) + \Delta_0(\sum C_j^0)] \\
&= -\infty, \qquad (7.6)
\end{aligned}$$

where the last equality follows from the fact that both $\Delta_0(C_{max})$ and $\Delta_0(\sum C_j^0)$ are finite and $\Delta_0(C_{max}) < 0$. □

More generally, if the difference in the sum of completion times between the two models is bounded and there is a non-zero probability that $\Delta_i(C_{max}) < 0$ for an infinite number of sub-problems, then, as the number of sub-problems, T, increases, the *makespan* model becomes better than the *completionTime* model with respect to the overall total sum of completion times.

Theorem 7.3.2. *If* $\Delta_i(\sum C_j^0) \leq -\phi\Delta_i(C_{max})$ *for all* i*, where* $1 \leq \phi < \infty$*, and* $P(\Delta_i(C_{max}) < 0) > 0$ *for an infinite number of sub-problems, then* $\lim_{T\to\infty} \sum_{i=0}^{T-1} \Delta_i(\sum C_j) = -\infty$.

Proof. Firstly, we derive a bound on $\sum_{i=0}^{T-1} \Delta_i(\sum C_j)$:

$$\begin{aligned}
\sum_{i=0}^{T-1} \Delta_i(\sum C_j) &= n[(T-1)\Delta_0(C_{max}) + (T-2)\Delta_1(C_{max}) + \dots \\
&+ \Delta_{(T-2)}(C_{max})] + \Delta_0(\sum C_j^0) + \dots + \Delta_{T-1}(\sum C_j^0) \\
&\leq n[(T-1)\Delta_0(C_{max}) + (T-2)\Delta_1(C_{max}) + \dots \\
&+ \Delta_{(T-2)}(C_{max})] - \phi\Delta_0(C_{max}) - \phi\Delta_1(C_{max}) - \dots - \phi\Delta_{T-1}(C_{max}) \\
&= (n(T-1) - \phi)\Delta_0(C_{max}) + (n(T-2) - \phi)\Delta_1(C_{max}) + \dots \\
&+ (n-\phi)\Delta_{(T-2)}(C_{max}) - \phi\Delta_{T-1}(C_{max}). \qquad (7.7)
\end{aligned}$$

Since ϕ and $\Delta_{T-1}(C_{max})$ are finite, and since our assumption that $P(\Delta_i(C_{max}) < 0) > 0$ for an infinite number of sub-problems implies that there will be an infinite number of non-zero $\Delta_i(C_{max})$ values, taking the limit as $T \to \infty$ of Expression (7.7) results in $-\infty$. Since Expression (7.7) is an upper bound on $\sum_{i=0}^{T-1} \Delta_i(\sum C_j)$, it follows that $\lim_{T\to\infty} \sum_{i=0}^{T-1} \Delta_i(\sum C_j) = -\infty$. □

The assumption that $\Delta_i(\sum C_j^0) \leq -\phi\Delta_i(C_{max})$ for all i, where $1 \leq \phi < \infty$, holds if the number of jobs in sub-problem i and all their processing times are finite, regardless of the distribution.[9] The proof of Theorem 7.3.2 suggests that the difference between the two models grows at a faster rate when n is larger (but still finite).

Therefore, the conclusions of our investigation are consistent with our motivations for considering methods based on optimizing makespan and total completion time and with the empirical results of Figure 7.4. Indeed, as the number of sub-problems increases, the *makespan*

[9]We assume that $\Delta_i(C_{max}) = 0$ implies that the schedules produced by *makespan* and *completionTime* are the same, also implying $\Delta_i(\sum C_j^0) = 0$. In practice, *makespan* and *completionTime* could produce schedules that would be the same with respect to one objective but not the other. It is easy to modify the algorithms to prevent such occurrences.

model, which considers long-run objectives, can achieve significant gains over the *completionTime* model, which optimizes short-run behaviour.

The above analysis shows that, under the assumptions of Section 7.3.1, in the dynamic flow shop, the *completionTime* model always performs better than the *makespan* model. While not always true for a finite number of sub-problems, the difference between the two models is reversed in favour of *makespan* as the number of sub-problems increases to infinity in the polling system.

Another important parameter for both the polling system and the dynamic flow shop is the number of jobs per sub-problem, n. It is therefore interesting to consider the value of $\lim_{n\to\infty} \sum_{i=0}^{T-1} \Delta_i(\sum C_j)$. In fact, the exact value of this limit depends on the distribution of the processing times, since the values of $\Delta_i(\sum C_j)$ are distribution-dependent. For the dynamic two-machine flow shop under our assumptions, however, it holds trivially that $\lim_{n\to\infty} \sum_{i=0}^{T-1} \Delta_i(\sum C_j) \geq \lim_{n\to\infty} \Delta_0(\sum C_j^0) \geq 0$ since $\Delta_0(\sum C_j^0) \geq 0$ for any n by definition. For the polling system, this argument does not hold since $\sum_{i=0}^{T-1} \Delta_i(\sum C_j)$ can be negative when $\Delta_0(\sum C_j^0) \geq 0$, and we leave the investigation of the properties of this limit for future work.

7.4 Discussion

In this section, we provide some discussion of our results and assumptions.

7.4.1 Polling System vs. Dynamic Flow Shop

Equations (7.2) and (7.4), together with our empirical results, explain the differences in the behaviour of the *makespan* and the *completionTime* models in the dynamic flow shop and in the polling system. They also lead to several insights regarding dynamic scheduling.

Firstly, in the case of the polling system, our analysis demonstrates a conflict between short-run and long-run objectives. That is, minimization of the sum of completion times at each scheduling point results in the best performance for the current sub-problem, but leads to poor performance in the long run. Minimization of the makespan, on the contrary, leads to sub-optimal sum of completion time values for each sub-problem, but results in significant overall mean flow time improvements. In dynamic two-machine flow shops there is no conflict between the way in which we have attempted to optimize long-run and short-run objectives: minimizing the total completion time of each sub-problem leads to better long-run mean flow time than does minimizing makespan. These observations demonstrate the importance of considering both short-run and long-run objectives when designing a scheduling algorithm for any

dynamic scheduling problem.

Secondly, we see that the difference between the performance of different periodic scheduling methods in the two systems is induced by the change in the time point at which a new sub-problem is solved. Recall that in the dynamic flow shop, scheduling occurs at the completion time of the previous sub-problem on *machine 1*, while in the polling system, it occurs at the completion time of the previous sub-problem on *machine 2*. This disparity in the review time point can be viewed as the result of either a structural difference in the systems or a change in the design of the periodic scheduler. Structurally, the two systems differ because the dynamic flow shop has one queue (serving one customer or producing one product type), while the polling system has multiple (serving multiple customers or producing multiple product types). Alternatively, we can think of the review time point as being a parameter in the design of the periodic scheduling approach. Thus, we could have, for example, solved the dynamic flow shop problem instances by reviewing the system status at the completion time of sub-problems on machine 2, as in the polling system. We therefore conclude that, when one attempts to create a scheduling approach for a dynamic system, one needs to keep in mind that changes in both the system structure and the parameters of the approach can have a significant effect on how the trade-off between short-run and long-run objectives affects the system's performance.

Thirdly, the choice of objective function for each sub-problem, which is a proxy measure for the long-run objective, may not always be obvious and is dependent on system structure. Thus, our work suggests that, for general dynamic scheduling problems, it is important to investigate various objective functions in order to develop an effective scheduling method. Our results also indicate that, in a polling system with a flow shop-like server, an approach that would find, for each sub-problem, the minimum makespan schedule with the smallest total completion time would outperform the methods we presented here. This hierarchical objective function would have the long-run focus of the *makespan* model, but would also be better than *makespan* in the short-run. In a dynamic flow shop, an approach that finds the optimal completion time schedule with the smallest makespan is of interest. However, our results imply that this approach would provide only marginal improvements over the *completionTime* model. It would also be interesting to investigate weighted combinations of objectives.[10]

7.4.2 Queueing vs. Scheduling Methods

Additionally, the polling system results suggest that periodic scheduling methods can perform better than queueing-based dispatching rules for optimization of long-run performance, since *makespan* significantly outperforms *FCFS* with respect to mean flow time. We discuss our

[10]Thank you to Dr. Michael Trick for this suggestion.

plans for further investigation of the relative performance of queueing and scheduling algorithms, and their hybrids, in Section 10.3.

7.4.3 Sub-problem Size Assumptions

One simplifying assumption that we made in the analysis of Section 7.3 is that every sub-problem consists of exactly n jobs. Suppose now that the number of jobs changes, and denote the number of jobs in sub-problem i by n_i. Then in the dynamic flow shop without any additional assumptions and in the polling system under the assumption that both the *makespan* and the *completionTime* models encounter exactly the same sub-problems, one needs to substitute n_i for each occurrence of n in the calculations of $\Delta_i(\sum C_j^0)$ and $\Delta_i(C_{max})$; the equivalents of all results of Section 7.3 would hold.

We also assume that a new set of n jobs is available *whenever* the previous sub-problem ends. This assumption prevents the occurrence of idle times between sub-problems. Specifically, in the polling system, it implies that there will be no idle times between the end of a sub-problem on machine 2 and the start of the next sub-problem on machine 1; in the dynamic flow shop, idle times between the end of a sub-problem on machine 1 and the start of the next sub-problem on machine 1 are prevented. In the general case, it is possible for such idle times to occur since the arrival process is stochastic, and Equations (7.2) and (7.4) would have to be modified to include the sum of all idle times. For arrival processes such as the Poisson process, the probability of getting no arrivals during a sub-problem would decrease as its length increases. We conjecture that the majority of results based on Equations (7.2) and (7.4) would not change due to the inclusion of idle times, though they might require the assumption of an upper bound on the total idle time.

Consider now our assumption that the two models solve exactly the same sub-problems. In the dynamic flow shop, this is in fact always true, since a new schedule is created when the last job in the previous sub-problem is completed on machine 1, which occurs at the same time in both the *completionTime* and the *makespan* models. In the polling system, the scheduling time point directly depends on the makespans of all previous sub-problems. Thus, a difference between the makespans resulting from the *makespan* and the *completionTime* models may imply that the *makespan* model will solve more sub-problems with a smaller number of jobs. The original situation in which the two approaches have to solve different sub-problems can happen only if the arrival rate and $|\Delta_i(C_{max})|$ values are sufficiently high: there has to exist some sub-problem i such that there is at least one job in the queue at the completion of this sub-problem under both models and there is an additional arrival in the time interval between $\sum_{k=0}^{i} C_{max_k}^M$ and $\sum_{k=0}^{i} \hat{C}_{max_k}$. After this case occurs, it is likely that the sub-problems would

continue being different when the arrival rates are high. Equation (7.4) would not be directly applicable if the two models encountered different problems, but it would be an approximation of the actual difference in the sum of completion times. The insight gained from this equation, namely that the makespans of the sub-problems have a direct effect on the sum of completion times, would still hold.

Therefore, our assumptions regarding the fixed sub-problem size for both models greatly simplify the analysis while still providing us with a good understanding of how the *makespan* and *completionTime* methods work for our systems of interest.

7.5 Conclusion

We investigated two dynamic environments: a two-machine flow shop, which has received significant attention in scheduling research, and a polling system with a flow shop server, an extension of systems typically studied in queueing theory. Based on analysis and experiments, and using ideas from both queueing theory and scheduling, we obtained new insights regarding the solution of dynamic scheduling problems by a periodic scheduling approach. Firstly, we demonstrated that it is important to evaluate the trade-off between short-run and long-run objectives in dynamic scheduling. In the polling system, minimizing makespan at each scheduling point leads to better mean flow time over a long horizon than does minimizing flow time itself or using simple queueing policies. The opposite is true in a dynamic flow shop without a polling structure. Secondly, our polling system results suggest that periodic scheduling methods can perform better than queueing-based dispatching rules for optimization of long-run performance. However, the choice of objective function for each sub-problem may not always be obvious. Thus, this chapter demonstrates that combining ideas, concepts and problem settings from queueing theory and scheduling can lead to a better understanding of dynamic scheduling.

In the next chapter, we consider the theoretical integration of queueing and scheduling. Specifically, we focus on the queueing notion of stability, and prove stability of *FCFS* and *makespan* in both the dynamic flow shop and the polling system with a flow shop server.

Chapter 8

Theoretical Integration of Scheduling and Queueing Theory: Stability

Stability analysis consists of identifying conditions under which the number of jobs in a system is guaranteed to remain bounded over time. In this chapter,[1] we analyze stability of the two-machine flow shop and the polling system with a flow shop server that have been presented in the previous chapter. In the dynamic flow shop, stability of a scheduling approach that periodically solves static deterministic sub-problems is shown using a sample path argument. In the polling system, stability of *FCFS* and of a periodic scheduling method is proven using the fluid model methodology of Dai (1995).

This chapter demonstrates that theoretical integration of queueing and scheduling can lead to long-run performance guarantees for scheduling algorithms that have previously been available only for queueing policies. In particular, in two environments, we prove stability of a scheduling method that is based on the traditional scheduling literature and utilizes processing time information to make sequencing decisions.

In more detail, the contributions of this chapter are:

1. We introduce long-run stability to combinatorial scheduling and prove that a periodic scheduling approach based on makespan minimization is stable in the dynamic flow shop environment.

2. We prove stability of *FCFS* in a polling system with a gated, cyclic discipline and a server that is a two-machine flow shop. We show that this proof extends to the case when the server is an M-machine flow shop or a d-stage flexible flow shop with M machines

[1] Parts of this chapter appear as a technical report (Terekhov et al., 2012b) and as a conference paper (Terekhov et al., 2012c). The material included in the conference paper (Terekhov et al., 2012c) is used with permission of the copyright holder, the Association for the Advancement of Artificial Intelligence ©, and the paper's co-authors.

at each stage. Moreover, our proofs imply that all non-idling policies that do not utilize processing time information are stable under the stability condition of *FCFS*.[2]

3. We prove stability of a periodic scheduling method that minimizes makespan of each static sub-problem for a polling system with a two-machine flow shop server. This proof extends to the case when the server is an M-machine flow shop or a d-stage flexible flow shop with M machines at each stage.

4. We demonstrate the applicability of the fluid model methodology for proving stability of scheduling policies based on processing times.

This chapter is organized as follows. Firstly, we provide a brief introduction to stability. Section 8.2 discusses the stability of *FCFS* and *makespan* in the dynamic flow shop. The proof of stability of the *makespan* approach is shown using a sample path argument. Section 8.3 proves stability of *FCFS* and *makespan* in the polling system using the fluid model methodology of Dai (1995). We conclude the chapter in Section 8.5.

8.1 Introduction to Stability

Informally, a system is *stable* if its queues remain bounded over time.[3] In queueing theory, establishing the stability of a system is considered a precursor to more detailed analysis (Kumar and Meyn, 1995). In early queueing work it was believed that to ensure a stable system, it was necessary and sufficient to have the load of each machine, defined as the ratio of the arrival rate to the processing rate, be strictly less than 1 (Dai and Weiss, 1996).

However, a series of papers (Lu and Kumar, 1991; Bramson, 1994; Seidman, 1994) provided counter-examples which demonstrated that this load condition is not always sufficient (Dai and Weiss, 1996). In particular, the stability of a system is dependent on the scheduling policy it employs: for a given set of problem parameters (arrival and processing rates), one policy may stabilize the system while another might not. Knowledge of whether a system is stable for a given job arrival rate, processing rate and scheduling policy is essential for practical applications. For instance, processes in semi-conductor manufacturing are frequently modelled as reentrant lines (Kumar, 1994), and Dai and Weiss (1996, p. 27) state that "stability is a

[2]Thank you to Dr. Timothy C. Y. Chan for this observation.

[3]In the scheduling literature, a predictive schedule is called *stable* if the schedule executed online is close to the planned schedule (Bidot et al., 2009). Similarly, in scheduling under uncertainty, stability analysis concerns the identification of the range of values that the processing times may take while the given schedule remains optimal (Sotskov et al., 2010). We do not use these definitions here, instead adopting the meaning of stability from queueing theory.

first issue one needs to address if one wishes to study optimal or near-optimal scheduling of a reentrant line."

Formally, a system operating under a particular queueing discipline is stable if the Markov process that describes the dynamics of the system is *positive Harris recurrent* (Dai, 1995). Positive Harris recurrence implies the existence of a unique stationary distribution. Due to the considerable notation required, we do not formally define positive Harris recurrence here, but instead refer the reader to the papers by Dai (1995), Dai and Meyn (1995) and Bramson (2008). In the case when the state space of the Markov process is countable[4] and all states communicate, positive Harris recurrence is equivalent to the more widely-known concept of positive recurrence (Bramson, 2008). A Markov chain is positive recurrent if every state s is positive recurrent: the probability that the process starting in state s will return to s is 1, and the expected time to return to this state is finite (Ross, 2003). In particular, this property guarantees that the system will empty.

Except for the stability of *FCFS* in a two-machine flow shop, the stability problems we address are different from those in the literature for two reasons. Firstly, our polling system with a two-machine flow shop server can be viewed as a hybrid of two well-studied systems, a tandem queue and a polling system with a single-machine server. Secondly, for both the dynamic flow shop and the polling system, we consider scheduling policies which assume knowledge of the processing times and are based on the notion of periodic scheduling and makespan minimization. As far as we are aware, neither the stability of the hybrid system nor the stability of the periodic scheduling method with makespan minimization have been addressed in the queueing literature. We are unaware of any work on the stability analysis of scheduling policies based on exact knowledge of processing times.

In this chapter, we assume the problem definitions presented in Section 7.1. However, we make some additional assumptions on the inter-arrival and processing time distributions that are standard in the queueing literature dealing with stability analysis (Dai, 1995):

- For each queue, the sequences of processing times for machine 1 and machine 2, and the sequence of inter-arrival times, are independent and identically distributed sequences. They are also mutually independent.
- Mean processing times and mean inter-arrival times for all queues are finite.
- The inter-arrival times are unbounded and continuous.

These assumptions are satisfied by commonly used distributions such as the exponential distribution. In order to avoid introducing too much notation, we do not state these assumptions

[4]This process is referred to as a *continuous-parameter Markov chain* in the book by Gross and Harris (1998).

formally here. Instead, they are formally defined, using the notation relevant to the polling system proofs, in Section 8.3.1.

8.2 Stability of the Dynamic Flow Shop

We firstly prove stability of the dynamic flow shop with two machines. Subsequently, we briefly investigate the case with M machines.

8.2.1 Two-Machine Case

In the dynamic flow shop with two machines, the following theorem holds:

Theorem 8.2.1. *If* $\frac{\lambda}{\min\{\mu_1,\mu_2\}} < 1$*, then the tandem queue with the periodic* FCFS *policy is stable.*

This result follows trivially from the fact that for the dynamic flow shop, the periodic *FCFS* policy is equivalent to the "standard", non-periodic implementation of *FCFS*, and the fact that, under our assumptions, the tandem queue is a generalized Jackson network. Stability of such networks under the condition that the load of each machine is strictly less than 1 has been proven previously by various methods, such as the fluid model methodology of Dai (1995). The same methodology can be used to show the stability of *FCFS* in an M-machine flow shop under the condition that $\frac{\lambda}{\min_{m\in\{1,\ldots,M\}}\{\mu_m\}} < 1$.

For the *makespan* policy, we prove a result which holds for every sample path in the evolution of the system. Let s_i^* and t_i^* be the time points at which sub-problem i starts on machine 1 and completes on machine 2, respectively, under the *makespan* approach. Let v_i and v_i^π be the total processing time of all jobs completed by time t_i^* under the *makespan* policy and an arbitrary policy π, respectively. Following the queueing literature, we further refer to v_i and v_i^π as the *work* completed by time t_i^*.

Lemma 8.2.1. *The amount of work completed by* t_i^* *is maximized by the* makespan *policy. That is,* $v_i \geq v_i^\pi$ *for all* i *and all non-idling* π*.*

Proof. For any arbitrary non-idling policy π, v_i^π can be written as $v_i^{1,\pi} + v_i^{2,\pi}$, where $v_i^{1,\pi}$ is the amount of work completed by t_i^* on machine 1 and $v_i^{2,\pi}$ is the amount of work completed by t_i^* on machine 2. (In Figure 8.1, t_0^* is 5, $v_0^{1,\pi} = 5$ and $v_0^{2,\pi} = 3$.) For the *makespan* approach, we use the notation without the superscript π, i.e., $v_i = v_i^1 + v_i^2$. In the dynamic flow shop, the amount of work completed by t_i^* on machine 1 is the same for any non-idling policy. Thus, it remains to prove that $v_i^2 \geq v_i^{2,\pi}$. We prove this by induction.

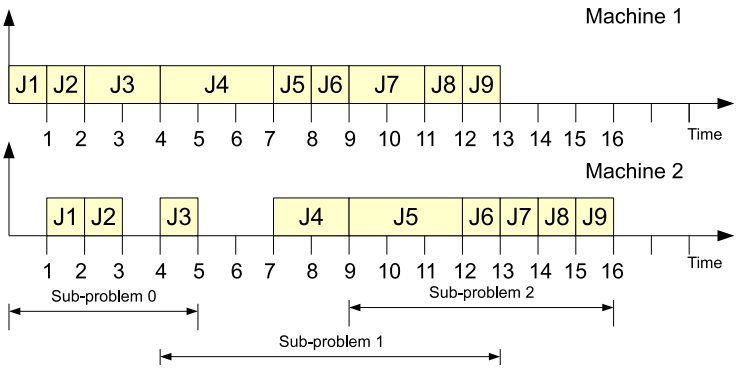

Figure 8.1: Schedule for policy π for a two-machine flow shop. This figure also appears in Chapter 7 as Figure 7.2.

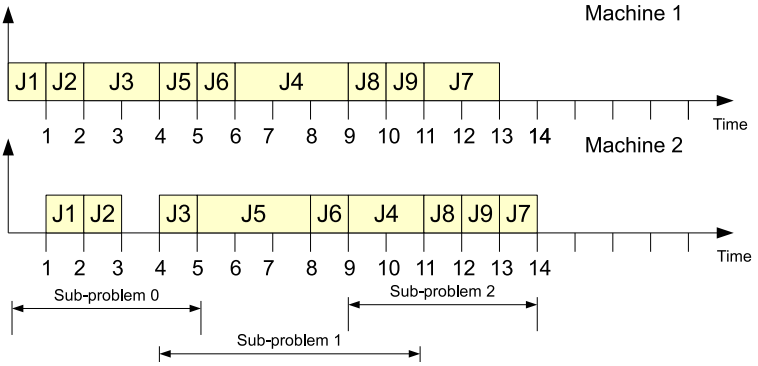

Figure 8.2: Schedule for *makespan* for the same problem instance as in Figure 8.1.

Base Case: $v_0^2 \geq v_0^{2,\pi}$ since any policy other than *makespan* can complete the set of initial jobs only at the same time as *makespan* (t_0^*) or later.
Inductive Hypothesis: Assume the property is true for t_i^*, that is, $v_i^2 \geq v_i^{2,\pi}$.
Inductive Step: We need to show the same property for t_{i+1}^*, i.e., that $v_{i+1}^2 \geq v_{i+1}^{2,\pi}$. For the *makespan* approach, $v_{i+1}^2 = v_i^2 + \gamma((s_i^*, s_{i+1}^*])$, where $\gamma((s_i^*, s_{i+1}^*])$ is the total machine 2 workload that arrives in the time period $(s_i^*, s_{i+1}^*]$ and, therefore, is the workload that is processed in the sub-problem starting at s_{i+1}^* and ending at t_{i+1}^*.

By the induction hypothesis, we know that at t_i^*, $v_i^2 \geq v_i^{2,\pi}$. The amount of work processed on machine 2 by time t_{i+1}^* by policy π, $v_{i+1}^{2,\pi}$, equals the amount of work processed by π by time t_i^* plus some fraction of the difference in the amount of work completed by π and *makespan* by t_i^* plus some fraction of the amount of machine 2 work that arrives in $(s_i^*, s_{i+1}^*]$. Thus, $v_{i+1}^{2,\pi} \leq v_i^{2,\pi} + (v_i^2 - v_i^{2,\pi}) + \gamma((s_i^*, s_{i+1}^*]) = v_i^2 + \gamma((s_i^*, s_{i+1}^*]) = v_{i+1}^2$. □

For example, consider the schedules in Figures 8.1 and 8.2: $s_0^* = 0$, $s_1^* = 4$, $s_2^* = 9$, $t_0^* = 5$,

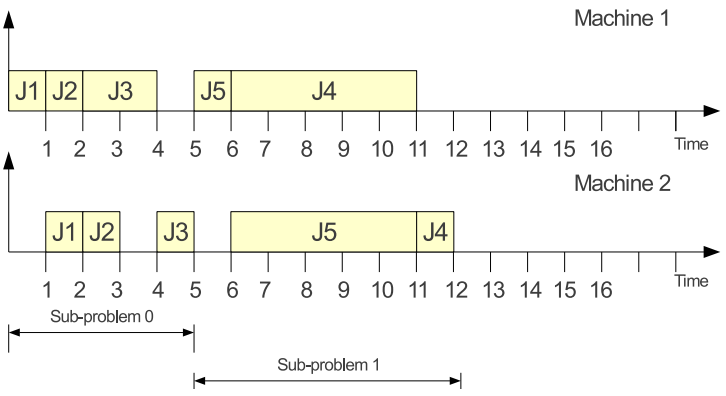

Figure 8.3: Schedule for an idling policy π for a two-machine flow shop.

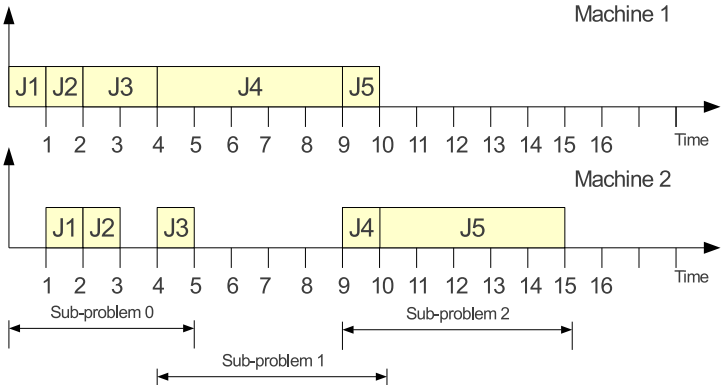

Figure 8.4: Schedule for *makespan* for the same problem instance as in Figure 8.3.

$t_1^* = 11$, $t_2^* = 14$, $v_2^2 = v_1^2 + \gamma((s_1^*, s_1^*]) = 9 + 3 = 12$, $v_2^{2,\pi} = 7 + (9 - 7) + 1 \leq 12$.[5]

The lemma does not hold if π is idling since an idling policy may create a better schedule by waiting and taking more jobs into account. Consider the system with sub-problem 0 as in the above examples. Suppose at time 4, there is an arrival of job $J4$ with $p_{14} = 5$ and $p_{24} = 1$; at time 5, there is an arrival of job $J5$ with $p_{14} = 1$ and $p_{24} = 5$. Figure 8.3 shows the schedule corresponding to a policy π that idles machine 1 for one time unit from 4 to 5 even though a job is available for processing. Figure 8.4 displays the schedule corresponding the non-idling *makespan* approach. We see that at $t_1^* = 10$, $v_1^\pi = 7 + 9 \leq 7 + 7 = v_1$. One explanation for this counter-example is presented in Section 9.2.1.

Theorem 8.2.2. *If* $\frac{\lambda}{\min\{\mu_1,\mu_2\}} < 1$ *then the tandem queue with the periodic* makespan *policy is stable.*

Proof. We know that *FCFS* is stable under the given condition. By Lemma 8.2.1, at every

[5]We assume that $J7$, $J8$ and $J9$ arrive in time period $(4, 9]$.

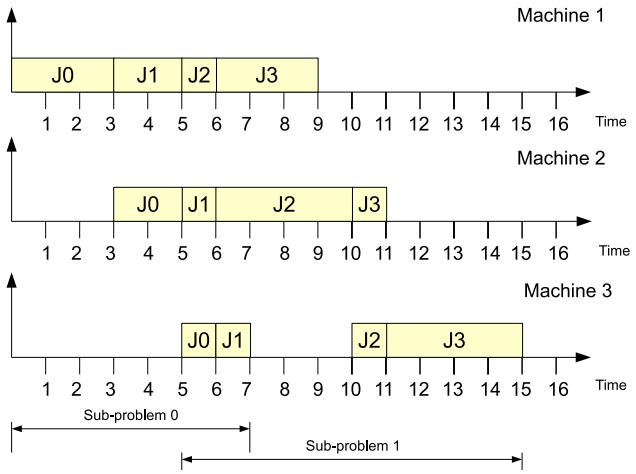

Figure 8.5: Schedule for *FCFS* for the Dynamic Flow Shop with Three Machines.

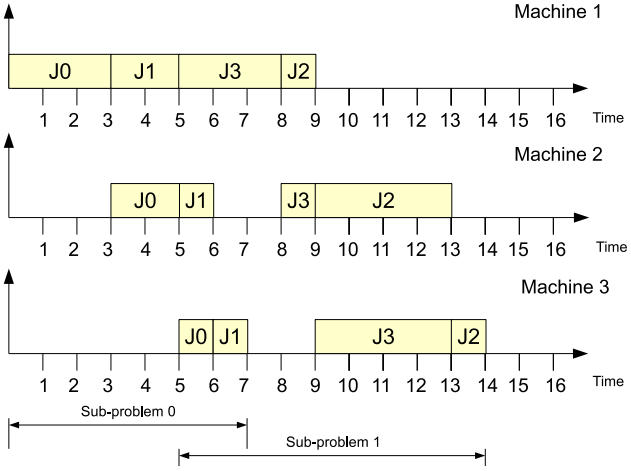

Figure 8.6: Schedule for *makespan* for the Dynamic Flow Shop with Three Machines.

sub-problem completion time, the *makespan* approach has finished at least as much work as *FCFS*. Therefore, the tandem queue with the periodic *makespan* policy is stable under the same condition as *FCFS*. □

The theorem provides a sufficient condition for stability of the tandem queue under the periodic *makespan* policy. From the literature, we know that $\frac{\lambda}{\mu_1} < 1$ and $\frac{\lambda}{\mu_2} < 1$ is a necessary condition for stability of this system. Since ensuring that $\frac{\lambda}{\min\{\mu_1,\mu_2\}} < 1$ is the same as ensuring $\frac{\lambda}{\mu_1} < 1$ and $\frac{\lambda}{\mu_2} < 1$, we see that $\frac{\lambda}{\min\{\mu_1,\mu_2\}} < 1$ is actually a necessary and sufficient condition for stability of the tandem queue under the *makespan* policy.

8.2.2 Extension

Lemma 8.2.1 does not hold for an M-machine flow shop when $M > 2$. To illustrate this fact, consider the problem instance in Figures 8.5 and 8.6. In this example, sub-problem 0 consists of jobs $J0$ and $J1$, and both the *makespan* policy and *FCFS* construct the same schedule, with $t_0^* = 7$. The second sub-problem consists of $J2$ and $J3$, and the two policies result in different schedules. At $t_0^* = 7$, the amount of work completed by *makespan* is $v_0 = 12$, whereas the amount of work completed by *FCFS* is $v_0^\pi = 13$. Therefore, in this case, $v_0 < v_0^\pi$, which shows that it is possible that the amount of work completed by t_i^* is *not* maximized by the *makespan* policy. We nonetheless conjecture that Theorem 8.2.2 extends to the case with more than two machines and can be proven using a fluid model approach. We leave this investigation for future work.

8.3 Stability of the Polling System

As mentioned previously, the stability of polling systems with a two-machine flow shop server and processing times that become known upon arrival of jobs to the system has not been addressed in the literature. For a summary of results on the stability of polling systems with a single-machine server, the reader is referred to pages 10 and 11 of the review by Takagi (1988). In particular, a cyclic gated polling system with B queues and *FCFS* policy employed within each queue is stable if and only if $\rho = \sum_{k=1}^{B} \frac{\lambda_k}{\mu_k} < 1$.

The sample path result of Lemma 8.2.1 does not hold in our polling system. This can be illustrated by Figures 8.7 and 8.8, which show schedules for the *completionTime* and the *makespan* approaches for an example instance with jobs $J0$, $J1$, $J2$ available at time 0, $J3$ arriving at time 13 and $J4$ arriving at time 14. Under the *makespan* approach, sub-problem 0 completes at time 13 and so sub-problem 1 consists of $J3$ only; when the *completionTime* method is used, sub-problem 0 completes at time 14, so sub-problem 1 consists of both jobs $J3$ and $J4$. Thus, by time $t_1^* = 18$, the total amount of work completed is 23 for the *makespan* model and 25 for the *completionTime* approach. However, stability of both *FCFS* and *makespan* is proven using the fluid model methodology in the subsequent sections. In order to apply this methodology, we firstly present a formal model of the polling system.

8.3.1 Formal Model

Consider a polling system with B queues, and $K = 2B$ classes. The server visits the queues in a cyclic manner and uses a gated discipline within each queue. Classes served at machine 1 in queues $1, 2, \ldots, B$ are numbered $1, 2, \ldots, \frac{K}{2}$; classes served at machine 2 in queues $1, 2, \ldots, B$

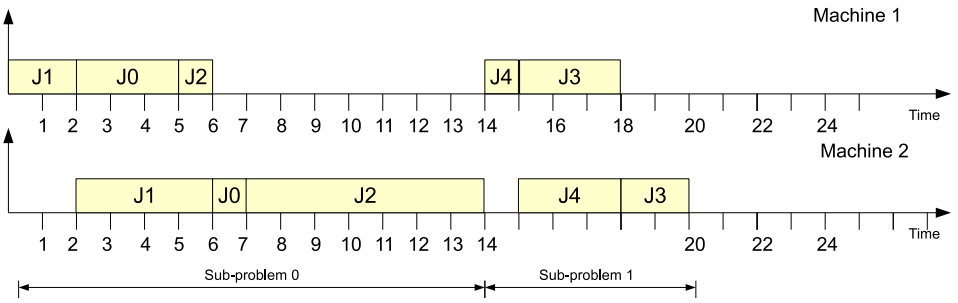

Figure 8.7: Schedule for the *completionTime* approach.

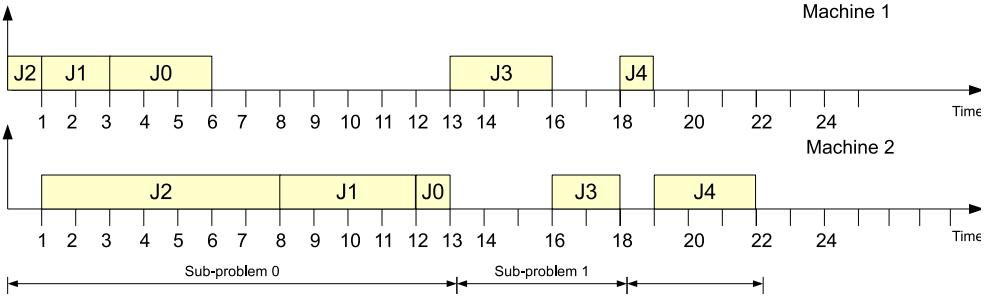

Figure 8.8: Schedule for *makespan* for the same problem instance as in Figure 8.7.

are numbered $\frac{K}{2}+1, \frac{K}{2}+2, \ldots, K$. We use notation similar to that of Dai (1995) to further describe the system. Jobs arrive at class k, $k = 1, 2, \ldots, \frac{K}{2}$, according to exogenous arrival processes with inter-arrival times $\xi_k = \{\xi_k(j), j \geq 1\}$, where j is the job number. The processing times of class k jobs are $\eta_k = \{\eta_k(j), j \geq 1\}$ for $k = 1, 2, \ldots, K$. As in the paper by Dai (1995), we make the following assumptions on the inter-arrival and processing times:

1. $\xi_1, \xi_2, \ldots, \xi_K, \eta_1, \eta_2, \ldots, \eta_K$ are mutually independent i.i.d. sequences;
2. $E[\xi_k(1)] < \infty$ for $k \in \{1, 2, \ldots, \frac{K}{2}\}$ and $E[\eta_k(1)] < \infty$ for $k = 1, 2, \ldots, K$;
3. Inter-arrival times are unbounded and spread-out: for each $k \in \{1, 2, \ldots, \frac{K}{2}\}$, there exists an integer $c_k > 0$ and some function $p_k(x) \geq 0$ on $\mathbb{R}^+$ with $\int_0^\infty p_k(x)dx > 0$, such that

$$P\{\xi_k(1) \geq x\} > 0 \text{ for any } x > 0 \text{ and} \tag{8.1}$$

$$P\{a \leq \sum_{j=1}^{c_k} \xi_k(j) \leq b\} \geq \int_a^b p_k(x)dx \text{ for any } 0 \leq a < b. \tag{8.2}$$

These assumptions allow us to apply Theorem 4.2 of the paper by Dai (1995), which states that if a fluid limit model of a particular queueing discipline is stable, then the Markov chain

describing the dynamics of the underlying system is positive Harris recurrent under this discipline. As noted above, positive Harris recurrence implies stability of the original queueing system. We further denote $E[\xi_k(1)] = \frac{1}{\lambda_k}$ and $E[\eta_k(1)] = \frac{1}{\mu_k}$.[6]

The server is composed of two machines in tandem: every job needs to be processed first by machine 1 and then by machine 2. Let $s(k)$ denote the machine that processes class k. Class l, for $l = 1, 2, \ldots, \frac{K}{2}$, is served by machine 1 ($s(l) = 1$), and, upon completion on machine 1 turns into class $k = l + \frac{K}{2}$, which is served by machine 2 ($s(k) = 2$). $\mathcal{C}_m$ denotes the set of classes served by machine m, so that $\mathcal{C}_1 = \{1, \ldots, \frac{K}{2}\}$ and $\mathcal{C}_2 = \{\frac{K}{2} + 1, \ldots, K\}$. If $\frac{1}{\mu_k} \geq \frac{1}{\mu_{(k+B)}}$ for class k, $k = 1, 2, \ldots, \frac{K}{2}$, then machine 1 is the bottleneck for the corresponding queue $b = k$; otherwise, machine 2 is the bottleneck for this queue.[7] The probability that a class l job, upon completion, turns into a class k job is $\Phi_k^l(j)$, and equals 1 if $k = l + \frac{K}{2}$, and 0 otherwise.

Let $s_{b,i}$ be the start time of sub-problem i in queue b when *FCFS* is used to schedule jobs within the sub-problems. (We number sub-problems starting at 0 for each queue.) Let $\tau_{b,i}$ and $t_{b,i}$ be the completion times of sub-problem i in queue b on machine 1 and machine 2, respectively. By the nature of a two-machine flow shop, $\tau_{b,i} < t_{b,i}$.

8.3.2 Stability of *FCFS*

In this section, we use the fluid methodology of Dai (1995) to prove stability of *FCFS* in the system described above. We start by providing a representation for the underlying Markov process and defining the overall network dynamics. The proof of stability is divided into two parts: we first consider the fluid dynamics of each individual sub-problem and then combine them via a function that represents the total workload on the bottleneck machine.

8.3.2.1 State Definition

The underlying Markov process is based on the following state representation:

$$X(t) \quad = \quad (J(t), \mathbb{Q}_1(t), \mathbb{Q}_2(t), \mathbb{A}(t), \mathbb{M}(t)), \tag{8.3}$$

where

- $J(t)$ is the queue being served at time t,

[6] Note that, unlike in the previous chapter and unlike in the previous section, the subscript k on μ now refers to the class (not machine).

[7] Since classes served by machine 1 and queues are numbered in exactly the same way, we use the two interchangeably.

- $\mathbb{Q}_1(t) = (v_1(t), N_1(t))$, where $v_1(t)$ is the residual processing time on machine 1, and $N_1(t)$ is the total number of jobs in the current sub-problem on machine 1,

- $\mathbb{Q}_2(t) = (v_2(t), Q_2(t))$, where $v_2(t)$ is the residual processing time on machine 2, and $Q_2(t)$ is the total number of jobs at machine 2,

- $\mathbb{A}(t) = (a_1(t), a_2(t), \ldots, a_B(t))$, where $a_b(t)$ is the residual inter-arrival time to queue b,

- $\mathbb{M}(t) = (M_1(t), M_2(t), \ldots, M_B(t))$, where $M_b(t)$ is the total number of jobs at queue b that are going to be part of the next sub-problem.

In the case of inter-arrival times being exponentially distributed for all queues, $\mathbb{A}(t)$ does not need to be included in the state definition.

Let the state space be denoted by $\mathcal{X}$ and the initial state of the network be $x \in \mathcal{X}$. $\mathcal{X} \subset \mathbb{Z}_+^{3+B} \times \mathbb{R}_+^{2+B}$, where $\mathbb{Z}_+$ is the set of non-negative integers and $\mathbb{R}_+$ denotes the non-negative real numbers. The notation $|x|$ is used to describe the "norm" of x, defined as the sum of the norms of all components of x.

8.3.2.2 Overall Network Dynamics

In order to state the overall network dynamics, we use some additional notation:

- $E_k^x(t) := \max_{j \geq 1}\{\xi_k(1) + \cdots + \xi_k(j) \leq t\}$. That is, $E_k^x(t)$ is the cumulative number of external arrivals to the system (i.e., to class k) by time t, $k = 1, 2, \ldots, \frac{K}{2}$,

- $S_k^x(t) := \max_{j \geq 1}\{\eta_k(1) + \cdots + \eta_k(j) \leq t\}$ is the cumulative number of departures (service completions) from class k, $k = 1, 2, \ldots, K$, if the server is busy for t time units in $[0, t]$,

- $T_k^x(t)$ is the cumulative time that server $s(k)$ has spent on class k jobs in $[0, t]$, for $k = 1, 2, \ldots, K$,

- $I_m^x(t)$ is the cumulative idle time of machine m, $m = 1, 2$,

- $Q_k^x(t)$ is the total number of jobs (waiting or in service, and part of the current or next sub-problem) in class k, $k = 1, 2, \ldots, K$, at time t.

The overall dynamics of the system are:

$$Q_k^x(t) = Q_k^x(0) + E_k^x(t) + \sum_{l=1}^{K} \Phi_k^l(S_l^x(T_l^x(t))) - S_k^x(T_k^x(t)), \quad k = 1, \ldots, K \tag{8.4}$$

$$Q^x(t) = (Q_1^x(t), \ldots, Q_K^x(t))^T \geq (0, \ldots, 0)^T \tag{8.5}$$

$$T^x(\cdot) = (T_1^x(t), \ldots, T_K^x(t))^T \text{ is non-decreasing in } t \tag{8.6}$$

$$T^x(0) = (0, \ldots, 0)^T \tag{8.7}$$

$$I_m^x(t) = t - \sum_{k \in \mathcal{C}_m} T_k^x(t) \text{ is non-decreasing in } t, \quad m = 1, 2 \tag{8.8}$$

$$\dot{T}_k^x(t) = 1 \text{ iff } s_{b,i} \leq t \leq \tau_{b,i} \text{ for some } i, \quad k = 1, 2, \ldots, \frac{K}{2}, \ k \text{ belonging to queue } b \tag{8.9}$$

$$\int_0^\infty (\sum_{k \in \mathcal{C}_2} Q_k^x(t)) dI_2^x(t) = 0. \tag{8.10}$$

Equation (8.4) states that, given an initial system state x, the number of jobs in class k at time t equals the initial number of jobs at time 0 plus all of the external and internal arrivals to class k by time t, minus the jobs that have left class k by t. Equation (8.5) ensures that the number of jobs at each class is non-negative. Equations (8.6) and (8.7) state that the cumulative busy time for each class is non-decreasing and initially equal to 0, respectively. Equation (8.8) defines the cumulative idle time of each machine as the difference between the total elapsed time and the total time that machine m has been busy. Equation (8.9) is a non-idleness condition for machine 1: the instantaneous allocation of time to class k ($\dot{T}_k^x(t)$ is the derivative of $T_k^x(t)$ with respect to t) is equal to 1 only for those time points t that occur within some sub-problem on machine 1. Equation (8.10) is a non-idleness condition for machine 2: the idleness of machine 2 can increase only if it is not processing any jobs and there are no jobs waiting.

We cannot derive a fluid limit model that would represent the overall system dynamics directly from the system dynamics equations above because we cannot apply the Strong Law of Large Numbers to derive the appropriate limits for all quantities. Specifically, there is a dependence between the departure process of machine 1 and the processing times on machine 2: the processing times on machine 2 in sub-problem i influence when the next sub-problem will start on machine 1 and therefore the number of jobs that will be processed in the subsequent sub-problem on both machine 1 and machine 2.

In order to examine the dynamics of each individual sub-problem, we define the following "sub-problem equivalents" of the above notation. For sub-problem i:

- $S_k^{x,i}(t) := \max_{j\geq 1}\{\eta_k(1) + \cdots + \eta_k(j) \leq t\}$ is the cumulative number of departures (service completions) from class k, $k = 1, 2, \ldots, K$, if the machine $s(k)$ is busy for t time units in $[s_{b,i}, s_{b,i} + t]$, and where we assume that the indexing of jobs is restarted from 1 at the beginning of each sub-problem,
- $T_k^{x,i}(t)$ is the cumulative time that machine $s(k)$ has spent on class k jobs in $[s_{b,i}, t]$, for $k = 1, 2, \ldots, K$ and $s_{b,i} \leq t \leq t_{b,i}$,
- $I_m^{x,i}(t)$ is the cumulative idle time of machine m, $m = 1, 2$, in $[s_{b,i}, t]$, $s_{b,i} \leq t \leq t_{b,i}$,
- $Q_k^{x,i}(t)$ is the total number of jobs in class k, $k = 1, 2, \ldots, K$, *that are part of the current sub-problem* at time t, $s_{b,i} \leq t \leq t_{b,i}$.

We now state and prove the stability condition for *FCFS*.

8.3.2.3 Proof

Theorem 8.3.1. *If* $\sum_{k=1}^{B} \frac{\lambda_k}{\min\{\mu_k,\mu_{k+B}\}} < 1$ *then the polling system described above is stable under* FCFS.

Proof. Suppose the initial state of the system is x, queue 1 is the initial queue to receive service, and there are y_0 jobs present at the queue at time 0 (y_0 at machine 1, 0 at machine 2).

We are going to focus on this initial sub-problem and ignore new arrivals to the system. Thus, for $t \in [s_{1,0}, t_{1,0}]$, where $s_{1,0} = 0$, the dynamics of *sub-problem 0* (ignoring arrivals) are:

$$
\begin{aligned}
Q_1^{x,0}(t) &= y_0 - S_1^{x,0}(T_1^{x,0}(t)) && (8.11)\\
Q_{(B+1)}^{x,0}(t) &= S_1^{x,0}(T_1^{x,0}(t)) - S_{(B+1)}^{x,0}(T_{(B+1)}^{x,0}(t)) && (8.12)\\
Q_1^{x,0}(t) \geq 0, &\quad Q_{(B+1)}^{x,0}(t) \geq 0 && (8.13)\\
T^{x,0}(\cdot) &= (T_1^{x,0}(\cdot), T_{(B+1)}^{x,0}(\cdot)) \text{ is non-decreasing} && (8.14)\\
T_1^{x,0}(s_{1,0}) = 0, &\quad T_{(B+1)}^{x,0}(s_{1,0}) = 0 && (8.15)\\
I_1^{x,0}(t) &= t - T_1^{x,0}(t) \text{ is non-decreasing} && (8.16)\\
I_2^{x,0}(t) &= t - T_{(B+1)}^{x,0}(t) \text{ is non-decreasing} && (8.17)\\
\int_{s_{1,0}}^{\tau_{1,0}} Q_1^{x,0}(t) dI_1^{x,0}(t) &= 0 && (8.18)\\
\int_{s_{1,0}}^{t_{1,0}} Q_{(B+1)}^{x,0}(t) dI_2^{x,0}(t) &= 0. && (8.19)
\end{aligned}
$$

Equation (8.11) states that the number of jobs present in class 1 (i.e., at machine 1 of queue 1) that are part of sub-problem 0 equals the initial number of jobs, y_0, minus the number of departures by time t. Equation (8.12) defines the number of jobs present in class $B+1$ (i.e., at machine 2 of queue 1) at time t in terms of the difference between the number of jobs that have arrived at this class by time t and the number of jobs that have left this class by time t. Equation (8.13) states the fact that the queue lengths cannot be negative. Equations (8.14) and (8.15) ensure that the cumulative time spent on class k by machine $s(k)$ is non-decreasing and equal to 0 at the start of sub-problem 0, respectively. Equations (8.16) and (8.17) define the cumulative idleness of the two machines by time t. Equations (8.18) and (8.19) are the non-idleness constraints.

Since jobs within the sub-problem are scheduled using the *FCFS* policy, there is no dependence between the processing times and the order in which the jobs are scheduled. Within a sub-problem, there is also no dependence between the processing times on machine 2 and the start of jobs on machine 1 (such dependence would arise, e.g., under Johnson's rule, or if we were considering the joint dynamics of two or more sub-problems, as mentioned in Section 8.3.2.2). Thus, we can apply the Strong Law of Large Numbers to obtain a fluid limit model.

Specifically, we consider the processes $\bar{Q}_k^{x,0}(t)$ and $\bar{T}_k^{x,0}(t)$ for $k = 1$ and $k = B+1$, defined as

$$\bar{Q}_k^{x,0}(t) = \frac{1}{|x|} Q_k^{x,0}(|x|t) \quad \text{and} \quad \bar{T}_k^{x,0}(t) = \frac{1}{|x|} T_k^{x,0}(|x|t). \tag{8.20}$$

If we let $y_0 \to \infty$ while keeping the remaining components of x constant, we obtain an undelayed fluid limit $(\bar{Q}_1^0(t), \bar{Q}_{(B+1)}^0(t), \bar{T}_1^0(t), \bar{T}_{(B+1)}^0(t))$. The fluid limit is a solution to Equations (8.21)–(8.29) (Dai, 1995). $\bar{s}_{b,i}$ denotes the start time of sub-problem i in queue b on the fluid scale. $\bar{\tau}_{b,i}$ and $\bar{t}_{b,i}$ denote the completion times of sub-problem i on the fluid scale on machine 1 and machine 2, respectively. Note that $\bar{s}_{1,0} = 0$. As a result of our assumptions and scaling,

$\bar{Q}^0_1(0) = 1$ and $\bar{Q}^0_{(B+1)}(0) = 0$.

$$\begin{aligned}
\bar{Q}^0_1(t) &= 1 - \mu_1 \bar{T}^0_1(t) && (8.21)\\
\bar{Q}^0_{(B+1)}(t) &= \mu_1 \bar{T}^0_1(t) - \mu_{(B+1)} \bar{T}^0_{(B+1)}(t) && (8.22)\\
\bar{Q}^0(t) &= (\bar{Q}^0_1(t), \bar{Q}^0_{(B+1)}(t))^T \geq 0 && (8.23)\\
\bar{T}^0(\cdot) &= (\bar{T}^0_1(\cdot), \bar{T}^0_{(B+1)}(\cdot)) \text{ is non-decreasing} && (8.24)\\
\bar{T}^0_1(\bar{s}_{1,0}) = 0, &\quad \bar{T}^0_{(B+1)}(\bar{s}_{1,0}) = 0 && (8.25)\\
\bar{I}^0_1(t) &= t - \bar{T}^0_1(t) \text{ is non-decreasing} && (8.26)\\
\bar{I}^0_2(t) &= t - \bar{T}^0_{(B+1)}(t) \text{ is non-decreasing} && (8.27)\\
\int_{\bar{s}_{1,0}}^{\bar{\tau}_{1,0}} \bar{Q}^0_1(t) d\bar{I}^0_1(t) &= 0 && (8.28)\\
\int_{\bar{s}_{1,0}}^{\bar{t}_{1,0}} \bar{Q}^0_{(B+1)}(t) d\bar{I}^0_2(t) &= 0 && (8.29)
\end{aligned}$$

We can now analyze the behaviour of this sub-problem on the fluid scale. There two possible cases: either $\frac{1}{\mu_1} \geq \frac{1}{\mu_{(B+1)}}$ so that machine 1 is the bottleneck, or $\frac{1}{\mu_1} < \frac{1}{\mu_{(B+1)}}$ so that machine 2 is the bottleneck.

Case 1: Machine 1 Bottleneck Firstly, since the service discipline is non-idling, Equation (8.21) implies that $\bar{Q}^0_1(\frac{1}{\mu_1}) = 0$. In other words, the queue of machine 1 empties at time $\frac{1}{\mu_1}$, or, equivalently, machine 1 completes sub-problem 0 at $\frac{1}{\mu_1}$.

Secondly, we consider the behaviour of machine 2. Let $\dot{\bar{T}}^i_k(t) = \frac{d}{dt}\bar{T}^i_k(t)$ if this derivative exists. For $t \in [0, \frac{1}{\mu_1})$, $\dot{\bar{T}}^0_1(t) = 1$ since machine 1 is non-idling and there is work in its queue, while for $t \in [\frac{1}{\mu_1}, \bar{t}_{1,0}]$, $\dot{\bar{T}}^0_1(t) = 0$ since machine 1 finishes sub-problem 0 at $t = \frac{1}{\mu_1}$ and there are no new arrivals within the sub-problem. It is known that

(a) if $\bar{Q}^0_{(B+1)}(t) > 0$ for $t \in [0, \frac{1}{\mu_1})$, then $\dot{\bar{T}}^0_{(B+1)}(t)$ exists and equals 1, and $\dot{\bar{Q}}^0_{(B+1)}(t) = \mu_1 \dot{\bar{T}}^0_1(t) - \mu_{(B+1)} \dot{\bar{T}}^0_{(B+1)}(t) = \mu_1 - \mu_{(B+1)} < 0$.

(b) if $\bar{Q}^0_{(B+1)}(t) > 0$ for $t \in [\frac{1}{\mu_1}, \bar{t}_{1,0}]$, then $\dot{\bar{T}}^0_{(B+1)}(t)$ exists and equals 1, and $\dot{\bar{Q}}^0_{(B+1)}(t) = \mu_1 \dot{\bar{T}}^0_1(t) - \mu_{(B+1)} \dot{\bar{T}}^0_{(B+1)}(t) = -\mu_{(B+1)} < 0$.

Given properties (a) and (b), and the fact that $\bar{Q}^0_{(B+1)}(0) = 0$, it follows that $\bar{Q}^0_{(B+1)}(t) \equiv 0$ for $0 \leq t \leq \bar{t}_{1,0}$ by Lemma 5.2 of Dai (1995).

Therefore, on the fluid scale, when sub-problem 0 of queue 1 finishes on machine 1, it also finishes on machine 2, i.e., $\bar{t}_{1,0} = \frac{1}{\mu_1}$.

Case 2: Machine 2 Bottleneck Machine 1 will complete sub-problem 0 of queue 1 at time $\frac{1}{\mu_1}$ and $\bar{Q}_1^0(t)$ will be 0 for $\frac{1}{\mu_1} \leq t \leq \bar{t}_{1,0}$. (Also, $\dot{\bar{T}}_1^0(t) = 1$ for $0 \leq t < \frac{1}{\mu_1}$, $\dot{\bar{T}}_1^0(t) = 0$ for $\frac{1}{\mu_1} \leq t \leq \bar{t}_{1,0}$.) Machine 2 will complete sub-problem 0 of queue 1 at time $\frac{1}{\mu_{B+1}}$. Hence, $\bar{t}_{1,0} = \frac{1}{\mu_{B+1}}$ and $\bar{Q}_1^0(\bar{t}_{1,0}) = \bar{Q}_{(B+1)}^0(\bar{t}_{1,0}) = 0$.

For both cases, we have shown that $\bar{Q}_1^0(\bar{t}_{1,0}) = 0$ and $\bar{Q}_{(B+1)}^0(\bar{t}_{1,0}) = 0$. Now, we show that this implies that, on the fluid scale, the next sub-problem starts immediately.

Since $\bar{Q}_1^0(\bar{t}_{1,0}) = 0$ and $\bar{Q}_{(B+1)}^0(\bar{t}_{1,0}) = 0$, we know that at $\tau_{1,0}$, there is a finite number of jobs, γ, present at machine 2. For $\tau_{1,0} \leq t \leq t_{1,0}$, machine 2 operates as a non-idling single-server queue with a finite number of jobs present at $\tau_{1,0}$, and no new arrivals. Therefore, these γ jobs are going to be finished in a finite amount of time ($t_{1,0} - \tau_{1,0}$ is finite), which implies that, on the fluid scale, the next sub-problem is going to start immediately (e.g., $\bar{s}_{2,0} = \bar{t}_{1,0}$ if there are at least two queues and the second queue is not empty at $\bar{t}_{1,0}$).

For each subsequent sub-problem i, a fluid model can be constructed similarly by assuming that we set the initial number of jobs (on the original scale) to $Q_k^x(s_{b,i})$ and applying the same scaling as for sub-problem 0.

We now need to show that there exists a time t^* at which the fluid level in all queues becomes 0 and stays at 0. We do so by using a function that represents the total bottleneck workload at time t and by considering the overall system dynamics, not just individual sub-problems.

Recall that the above sub-problem analysis assumes that, at the start of each sub-problem, there is a set of jobs that needs to be processed – jobs that have arrived at the corresponding queue since the time the previous sub-problem for this queue started. We can look at this arrival process and characterize its behaviour on the fluid scale. Specifically, we know that $E_k^x(t)$ is the number of arrivals by time t to class k and, if we apply the same scaling as used above in the sub-problem analysis, we obtain

$$\bar{E}_k^x(t) = \frac{1}{|x|} E_k^x(|x|t) \quad \text{and} \quad \lim_{|x|\to\infty} \frac{1}{|x|} E_k^x(|x|t) = \lambda_k t. \tag{8.30}$$

Let B_m denote the set of queues for which machine m is the bottleneck. We can then define the total bottleneck workload at time t as

$$\bar{z}(t) = \sum_{k\in B_1} \frac{\bar{Q}_k(t)}{\mu_k} + \sum_{k\in B_2} \frac{\bar{Q}_k(t) + \bar{Q}_{k+B}(t)}{\mu_{k+B}}. \tag{8.31}$$

If the derivative of $\bar{z}(t)$ exists, then it is defined as

$$\dot{\bar{z}}(t) = \sum_{k \in B_1} \frac{\dot{\bar{Q}}_k(t)}{\mu_k} + \sum_{k \in B_2} \frac{\dot{\bar{Q}}_k(t) + \dot{\bar{Q}}_{k+B}(t)}{\mu_{k+B}}. \tag{8.32}$$

Suppose $\bar{Q}_k(t) > 0$ for at least one k. Then it must be that the server is at one of these queues, e.g., queue b^* (corresponding to classes $k^* = b^*$ and $k^* + B$).

If the server is at queue b^* at time t, then for all queues $b \neq b^*$, there can be no fluid leaving the queue at t, but there is arriving fluid, so that $\dot{\bar{Q}}_k(t) = \lambda_k$ and $\dot{\bar{Q}}_{k+B}(t) = 0$ for classes k and $k + B$ that are part of queue b.

If queue $b^* \in B_1$, then $\dot{\bar{T}}^i_{k^*}(t) = \dot{\bar{T}}_{k^*}(t) = 1$ for $t \in [\bar{s}_{b^*,i}, \bar{t}_{b^*,i})$. In this case,

$$\begin{aligned} \dot{\bar{Q}}_{k^*}(t) &= \lambda_{k^*} - \mu_{k^*} \dot{\bar{T}}_{k^*}(t) && (8.33) \\ &= \lambda_{k^*} - \mu_{k^*}, && (8.34) \end{aligned}$$

and

$$\begin{aligned} \dot{\bar{z}}(t) &= \frac{\lambda_{k^*} - \mu_{k^*}}{\mu_{k^*}} + \sum_{k \in B_1 \setminus k^*} \frac{\lambda_k}{\mu_k} + \sum_{k \in B_2} \frac{\lambda_k}{\mu_{k+B}} && (8.35) \\ &= \frac{\lambda_{k^*}}{\mu_{k^*}} - 1 + \sum_{k \in B_1 \setminus k^*} \frac{\lambda_k}{\mu_k} + \sum_{k \in B_2} \frac{\lambda_k}{\mu_{k+B}} && (8.36) \\ &< 0, \text{ since } \sum_{k=1}^{B} \frac{\lambda_k}{\min\{\mu_k, \mu_{k+B}\}} < 1. && (8.37) \end{aligned}$$

If queue $b^* \in B_2$, then for $t \in [\bar{s}_{b^*,i}, \bar{\tau}_{b^*,i})$, $\dot{\bar{T}}^i_{k^*}(t) = \dot{\bar{T}}_{k^*}(t) = 1$ and $\dot{\bar{T}}^i_{k^*+B}(t) = \dot{\bar{T}}_{k^*+B}(t) = 1$. In this case,

$$\begin{aligned} \dot{\bar{Q}}_{k^*}(t) &= \lambda_{k^*} - \mu_{k^*} \dot{\bar{T}}_{k^*}(t) && (8.38) \\ &= \lambda_{k^*} - \mu_{k^*} && (8.39) \\ \dot{\bar{Q}}_{k^*+B}(t) &= \mu_{k^*} \dot{\bar{T}}_{k^*}(t) - \mu_{k^*+B} \dot{\bar{T}}_{k^*+B}(t) && (8.40) \\ &= \mu_{k^*} - \mu_{k^*+B}, && (8.41) \end{aligned}$$

and

$$\dot{\bar{z}}(t) = \frac{(\lambda_{k^*} - \mu_{k^*}) + (\mu_{k^*} - \mu_{k^*+B})}{\mu_{k^*+B}} + \sum_{k\in B_1} \frac{\lambda_k}{\mu_k} + \sum_{k\in B_2\setminus k^*} \frac{\lambda_k}{\mu_{k+B}} \tag{8.42}$$

$$= \frac{\lambda_{k^*}}{\mu_{k^*+B}} - 1 + \sum_{k\in B_1} \frac{\lambda_k}{\mu_k} + \sum_{k\in B_2\setminus k^*} \frac{\lambda_k}{\mu_{k+B}} \tag{8.43}$$

$$< 0, \text{ since } \sum_{k=1}^{B} \frac{\lambda_k}{\min\{\mu_k, \mu_{k+B}\}} < 1. \tag{8.44}$$

If queue $b^* \in B_2$, then for $t \in [\bar{\tau}_{b^*,i}, \bar{t}_{b^*,i})$, $\dot{\bar{T}}^i_{k^*}(t) = \dot{\bar{T}}_{k^*}(t) = 0$ and $\dot{\bar{T}}^i_{k^*+B}(t) = \dot{\bar{T}}_{k^*+B}(t) = 1$, so

$$\dot{\bar{Q}}_{k^*}(t) = \lambda_{k^*} - \mu_{k^*}\dot{\bar{T}}_{k^*}(t) \tag{8.45}$$

$$= \lambda_{k^*} \tag{8.46}$$

$$\dot{\bar{Q}}_{k^*+B}(t) = \mu_{k^*}\dot{\bar{T}}_{k^*}(t) - \mu_{k^*+B}\dot{\bar{T}}_{k^*+B}(t) \tag{8.47}$$

$$= -\mu_{k^*+B}, \tag{8.48}$$

and

$$\dot{\bar{z}}(t) = \frac{\lambda_{k^*} - \mu_{k^*+B}}{\mu_{k^*+B}} + \sum_{k\in B_1} \frac{\lambda_k}{\mu_k} + \sum_{k\in B_2\setminus k^*} \frac{\lambda_k}{\mu_{k+B}} \tag{8.49}$$

$$= \frac{\lambda_{k^*}}{\mu_{k^*+B}} - 1 + \sum_{k\in B_1} \frac{\lambda_k}{\mu_k} + \sum_{k\in B_2\setminus k^*} \frac{\lambda_k}{\mu_{k+B}} \tag{8.50}$$

$$< 0, \text{ since } \sum_{k=1}^{B} \frac{\lambda_k}{\min\{\mu_k, \mu_{k+B}\}} < 1. \tag{8.51}$$

We have shown that $\dot{\bar{z}}(t) < 0$ whenever there exists at least one k such that $\bar{Q}_k(t) > 0$. This is equivalent stating that $\dot{\bar{z}}(t) < 0$ whenever $z(t) > 0$. It follows that $\bar{z}(t) \equiv 0$ for $t \geq \frac{\bar{z}(0)}{1-\sum_{k=1}^{B} \frac{\lambda_k}{\min\{\mu_k,\mu_{k+B}\}}}$ by Lemma 5.2 of Dai (1995).

Consequently, there is a time point $t^* = \frac{\bar{z}(0)}{1-\sum_{k=1}^{B} \frac{\lambda_k}{\min\{\mu_k,\mu_{k+B}\}}}$ at which the total bottleneck workload in the system reaches 0 and after which it remains at 0. This implies that if $\sum_{k=1}^{B} \frac{\lambda_k}{\min\{\mu_k,\mu_{k+B}\}} < 1$, then the system is stable. □

The above proof applies to any non-idling policy that does not use processing time information. In particular, under any such policy, we can write the fluid model of the system dynamics for one sub-problem in the same way as for *FCFS*, since there is neither dependence between

the processing times and the order in which the jobs are scheduled nor dependence between the processing times on machine 2 and the start of jobs on machine 1. The steps of the proof remain the same because on the fluid scale, there is no difference in the behaviour of non-idling policies that do not use processing times.

8.3.3 Instability Example

After proving stability of *FCFS* and observing that the same proof applies to all non-idling policies that do not use processing time information, a natural question is whether all non-idling policies (i.e., even those using processing times) are stable under the stability conditions of *FCFS*. Consider a polling system with five queues: at time 0, there are 1000 jobs in queue 1, and no jobs in the other queues. Jobs arrive at each queue according to (independent) Poisson processes with rate 1. Processing times are exponentially distributed with rate 6. Therefore, for each machine and a given queue, the load is $\frac{1}{6} < 1$, and the load on the system is $5(\frac{1}{6}) = 0.833 < 1$. In Figure 8.9, we show the number of jobs in queue 1 over time for four policies: *FCFS*, *makespan*, SPT_{sum} and *reverse*. The *reverse* policy is a modification of Johnson's rule: set II is scheduled before set I (i.e., in reverse order as compared to Johnson's rule).

In Figure 8.9, we see that all policies initially have a large number of jobs at machine 2 in queue 1 – this corresponds to the sub-problem containing the initial 1000 jobs. Under *FCFS*, *makespan* and SPT_{sum} there is little variation in the number of jobs at machine 2 after approximately 150,000 time units: these policies keep the number of jobs below 25. For the *reverse* policy, on the contrary, the number of jobs in sub-problems following the first one grows over time. After 250,000 time units, the number of jobs in queue 1 is greater than in the initial sub-problem. The figure therefore suggests that the number of jobs in queue 1 for *FCFS*, *makespan* and SPT_{sum} will remain bounded, corresponding to stable behaviour; for the *reverse* policy, the increasing number of jobs in queue 1 suggests instability and indicates that not all non-idling policies are stable when *FCFS* is stable. We formally prove the instability of *reverse* for the system described in this example in Section 9.2.3 of the next chapter.

8.3.4 Stability of the *makespan* Approach

In the previous section, we have proved stability of *FCFS* in a polling system with a two-machine flow shop server. We have subsequently shown that non-idling policies that use processing time information may be unstable under the stability conditions of *FCFS*. Now, we investigate the stability of the *makespan* approach.

Let $s_{b,i}^{makespan}$ be the start time of sub-problem i in queue b when *makespan* is used to schedule jobs within the sub-problems. Recall that the *makespan* approach uses Johnson's rule

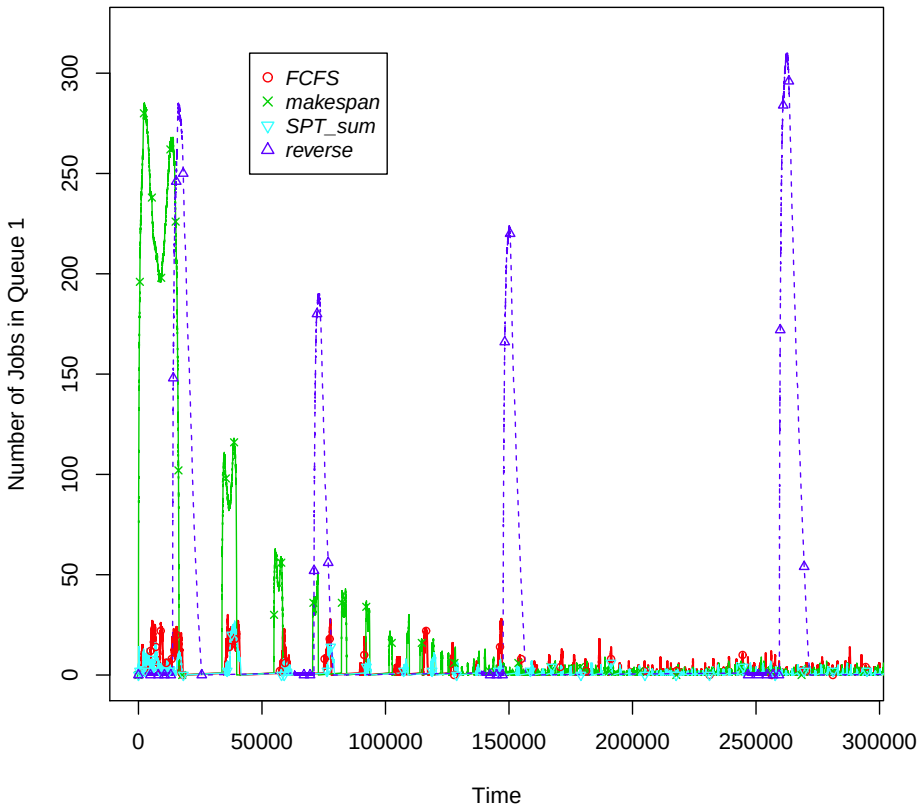

Figure 8.9: Number of Jobs at Machine 2 in Queue 1 Over Time.

to minimize the makespan of each sub-problem. Let $\tau_{b,i}^{makespan}$ and $t_{b,i}^{makespan}$ be the completion times of sub-problem i in queue b on machine 1 and machine 2, respectively. By the nature of a two-machine flow shop, $\tau_{b,i}^{makespan} < t_{b,i}^{makespan}$. Similarly, $s_{b,i}^{FCFS}$ denotes the start of the ith sub-problem of queue b under *FCFS*, and $\tau_{b,i}^{FCFS}$ and $t_{b,i}^{FCFS}$ denote the completion time of the ith sub-problem of queue b on machine 1 and machine 2, respectively, under *FCFS*.

We use square brackets to denote the position of a job in the sequence specified by the policy. For example, $\eta_1([7])$ is the machine 1 processing time of the job sequenced in the seventh position of the current sub-problem.

8.3.4.1 State Definition

Under the *makespan* approach, the state representation is more complex than under *FCFS*, since it is necessary to keep track of the processing times of the jobs at each machine. The

state representation is as follows:

$$X(t) = (J(t), \mathbb{Q}_1(t), \mathbb{Q}_2(t), \mathbb{A}(t), \mathbb{M}(t)), \tag{8.52}$$

where

$$\mathbb{Q}_1(t) = (v_1(t), \eta_1([L_1(t)+2]), \ldots, \eta_1([L_1(s_{J(t),i}) + Q_{J(t)}(s_{J(t),i})])) \tag{8.53}$$

$$\mathbb{Q}_2(t) = (v_2(t), \eta_2([L_2(t)+2]), \ldots, \eta_2([L_1(t)])) \tag{8.54}$$

and

- $J(t)$ is the queue being served at time t,
- $v_m(t)$ is the residual processing time on machine m,
- $L_m(t)$ is the number of jobs processed on machine m by time t,
- The vector $\mathbb{Q}_1(t)$ states the processing times of jobs, on machine 1, *of the current sub-problem* in the sequence in which these jobs will be processed: $v_1(t)$ is the remaining processing time of the job currently being processed on machine 1 (the job that is in position $L_1(t)+1$ of the overall schedule); $\eta_1([L_1(t)+2])$ is the processing time of the job in the next position in the schedule for the current sub-problem. The last job in this vector for machine 1 is the job in position $L_1(s_{J(t),i}) + Q_{J(t)}(s_{J(t),i})$, since $L_1(s_{J(t),i})$ is the number of jobs completed by the start of the current sub-problem and $Q_{J(t)}(s_{J(t),i})$ is the number of jobs in the queue of machine 1 at that time point,
- The vector $\mathbb{Q}_2(t)$ states the processing times of jobs, on machine 2, in the sequence in which these jobs will be processed: $v_2(t)$ is the remaining processing time of the job currently being processed on machine 2 (the job in position $L_2(t)+1$ of the overall schedule); $\eta_2([L_2(t)+2])$ is the processing time of the job in the next position in the schedule for the current sub-problem. The last job in this vector for machine 2 is the job in position $L_1(t)$ since $L_1(t)$ jobs have arrived at machine 2 by t,
- $\mathbb{A}(t) = (a_1(t), a_2(t), \ldots, a_B(t))$, where $a_b(t)$ is the residual inter-arrival time to queue b,
- $\mathbb{M}(t) = (M_1(t), M_2(t), \ldots, M_B(t))$, where $M_b(t)$ is the total number of jobs at queue b that are going to be part of the next sub-problem.

In the case of inter-arrival times being exponentially distributed for all queues, $\mathbb{A}(t)$ does not need to be included in the state definition.

Let the state space be denoted by $\mathcal{X}$ and the initial state of the network be $x \in \mathcal{X}$. $\mathcal{X} \subset (\mathbb{R}_+^\infty)^2 \times \mathbb{Z}_+^B \times \mathbb{R}_+^B$, where $\mathbb{R}_+^\infty$ is the set of sequences taking values is $\mathbb{R}_+^\infty$, $\mathbb{Z}_+$ is the set of non-negative integers, $\mathbb{R}_+$ denotes the non-negative real numbers. The notation $|x|$ is used to describe the "norm" of x, defined as the sum of the norms of all components of x.

8.3.4.2 System Dynamics

We use the same notation as in Section 8.3.2.2 with one exception. Instead of $S_k^x(t)$, we define $D_k^x(t) := \max_{j\geq 1}\{\eta_k([1]) + \cdots + \eta_k([j]) \leq t\}$ as the cumulative number of departures (service completions) from class k, $k = 1, 2, \ldots, K$, by time t if Johnson's rule is employed. The system dynamics can then be represented as:

$$Q_k^x(t) = Q_k^x(0) + E_k^x(t) + \sum_{l=1}^{K} \Phi_k^l(D_l^x(t)) - D_k^x(t), \quad k = 1, \ldots, K \tag{8.55}$$

$$\dot{T}_k^x(t) = 1 \text{ iff } s_{b,i}^{makespan} \leq t \leq \tau_{b,i}^{makespan} \text{ for some } i, \quad k = 1, 2, \ldots, \frac{K}{2},\ k \text{ belonging to queue } b \tag{8.56}$$

$$\text{Equations } (8.5) - (8.8), (8.10) \tag{8.57}$$

In other words, the system dynamics are identical to those under *FCFS*, except that $D_k^x(t)$ is used instead of $S_k^x(t)$ in Equation (8.4) (resulting in Equation (8.55)) and the notation in Equation (8.9) is changed to reflect the *makespan* policy (resulting in Equation (8.56)). The sub-problem equivalents of the above notation are defined in the same way as in Section 8.3.2.2.

8.3.4.3 Proof

Theorem 8.3.2. *If* $\sum_{k=1}^{B} \frac{\lambda_k}{\min\{\mu_k, \mu_{k+B}\}} < 1$*, then the system is stable under the* makespan *approach.*

Proof. Suppose the initial state of the system is x, queue 1 is the initial queue to receive service, and there are y_0 jobs present at the queue at time 0 (y_0 at machine 1, 0 at machine 2).

We are going to focus on this initial sub-problem and ignore new arrivals to the system. Thus, for $t \in [s_{1,0}^{makespan}, t_{1,0}^{makespan}]$, where $s_{1,0}^{makespan} = 0$, the dynamics of *sub-problem 0*

(ignoring arrivals) are:

$$Q_1^{x,0}(t) = y_0 - D_1^{x,0}(t) \tag{8.58}$$

$$Q_{(B+1)}^{x,0}(t) = D_1^{x,0}(t) - D_{(B+1)}^{x,0}(t) \tag{8.59}$$

$$Q_1^{x,0}(t) \geq 0, \quad Q_{(B+1)}^{x,0}(t) \geq 0 \tag{8.60}$$

$$T^{x,0}(\cdot) = (T_1^{x,0}(\cdot), T_{(B+1)}^{x,0}(\cdot)) \text{ is non-decreasing} \tag{8.61}$$

$$T_1^{x,0}(s_{1,0}^{makespan}) = 0, \quad T_{(B+1)}^{x,0}(s_{1,0}^{makespan}) = 0 \tag{8.62}$$

$$I_1^{x,0}(t) = t - T_1^{x,0}(t) \text{ is non-decreasing} \tag{8.63}$$

$$I_2^{x,0}(t) = t - T_{(B+1)}^{x,0}(t) \text{ is non-decreasing} \tag{8.64}$$

$$\int_{s_{1,0}^{makespan}}^{\tau_{1,0}^{makespan}} Q_1^{x,0}(t) dI_1^{x,0}(t) = 0 \tag{8.65}$$

$$\int_{s_{1,0}^{makespan}}^{t_{1,0}^{makespan}} Q_{(B+1)}^{x,0}(t) dI_2^{x,0}(t) = 0 \tag{8.66}$$

Consider the processes $\bar{Q}_k^{x,0}(t)$ and $\bar{T}_k^{x,0}(t)$ for $k = 1$ and $k = B + 1$, defined as

$$\bar{Q}_k^{x,0}(t) = \frac{1}{|x|} Q_k^{x,0}(|x|t) \quad \text{and} \quad \bar{T}_k^{x,0}(t) = \frac{1}{|x|} T_k^{x,0}(|x|t). \tag{8.67}$$

If we let $y_0 \to \infty$ while keeping the remaining components of x constant, we obtain an undelayed fluid limit $(\bar{Q}_1^0(t), \bar{Q}_{(B+1)}^0(t), \bar{T}_1^0(t), \bar{T}_{(B+1)}^0(t))$.

$\bar{s}_{b,i}^{makespan}$ denotes the start time of sub-problem i in queue b on the fluid scale. $\bar{\tau}_{b,i}^{makespan}$ and $\bar{t}_{b,i}^{makespan}$ denote the completion times of sub-problem i on the fluid scale on machine 1 and machine 2, respectively. Note that $\bar{s}_{1,0}^{makespan} = 0$. As a result of our assumptions and scaling, $\bar{Q}_1^0(0) = 1$ and $\bar{Q}_{(B+1)}^0(0) = 0$.

Although we cannot write down an explicit set of equations for the fluid model because of the dependency between the departure process from machine 1 and processing times, we can still analyze some of the behaviour of this sub-problem on the fluid scale.

We know that on the original scale,

- $\tau_{1,0}^{makespan} = \tau_{1,0}^{FCFS}$ since the amount of work present at the start of the sub-problem is independent of the scheduling policy and both *FCFS* and *makespan* are non-idling,
- $t_{1,0}^{makespan} \leq t_{1,0}^{FCFS}$ since the *makespan* approach results in the smallest possible sub-problem length (makespan value) for a given set of jobs,
- $\tau_{1,0}^{makespan} < t_{1,0}^{makespan}$ since we are considering a two-machine flow shop.

On the fluid scale, similar properties are true:

- $\bar{\tau}_{1,0}^{makespan} = \bar{\tau}_{1,0}^{FCFS}$, (Property 1)
- $\bar{t}_{1,0}^{makespan} \leq \bar{t}_{1,0}^{FCFS}$, (Property 2)
- $\bar{\tau}_{1,0}^{makespan} \leq \bar{t}_{1,0}^{makespan}$. (Property 3)

Property 1 and 2 are true since they are true on the original scale. Property 3 is slightly different from the corresponding characteristic on the original scale ($\tau_{1,0}^{makespan} < t_{1,0}^{makespan}$) since it is based on fluids rather than discrete jobs.

We divide further analysis of sub-problem 0 into two cases: either $\frac{1}{\mu_1} \geq \frac{1}{\mu_{(B+1)}}$ so that machine 1 is the bottleneck, or $\frac{1}{\mu_1} < \frac{1}{\mu_{(B+1)}}$ so that machine 2 is the bottleneck.

Case 1: Machine 1 Bottleneck Since $\bar{\tau}_{1,0}^{FCFS} = \frac{1}{\mu_1}$ and $\bar{t}_{1,0}^{FCFS} = \frac{1}{\mu_1}$, and by Properties 1 to 3, we know that

$$\frac{1}{\mu_1} = \bar{\tau}_{1,0}^{makespan} \leq \bar{t}_{1,0}^{makespan} \leq \bar{t}_{1,0}^{FCFS} = \frac{1}{\mu_1}. \tag{8.68}$$

Consequently, $\bar{\tau}_{1,0}^{makespan} = \bar{t}_{1,0}^{makespan} = \frac{1}{\mu_1}$. Therefore, $\bar{Q}_1^0(\frac{1}{\mu_1}) = 0$ and $\bar{Q}_{B+1}^0(\frac{1}{\mu_1}) = 0$ under *makespan*, just like under *FCFS*. (While under *FCFS* we can show that $\bar{Q}_{B+1}^0(t) \equiv 0$ for $0 \leq t \leq \frac{1}{\mu_1}$, problem instance 4 of Section 9.1.2 and Figure 9.6 provide an example of where this is not true for the *makespan* approach.)

Since $\bar{Q}_{(B+1)}^0(\bar{\tau}_{1,0}^{makespan}) = 0$, we know that at $\tau_{1,0}^{makespan}$, there is a finite number of jobs, γ, present at machine 2. For $\tau_{1,0}^{makespan} \leq t \leq t_{1,0}^{makespan}$, machine 2 operates as a non-idling single-server queue with a finite number of jobs present at $\tau_{1,0}^{makespan}$, and no new arrivals. Therefore, these γ jobs are going to be finished in a finite amount of time ($t_{1,0}^{makespan} - \tau_{1,0}^{makespan}$ is finite), which implies that, on the fluid scale, the next sub-problem is going to start immediately (e.g., $\bar{s}_{2,0}^{makespan} = \bar{t}_{1,0}^{makespan}$ assuming the system has at least 2 queues, and queue 2 is not empty at $\bar{t}_{1,0}^{makespan}$).

Case 2: Machine 2 Bottleneck By Property 1 and the fact that $\bar{\tau}_{1,0}^{FCFS} = \frac{1}{\mu_1}$, we know that machine 1 will complete sub-problem 1 of queue 1 at time $\frac{1}{\mu_1}$. $\bar{Q}_1^0(t)$ will be 0 for $\frac{1}{\mu_1} \leq t \leq \bar{t}_{1,0}^{makespan}$ since there are no arrivals to this sub-problem.

From Property 2 and the fact that $\bar{t}_{1,0}^{FCFS} = \frac{1}{\mu_{(B+1)}}$, it follows that

$$\bar{t}_{1,0}^{makespan} \leq \bar{t}_{1,0}^{FCFS} = \frac{1}{\mu_{(B+1)}}. \tag{8.69}$$

At the start of sub-problem 0 of queue 1, there is $\frac{1}{\mu_{B+1}}$ machine 2 work to be done during the sub-problem. Thus, the smallest amount of time in which this work can be completed is also $\frac{1}{\mu_{B+1}}$ (which can happen if and only if machine 2 is kept busy for that amount of time). As a consequence, we can strengthen the above to:

$$\frac{1}{\mu_{(B+1)}} \leq \bar{t}_{1,0}^{makespan} \leq \frac{1}{\mu_{(B+1)}}. \tag{8.70}$$

Thus, sub-problem 0 of queue 1 ends at $\frac{1}{\mu_{(B+1)}}$, i.e., at the same time as under *FCFS*.

Since $\bar{Q}_1(\bar{\tau}_{1,0}^{makespan}) = 0$, we know there is a time point $s_1^* < \tau_{1,0}^{makespan}$ when there is a finite number of jobs, γ_1, present at machine 1. For $s_1^* \leq t \leq t_{1,0}^{makespan}$, machine 1 operates as a non-idling single-server queue with a finite number of jobs present at s_1^*, and no new arrivals. Therefore, these γ_1 jobs are going to be finished in a finite amount of time ($\tau_{1,0}^{makespan} - s_1^*$ is finite). Since this is the case and since $\bar{Q}_{(B+1)}(\bar{t}_{1,0}^{makespan}) = 0$, we know that there is a time point $\tau_{1,0}^{makespan} \leq s_2^* < t_{1,0}^{makespan}$ when there is a finite number of jobs, γ_2, present at machine 2. For $s_2^* \leq t \leq t_{1,0}^{makespan}$, machine 2 operates as a non-idling single-server queue with a finite number of jobs present at s_2^*, and no new arrivals. Therefore, these γ_2 jobs are going to be finished in a finite amount of time ($t_{1,0}^{makespan} - s_2^*$ is finite), which implies that, on the fluid scale, the next sub-problem is going to start immediately (e.g., $\bar{s}_{2,0}^{makespan} = \bar{t}_{1,0}^{makespan}$ assuming the system has at least two queues, and queue 2 is not empty at $\bar{t}_{1,0}^{makespan}$).

For each subsequent sub-problem i, we can provide the same type of analysis by setting the initial fluid level appropriately (i.e., to reflect the arrivals since the start of the last visit to the same queue) and applying the same scaling as for sub-problem 0.

We have shown that on the fluid scale, given the same amount of work, the *FCFS* and the *makespan* policy complete a sub-problem in the same length of time. Since the initial amount of work present in the system would be the same regardless of whether *FCFS* or *makespan* is applied, it follows that on the fluid scale each sub-problem would have the same length under *makespan* and under *FCFS*. Therefore, *makespan* is stable under the same condition as *FCFS*, that is, when $\sum_{k=1}^{B} \frac{\lambda_k}{\min\{\mu_k, \mu_{B+k}\}} < 1$. □

The structure of the above proof implies that for any non-idling policy π, if we can show that the makespan of sub-problem 0 on the fluid scale is the same as for *FCFS*, then this policy can be shown stable under the same condition as *FCFS*.[8] A similar idea is used in Section 9.2.3 to determine stability conditions for a wide range of policies.

[8]Thank you to Maliheh Aramon Bajestani for this observation.

8.4 Generalizations

We now examine the stability of *FCFS* and *makespan* in generalizations of the polling system considered above.

8.4.1 Stability of *FCFS*

Theorem 8.3.1 can be generalized to the case in which the server is an M-machine flow shop or a d-stage flexible flow shop with M machines at each stage. Moreover, the stability results that hold for *FCFS* are also true for any non-idling policy that does not use processing time information. In the proofs below, we use the notation of the previous sections, unless otherwise specified.

8.4.1.1 Generalized M-machine Flow Shop Server

In an M-machine flow shop, each job requires processing on M machines in the same order. A polling system with an M-machine flow shop server is therefore a simple extension of our polling system in which the two-machine server is replaced by an M-machine server. A further extension is a polling system with a generalized M-machine flow shop server, in which each queue may utilize a different subset of the M machines (but all jobs within that queue go through the machines in the same order).

The set of classes belonging to queue b is denoted by $\mathcal{S}_b$. Each job in queue b consists of $|\mathcal{S}_b|$ activities and there is a one-to-one correspondence between activities and classes within a given queue. In total, there are $K = \sum_{b=1}^{B} |\mathcal{S}_b|$ classes. Each class k is served by exactly one of the M machines, so $|\mathcal{S}_b| \leq M$. Since the number of classes depends on the queue and each class is served by exactly one machine, the number of machines utilized by different queues may be different. We assume that there is at least one queue that utilizes all M machines.

Classes within each queue are numbered in the order in which they receive processing. Classes within queue 1 are numbered $1, 2, \ldots, |\mathcal{S}_1|$, classes within queue 2 are numbered $|\mathcal{S}_1|+1, |\mathcal{S}_1|+2, \ldots, |\mathcal{S}_1|+|\mathcal{S}_2|$, etc., so that classes within queue b, $2 \leq b \leq B$, are numbered $|\mathcal{S}_{b-1}|+1, |\mathcal{S}_{b-1}|+2, \ldots, |\mathcal{S}_{b-1}|+|\mathcal{S}_b|$. Routing is deterministic, so that upon completion, an activity of class k turns into an activity of class $l = k+1$ for $k, l \in \mathcal{S}_b$. A job leaves the system once its last activity is finished (i.e., for queue 1, this happens when the job exits class $|\mathcal{S}_1|$; for queue b, $2 \leq b \leq B$, this happens when the job exits class $|\mathcal{S}_{b-1}| + |\mathcal{S}_b|$). Thus, $\Phi_k^l(j) = 1$ for $k = l+1$ and $k, l \in \mathcal{S}_b$, and 0 otherwise.

It is possible for $\xi_k(j) \equiv 0$ for all j and some k, in which case the external arrival process to class k is null. $\mathcal{A}$ denotes the set of classes with non-null exogenous arrivals. We assume

that there is exactly one non-null exogenous arrival process to a queue. Thus, there are B non-null arrival processes in total, and $\mathcal{A} = \{1, |\mathcal{S}_1| + 1, |\mathcal{S}_2| + 1, \ldots, |\mathcal{S}_{B-1}| + 1\}$. The only assumptions we make regarding the inter-arrival and service times are assumptions (1.2)–(1.5) of Dai (1995), which are restated in Section 8.3.1.

As before, $s_{b,i}$ and $t_{b,i}$ denote the start time and completion time, respectively, of sub-problem i in queue b when *FCFS* is used to schedule jobs within the sub-problems. We denote by $\tau_{b,i}^m$ the completion time of sub-problem i in queue b on machine m. The stability condition for the polling system with a generalized M-machine flow shop server is given in the following theorem.

Theorem 8.4.1. *If* $\sum_{k\in\mathcal{A}} \frac{\lambda_k}{\min_{l\in\{\mathcal{S}_b|k\in\mathcal{S}_b\}}\{\mu_l\}} < 1$*, then the polling system with a generalized* M*-machine flow shop server is stable under* FCFS.

8.4.1.1.1 State Definition and System Dynamics The state for this system is defined as in Section 8.3.2.1, with the exception that now we let $\mathbb{Q}_m(t) = (v_m(t), N_m(t))$, be the state of machine m, where $v_m(t)$ is the residual processing time for machine m, and $N_m(t)$ is the total number of jobs in the *current* sub-problem of machine m, for $m = 1, \ldots, M$. The state space is $\mathcal{X} \subset \mathbb{Z}_+^{B+M+1} \times \mathbb{R}_+^{B+M}$.

The dynamics of the system are described using the notation of Section 8.3.2.2, with one small change that $E_k^x(t)$, the cumulative number of external arrivals to the system, is now defined for $k \in \mathcal{A}$. The overall dynamics of the system are then:

$$I_m^x(t) = t - \sum_{k\in\mathcal{C}_m} T_k^x(t) \text{ is non-decreasing in } t, \tag{8.71}$$

$$m = 1, 2, \ldots, M \tag{8.72}$$

$$\dot{T}_k^x(t) = 1 \text{ iff } s_{b,i} \le t \le \tau_{b,i}^m \text{ for some } i,\ s(k) = m,\ k \in \mathcal{A} \tag{8.73}$$

$$\int_0^\infty (\sum_{k\in\mathcal{C}_m} Q_k^x(t)) dI_i^x(t) = 0, \quad \text{for } m = 1, 2, \ldots, M,\ k \notin \mathcal{A} \tag{8.74}$$

$$\text{Equations } (8.4) - (8.7) \tag{8.75}$$

We cannot derive a fluid limit model that would represent the overall system dynamics directly from the system dynamics equations above because we cannot apply the Strong Law of Large Numbers to derive the appropriate limits for all quantities. Specifically, there is a dependence between the departure process of machine 1 and the processing times on the other machines. For example, consider sub-problem i in which machine m is the last machine utilized, and the subsequent sub-problem $i + 1$ in which machine 1 is the first machine: the processing times on machine m in sub-problem i influence when the next sub-problem will start on machine 1

and therefore the number of jobs that will be processed in sub-problem $i+1$ on both machine 1 and machine m.

8.4.1.1.2 Proof The formal proof of stability is very similar to the proof of Theorem 8.3.1, and is given below. We omit the first part of the proof (the statement of the system dynamics and the fluid limit derivations) due to this similarity.

Proof. As in the proof of Theorem 8.3.1, the fluid dynamics equations for one sub-problem are:

$$\bar{Q}_1^0(t) = 1 - \mu_1 \bar{T}_1^0(t) \tag{8.76}$$

$$\bar{Q}_k^0(t) = \mu_{k-1}\bar{T}_{k-1}^0(t) - \mu_k \bar{T}_k^0(t),\ k = 2, \ldots, |\mathcal{S}_1| \tag{8.77}$$

$$\bar{Q}^0(t) = (\bar{Q}_1^0(t), \ldots, \bar{Q}_{|\mathcal{S}_1|}^0(t))^T \geq 0 \tag{8.78}$$

$$\bar{T}^0(\cdot) = (\bar{T}_1^0(\cdot), \ldots, \bar{T}_{|\mathcal{S}_1|}^0(\cdot)) \text{ is non-decreasing} \tag{8.79}$$

$$\bar{T}_1^0(\bar{s}_{1,0}) = 0, \quad \ldots \quad , \bar{T}_{|\mathcal{S}_1|}^0(\bar{s}_{1,0}) = 0 \tag{8.80}$$

$$\bar{I}_1^0(t) = t - \bar{T}_1^0(t) \text{ is non-decreasing} \tag{8.81}$$

$$\bar{I}_k^0(t) = t - \bar{T}_k^0(t) \text{ is non-decreasing, } k = 2, \ldots, |\mathcal{S}_1| \tag{8.82}$$

$$\int_{\bar{s}_{1,0}}^{\bar{\tau}_{1,0}^{s(k)}} \bar{Q}_k^0(t) d\bar{I}_{s(k)}^0(t) = 0,\ k = 1, \ldots, |\mathcal{S}_1| - 1 \tag{8.83}$$

$$\int_{\bar{s}_{1,0}}^{\bar{t}_{1,0}} \bar{Q}_{|\mathcal{S}_1|}^0(t) d\bar{I}_{s(|\mathcal{S}_1|)}^0(t) = 0 \tag{8.84}$$

We can now analyze the behaviour of this sub-problem on the fluid scale. Let the class served by the bottleneck machine be $\hat{k}$. The amount of time it takes to drain all class l fluid of this sub-problem depends on the position of l in the route of jobs of the current queue:

- *Class $l = 1$:* The length of the sub-problem on machine 1 (class 1) is always $\frac{1}{\mu_1}$ since the policy is non-idling and the fluid level is 1 at the beginning of the sub-problem.
- *Class $l \in \mathcal{S}_1$ such that $2 \leq l < \hat{k}$:* There are two[9] cases:
  - If $\mu_{l-1} \geq \mu_l$, then work is arriving to machine $s(l)$ at rate $\mu_{l-1}/\mu_l \geq 1$. Since *FCFS* is non-idling, this implies that the length of the sub-problem for class l is $\frac{1}{\mu_l}$.
  - If $\mu_{l-1} < \mu_l$, then work is arriving to machine $s(l)$ at rate $\mu_{l-1}/\mu_l < 1$. This implies[10] that

[9] The relation between any pair of consecutive machines is the same as between machines 1 and 2 in the proof for the two-machine case (Section 8.3). That is, if $\mu_{l-1} \leq \mu_l$, machine $l-1$ is the bottleneck for this pair of machines, and vice versa, if $\mu_{l-1} > \mu_l$, then machine l is the bottleneck for this pair.

[10] More details for why this is true can be found in the two-machine proof of Section 8.3.

(a) $\bar{Q}_l(t) \equiv 0$ for $s_{1,0} \leq t \leq t_{1,0}$, and
(b) the sub-problem will complete on machine $s(l)$ at the same time as on machine $s(l-1)$.

- *Bottleneck class $l = \hat{k}$, $\hat{k} \geq 2$:* The rate at which work arrives at class $\hat{k}$ is $\mu_{\hat{\hat{k}}}/\mu_{\hat{k}}$, where $\hat{\hat{k}}$ is the class with the greatest mean processing time from $\{1, \ldots, \hat{k}-1\}$ (i.e., the "bottleneck" among all the classes preceeding the actual bottleneck $\hat{k}$). Since $\mu_{\hat{k}} \leq \mu_k$ for all $k \in \mathcal{S}_1$, the rate at which work arrives at the bottleneck class $\hat{k}$ is greater than or equal to 1. Since the machine is non-idling and the rate at which work arrives is greater than 1, the length of the sub-problem on the bottleneck machine $s(\hat{k})$ is $\frac{1}{\mu_{\hat{k}}}$.

- *Class $l \in \mathcal{S}_1$ such that $\hat{k} < l \leq |\mathcal{S}_1|$:* Since $\mu_{\hat{k}} \leq \mu_k$ for all $k \in \mathcal{S}_1$, the rate at which work arrives at class l is $\mu_{\hat{k}}/\mu_l < 1$. It can therefore be shown that
(a) $\bar{Q}_l(t) \equiv 0$ for $s_{1,0} \leq t \leq t_{1,0}$, and
(b) the sub-problem will complete on machine $s(l)$ at the same time as on the bottleneck machine.

Hence, $t_{1,0} = (\mu_{\hat{k}})^{-1}$. By using the same argument as in the proof of Theorem 8.3.1, we find that, on the fluid scale, the next sub-problem is going to start immediately after the end of the current sub-problem. For each subsequent sub-problem i, a fluid model can be constructed similarly by assuming that we set the initial number of jobs to $Q_k^x(s_{b,i})$ and applying the same scaling as for sub-problem 0.

We now need to show that there exists a time t^* at which the fluid level in all queues becomes 0 and stays at 0. We do so by using a function that represents the total bottleneck workload at time t and by considering the dynamics of the overall system, not just each sub-problem. We know that $E_k^x(t)$ is the number of arrivals by time t to class k and, if we apply the same scaling as used above in the sub-problem, we obtain $\bar{E}_k^x(t) = \frac{1}{|x|}E_k^x(|x|t)$ and $\lim_{|x|\to\infty} \frac{1}{|x|}E_k^x(|x|t) = \lambda_k t$.

Let $\beta = \{1, 2, \ldots, B\}$, the set of all queues in the system. We can then define the total bottleneck workload at time t as

$$\bar{z}(t) = \sum_{b\in\beta} \sum_{k\in\mathcal{S}_b} \frac{\bar{Q}_k(t)}{\min_{\{l\in\mathcal{S}_b\}} \mu_l}. \tag{8.85}$$

If the derivative of $\bar{z}(t)$ exists, then it is defined as

$$\dot{\bar{z}}(t) = \sum_{b\in\beta} \sum_{k\in\mathcal{S}_b} \frac{\dot{\bar{Q}}_k(t)}{\min_{\{l\in\mathcal{S}_b\}} \mu_l}. \tag{8.86}$$

Suppose $\bar{Q}_k(t) > 0$ for at least one k. Then it must be that the server is at one of these queues, say queue b^*. If the server is at queue b^* at time t, then for all queues $b \neq b^*$, there can be no fluid leaving the queue at t, but there is arriving fluid, so that $\dot{\bar{Q}}_k(t) = \lambda_k$, $k \in \mathcal{S}_b \cap \mathcal{A}$, and $\dot{\bar{Q}}_l(t) = 0$ for classes $l \in \mathcal{S}_b$. In other words,

$$\dot{\bar{z}}(t) = \sum_{k \in S_{b^*}} \frac{\dot{\bar{Q}}_k(t)}{\min_{\{l \in S_b\}} \mu_l} + \sum_{b \in \beta \backslash b^*} \sum_{k \in S_b} \frac{\dot{\bar{Q}}_k(t)}{\min_{\{l \in S_b\}} \mu_l} \tag{8.87}$$

$$= \sum_{k \in S_{b^*}} \frac{\dot{\bar{Q}}_k(t)}{\min_{\{l \in S_b\}} \mu_l} + \sum_{b \in \beta \backslash b^*} \sum_{k \in S_b \cap \mathcal{A}} \frac{\lambda_k}{\min_{\{l \in S_b\}} \mu_l}. \tag{8.88}$$

It is easiest to divide the analysis into two cases: when the bottleneck class $\hat{k}$ in queue b^* has a non-null exogenous arrival process (i.e., $\hat{k} \in \mathcal{A}$), and when it does not.

Case 1: If class $\hat{k} \in \mathcal{A}$, then $\dot{\bar{T}}^i_{\hat{k}}(t) = \dot{\bar{T}}_{\hat{k}}(t) = 1$ for $t \in [\bar{s}_{b^*,i}, \bar{t}_{b^*,i})$ (since in this case the length of the sub-problem on the fluid scale is exactly the length of time necessary to process all the work in the bottleneck class). In this case,

$$\dot{\bar{Q}}_{\hat{k}}(t) = \lambda_{\hat{k}} - \mu_{\hat{k}} \dot{\bar{T}}_{\hat{k}}(t) \tag{8.89}$$

$$= \lambda_{\hat{k}} - \mu_{\hat{k}}, \tag{8.90}$$

and $\dot{\bar{Q}}_k(t) = 0$ for $k \in \mathcal{S}_b \backslash \hat{k}$ (see analysis above). Therefore,

$$\dot{\bar{z}}(t) = \sum_{k \in S_{b^*}} \frac{\dot{\bar{Q}}_k(t)}{\min_{\{l \in S_b\}} \mu_l} + \sum_{b \in \beta \backslash b^*} \sum_{k \in S_b \cap \mathcal{A}} \frac{\lambda_k}{\min_{\{l \in S_b\}} \mu_l} \tag{8.91}$$

$$= \frac{\lambda_{\hat{k}} - \mu_{\hat{k}}}{\mu_{\hat{k}}} + \sum_{b \in \beta \backslash b^*} \sum_{k \in S_b \cap \mathcal{A}} \frac{\lambda_k}{\min_{\{l \in S_b\}} \mu_l} \tag{8.92}$$

$$= \frac{\lambda_{\hat{k}} - \mu_{\hat{k}}}{\mu_{\hat{k}}} + \sum_{k \in \mathcal{A} \backslash \hat{k}} \frac{\lambda_k}{\min_{l \in \{S_b | k \in S_b\}} \mu_l} \tag{8.93}$$

$$= \frac{\lambda_{\hat{k}}}{\mu_{\hat{k}}} - 1 + \sum_{k \in \mathcal{A} \backslash \hat{k}} \frac{\lambda_k}{\min_{l \in \{S_b | k \in S_b\}} \mu_l} \tag{8.94}$$

$$< 0, \text{ since } \sum_{k \in \mathcal{A}} \frac{\lambda_k}{\min_{l \in \{S_b | k \in S_b\}} \mu_l} < 1. \tag{8.95}$$

Case 2: If $\hat{k} \notin \mathcal{A}$, then the values of $\dot{\bar{Q}}_k(t)$ are 0 for all $k > \hat{k}$, but may be non-zero for

$k < \hat{k}$.[11] In this case,

$$\sum_{k \in S_{b^*}} \frac{\dot{\bar{Q}}_k(t)}{\min_{\{l \in S_b\}} \mu_l} = \sum_{k \leq \hat{k}} \frac{\dot{\bar{Q}}_k(t)}{\min_{\{l \in S_b\}} \mu_l}. \tag{8.96}$$

$\dot{\bar{Q}}_k(t)$ is defined as follows:

- If k is the initial class of the queue, $k \in \mathcal{A}$, then $\dot{\bar{Q}}_k(t) = \lambda_k - \mu_k$ for $t \in [\bar{s}_{b^*,i}, \bar{\tau}^{s(k)}_{b^*,i})$, and $\dot{\bar{Q}}_k(t) = \lambda_k$ for $t \in [\bar{\tau}^{s(k)}_{b^*,i}, \bar{t}_{b^*,i})$.

- If $k = \hat{k}$, $\dot{\bar{Q}}_k(t) = \mu_{k-1} - \mu_k$ for $t \in [\bar{s}_{b^*,i}, \bar{\tau}^{s(k)}_{b^*,i})$, and $\dot{\bar{Q}}_k(t) = -\mu_k$ for $t \in [\bar{\tau}^{s(k)}_{b^*,i}, \bar{t}_{b^*,i})$.

- If k is not the initial class and not the bottleneck class, then there are two possibilities.

  If $\mu_{k-1} \leq \mu_k$, then $\dot{\bar{Q}}_k(t) = 0$ for all $t \in [\bar{s}_{b^*,i}, \bar{t}_{b^*,i})$ and the (internal) arrival rate to the next class $l > k$ s.t. $\dot{\bar{Q}}_l(t) \neq 0$ is equal to the processing rate in the previous class $j < k$ for which $\dot{\bar{Q}}_j(t) \neq 0$. If $\mu_{k-1} > \mu_k$, then $\dot{\bar{Q}}_k(t) = \mu_{k-1} - \mu_k$ for all $t \in [\bar{s}_{b^*,i}, \bar{\tau}^{s(k)}_{b^*,i})$ and $\dot{\bar{Q}}_k(t) = -\mu_k$ for all $t \in [\bar{\tau}^{s(k)}_{b^*,i}, \bar{t}_{b^*,i})$.

Note that, due to the flow shop structure and the fact that classes are numbered according to the order in which they receive processing, $\bar{\tau}^{s(k)}_{b^*,i} \leq \bar{\tau}^{s(l)}_{b^*,i}$ for $k < l$.

Therefore, for any $t \in [\bar{s}_{b^*,i}, \bar{t}_{b^*,i})$ (any time point within sub-problem i at queue b^*), $\sum_{k \leq \hat{k}} \dot{\bar{Q}}_k(t) = \lambda_\kappa - \mu_{\hat{k}}$, where κ denotes the class of queue b^* with a non-zero exogenous arrival process. Hence,

$$\begin{aligned}
\dot{\bar{z}}(t) &= \sum_{k \in S_{b^*}} \frac{\dot{\bar{Q}}_k(t)}{\min_{\{l \in S_b\}} \mu_l} + \sum_{b \in \beta \setminus b^*} \sum_{k \in S_b \cap \mathcal{A}} \frac{\lambda_k}{\min_{\{l \in S_b\}} \mu_l} && (8.97)\\
&= \frac{\lambda_\kappa - \mu_{\hat{k}}}{\mu_{\hat{k}}} + \sum_{k \in \mathcal{A} \setminus \hat{k}} \frac{\lambda_k}{\min_{l \in \{S_b | k \in S_b\}} \mu_l} && (8.98)\\
&= \frac{\lambda_\kappa}{\mu_{\hat{k}}} - 1 + \sum_{k \in \mathcal{A} \setminus \hat{k}} \frac{\lambda_k}{\min_{l \in \{S_b | k \in S_b\}} \mu_l} && (8.99)\\
&< 0, \text{ since } \sum_{k \in \mathcal{A}} \frac{\lambda_k}{\min_{l \in \{S_b | k \in S_b\}} \mu_l} < 1. && (8.100)
\end{aligned}$$

For example, consider the case with 5 machines in queue 1 and $\mu_1 \geq \mu_2$, $\mu_2 < \mu_3$, $\mu_3 \geq \mu_4$ and $\mu_5 = \max\{\mu_1, \mu_2, \mu_3, \mu_4, \mu_5\}$. In this case, for $t \in [\bar{s}_{1,i}, \bar{\tau}^{s(1)}_{1,i})$, $\dot{\bar{Q}}_1(t) = \lambda_1 - \mu_1$, $\dot{\bar{Q}}_2(t) = \mu_1 - \mu_2$, $\dot{\bar{Q}}_3(t) = 0$, $\dot{\bar{Q}}_4(t) = \mu_3 - \mu_4 = \mu_2 - \mu_4$ and $\dot{\bar{Q}}_5(t) = \mu_4 - \mu_5$, so that the

[11] The fact that $\dot{\bar{Q}}_k(t) = 0$ follows from the argument described in the proof for the two-machine system in Section 8.3.

numerator of the first sum in Equation (8.97) becomes $\lambda_1 - \mu_5$.

We have shown that if $\sum_{k\in\mathcal{A}} \frac{\lambda_k}{\min_{l\in\{S_b|k\in S_b\}}\mu_l} < 1$ then in both cases 1 and 2, $\dot{\bar{z}}(t) < 0$. Therefore, $\dot{\bar{z}}(t) < 0$ whenever there exists at least one k such that $\bar{Q}_k(t) > 0$. This is equivalent stating that $\dot{\bar{z}}(t) < 0$ whenever $z(t) > 0$. It follows that $\bar{z}(t) \equiv 0$ for $t \geq \frac{\bar{z}(0)}{1-\sum_{k\in\mathcal{A}} \frac{\lambda_k}{\min_{l\in\{S_b|k\in S_b\}}\mu_l}}$ by Lemma 5.2 of Dai (1995).

Consequently, there is a time point $t^* = \frac{\bar{z}(0)}{1-\sum_{k\in\mathcal{A}} \frac{\lambda_k}{\min_{l\in\{S_b|k\in S_b\}}\mu_l}}$ at which the total bottleneck workload in the system reaches 0 and after which it remains at 0. This implies that if $\sum_{k\in\mathcal{A}} \frac{\lambda_k}{\min_{l\in\{S_b|k\in S_b\}}\mu_l} < 1$, then the system is stable. □

8.4.1.2 Flexible Flow Shop Server

A flexible flow shop is a generalization of the flow shop and parallel machine environments: there are d stages in series, and, at every stage, there are M parallel identical machines. Every job needs to be processed at stage 1, stage 2, etc. as in a flow shop. At every stage, the job requires exactly one machine and since the machines are identical, any of them can be employed (Pinedo, 2003).[12] Thus, another extension of the polling system discussed in Section 8.3 is a polling system with a flexible flow shop server.

In a flexible flow shop, each stage can be viewed as a $G/G/M$ queue. Each stage corresponds to a class. If the processing rate of every machine at stage s is μ_s and there are n jobs present at this stage, then the overall processing rate of the corresponding class k is $\mu_k = n\mu_s$ if $1 \leq n \leq M$ and $\mu_k = M\mu_s$ if $n > M$. The stability condition for this system is the same as for the system with a generalized M-machine flow shop server:

Theorem 8.4.2. *If $\sum_{k\in\mathcal{A}} \frac{\lambda_k}{\min_{l\in\{S_b|k\in S_b\}}\{\mu_l\}} < 1$, then the polling system with a flexible d-stage flow shop server with M machines at each stage is stable under* FCFS.

The proof of this theorem is essentially identical to the one used for the polling system with a generalized M-machine server, except that the machines in that proof have to now be treated as stages. The main observation that allows the proof to be easily extended is that, to derive the fluid limit, we take the initial number of jobs in the system to infinity, implying that on the fluid scale, the processing rate at stage s is $M\mu_s$. Thus, to ensure the validity of the proof, we need to set $\mu_k = M\mu_s$.

[12] Such a flexible flow shop with M machines and processing rate μ_s for every machine of stage s is not equivalent to the flow shop that has a single machine at every stage with processing rate $M\mu_s$. For example, if there is exactly one job at stage s and $M \geq 2$, the expected processing time of this job will be $\frac{1}{\mu_s}$ in the flexible flow shop model, but $\frac{1}{M\mu_s}$ in the single-machine flow shop.

If we relax the assumption that the number of machines at every stage is the same, then, on the original scale, there is some ambiguity regarding the definition of a bottleneck machine/stage. For instance, suppose we have a system with two stages. At stage 1, there is one machine with a processing rate of 5. At stage 2, there are two machines, each with a processing rate of 3. Thus, if we define a bottleneck stage as a stage which has the smallest processing rate per machine, then in our example, the bottleneck is stage 2. Alternatively, we can define the bottleneck to be the stage with the smallest *overall* processing rate; this definition would imply stage 1 is the bottleneck in our example. On the fluid scale, this ambiguity disappears since it is assumed that a very large number of jobs is initially present in the system and the second of our two definitions has to be employed. Given this definition, we see that the stability condition and its proof become identical to those of the case with the same number of machines at each stage.

Finally, if we change the assumption of parallel machines to *cooperating* machines,[13] as in the paper by Andradóttir et al. (2003), then, even on the original scale, every stage s with M machines is equivalent to a server with processing rate $M\mu_s$. Therefore, Theorem 8.4.2 also holds, with μ_l set to the appropriate $M\mu_s$, regardless of whether the number of machines varies between stages.

8.4.2 Stability of *makespan*

As for *FCFS*, generalizing the proof of stability to the case where the server is an M-machine flow shop with $M > 2$ or a d-stage flexible flow shop is straightforward. This may appear surprising: in neither of these environments does the optimal makespan possess the structure provided by Johnson's rule for the two-machine case; finding the optimal is, in fact, NP-hard (Pinedo, 2003). However, the proof does not explicitly utilize the structure within the schedule, only the fact that the schedule is guaranteed to have the minimum makespan. Therefore, for these environments, the difficulty of obtaining the minimum makespan has no effect on the stability proof.

8.5 Conclusion

In this chapter we analyzed the stability of two scheduling methods in the systems presented in the previous chapter. Specifically, we proved stability of the *makespan* method in a dynamic flow shop using a sample path argument. We showed stability of *FCFS* and *makespan* in a polling system with a two-machine flow shop server. These proofs of stability extend to the

[13] It is probably more reasonable to think of cooperating *servers* rather than *machines*.

case when the server is a flow shop with M machines or a flexible flow shop with d stages and M machines at each stage. Both proofs employed the fluid model methodology of Dai (1995). Thus, we introduced long-run stability into combinatorial scheduling and demonstrated that stability of a method from the traditional scheduling literature that uses exact processing time information can be formulated and proved. We therefore demonstrated that theoretical long-run performance guarantees can be obtained for periodic scheduling methods.

This chapter creates a connection between work in dynamic scheduling, which has not considered stability, and work in queueing theory, which has developed formal methodologies for proving stability of particular scheduling disciplines, thereby integrating queueing theory and scheduling on a theoretical level. In the next chapter, we show that additional insights can be gained from such integration. Specifically, we analyze the fluid limits associated with different scheduling policies and further examine the stability of non-idling policies in the polling system with a two-machine flow shop server.

Chapter 9

Theoretical Integration of Scheduling and Queueing Theory: Fluid Analysis

In the previous chapter, we showed that fluid model methodology can be used to prove stability of periodic scheduling methods in both the dynamic flow shop and the polling system with a flow shop server. In the current chapter, we continue our investigation of the theoretical level of integration by analyzing, for different policies, the fluid limits of work arriving to machine 2 and of work present at machine 2 within one (static) sub-problem. This analysis leads to:

- identification of a key feature of the scheduling algorithm that minimizes makespan;
- observations regarding the use of fluid limits for predicting algorithm performance;
- understanding of stability for a variety of periodic scheduling approaches.

To our knowledge, the only paper that applies fluid model methodology to static scheduling problems in which all jobs may be distinct is by Nazarathy and Weiss (2010). They use the fluid representation of the problem to develop an asymptotically-optimal scheduling heuristic. Our work, on the contrary, proposes fluid analysis of known scheduling algorithms as a tool for gaining a better understanding of how these algorithms perform both with respect to a classical scheduling objective in static problems and with respect to stability in dynamic problems.

This chapter consists of two main sections. In Section 9.1, we provide an analysis of fluid limits. The majority of our evaluation is empirical, although some theoretical results are presented. In Section 9.2, we describe the insights gained from our analysis. Section 9.3 concludes the chapter.

9.1 Analysis of Fluid Limits

We focus on the behaviour of fluid limits in one of the sub-problems encountered by a periodic scheduling approach in a dynamic two-machine flow shop or in a polling system with a two-machine flow shop server.[1] We investigate the fluid limits of work arriving to machine 2 and of work present at machine 2 for seven policies:

- *FCFS*, which processes jobs in non-decreasing order of their arrival times to the queue.
- SPT_{sum}, which processes jobs in non-decreasing order of the *sum* of their durations on machine 1 and machine 2.
- *makespan*, which minimizes the makespan of each sub-problem by employing Johnson's rule. This rule divides jobs into two sets: set I consists of all jobs whose processing time on machine 1 is less than or equal to its processing time on machine 2, and set II consists of all the remaining jobs. Set I is processed before set II. Johnson's rule creates permutation schedules, that is, schedules in which the order of jobs is the same on both machines. Within set I, jobs are sequenced in non-decreasing order of the processing times on machine 1, while within set II, jobs are sequenced in non-increasing order of the processing times on machine 2.
- *J|FCFS*, which denotes *Johnson's FCFS*, operates like Johnson's rule in that it divides jobs in two sets based on processing times and schedules set I before set II, but processes jobs within each set in *FCFS* order.
- *reverse*, which schedules set II of Johnson's rule before set I. Within each set, jobs are processed in the same order as under Johnson's rule.
- *reverse|FCFS* is the same as *reverse* in that it schedules set II before set I. However, it sequences jobs in *FCFS* order within each set.
- *reverseMakespan*, which maximizes the makespan by sequencing jobs in the order that is opposite from Johnson's rule (Kim, 1993). This means that set II is scheduled before set I; within set II, jobs are processed in shortest-processing time order on machine 2, and within set I, jobs are processed in longest-processing time order on machine 1.

We assume that all of these policies construct permutation schedules, that is, schedules in which jobs are sequenced in the same order on machine 1 and 2. Due to their complexity, we leave the investigation of the fluid limits of the *completionTime* model, which minimizes the sum of

[1] Analysis of one sub-problem corresponds to analysis of a static two-machine flow shop problem.

completion times of a given set of jobs, for future work. Given that SPT_{sum} is asymptotically optimal for the average completion time objective in a static two-machine flow shop as the number of jobs increases (Xia et al., 2000), it would be interesting to determine whether the fluid limits of SPT_{sum} are similar to those of *completionTime*.

9.1.1 Formal Definitions

We start by formally defining the fluid limits of work arriving to machine 2 and of work present at machine 2 within one sub-problem. Assuming the state definition of Section 8.3.2.1, we denote the state at the start of the sub-problem by x. This state is characterized by y_0 jobs at machine 1, with all the other components, including the number of jobs at machine 2, being 0. As before, the mean processing time on machine m is denoted $E[\eta_m(1)] = \frac{1}{\mu_m}$.

$D_m^{x,\pi}(t)$ denotes the cumulative number of departures from machine m by time t under policy π given initial state x. This quantity is dependent on the scheduling policy since the scheduling policy defines the order in which jobs leave machine 1.

Let $\mathcal{E}_2^{x,\pi}(t)$ be a function that represents the cumulative workload arriving to machine 2 from machine 1 in $[0,t]$ under policy π given initial state x:

$$\mathcal{E}_2^{x,\pi}(t) = \sum_{h=1}^{D_1^{x,\pi}(t)} \eta_2([h]), \tag{9.1}$$

where $[h]$ represents the hth position in the schedule constructed by π. Let $\mathcal{W}_2^{x,\pi}(t)$ represent the work present at machine 2 at time t:

$$\mathcal{W}_2^{x,\pi}(t) = \mathcal{E}_2^{x,\pi}(t) - \sum_{h=1}^{D_2^{x,\pi}(t)} \eta_2([h]). \tag{9.2}$$

We are interested in evaluating the undelayed[2] fluid limits of the processes defined in Equations (9.1) and (9.2). Thus, we take the limit as $|x|$ goes to ∞ by letting $y_0 \to \infty$ while keeping the remaining components of x constant:

$$\bar{\mathcal{E}}_2^{\pi}(t) = \lim_{|x|\to\infty} \frac{1}{|x|}\mathcal{E}_2^{x,\pi}(|x|t), \tag{9.3}$$

$$\bar{\mathcal{W}}_2^{\pi}(t) = \lim_{|x|\to\infty} \frac{1}{|x|}\mathcal{W}_2^{x,\pi}(|x|t). \tag{9.4}$$

The fluid limits under *FCFS* are dependent only on the means of the processing time distribu-

[2]See the work of Dai (1995), Dai and Meyn (1995) for formal definitions of delayed and undelayed fluid limits.

tions:

$$\bar{\mathcal{E}}_2^{\pi}(t) = \frac{\mu_1}{\mu_2}t, \tag{9.5}$$

$$\bar{\mathcal{W}}_2^{\pi}(t) = \frac{\mu_1}{\mu_2}t - 1. \tag{9.6}$$

The rest of the policies use detailed processing time information, making it difficult to evaluate their fluid limits analytically. We therefore investigate their behaviour empirically below.

9.1.2 Empirical Evaluation

Our empirical evaluation is based on three categories of problem instances:

- **Category 1:** the processing time of job j on machine 1 is smaller than or equal to its processing time on machine 2, $\eta_1(j) \leq \eta_2(j)$, for all j;
- **Category 2:** the processing time of job j on machine 1 is greater than its processing time on machine 2, $\eta_1(j) > \eta_2(j)$, for all j;
- **Category 3:** there is at least one job j such that $\eta_1(j) \leq \eta_2(j)$ and at least one job l such that $\eta_1(l) > \eta_2(l)$.

Category 1 instances consist only of jobs that belong to set I of Johnson's rule, and category 2 instances consist only of jobs of set II. Category 3 instances contain both set I and II jobs. Throughout this chapter, we refer to a job j that satisfies the property $\eta_1(j) \leq \eta_2(j)$ as a set I job and to a job j that satisfies $\eta_1(j) > \eta_2(j)$ as a set II job.

We evaluate six instances in total: two from category 1, two from category 2, and two from category 3. In instances 0 to 3, which are listed in Table 9.1, there are two job types, A and B, each occurring with equal probability and having deterministic processing times. Since all A jobs are identical and all B jobs are identical, we denote their processing times by $\eta_m(A)$ and $\eta_m(B)$, respectively, for machine m. Instance 4 is based on a two-point distribution: $P(\eta_1(j) = 1) = 0.5$, $P(\eta_1(j) = 3) = 0.5$, $P(\eta_2(j) = 1) = 0.5$ and $P(\eta_2(j) = 2) = 0.5$. Thus, four job types are possible: A with processing times $(\eta_1(A), \eta_2(A)) = (1, 1)$, B with processing times $(1, 2)$, C with processing times $(3, 1)$, and D with processing times $(3, 2)$. Type A and B jobs are part of set I of Johnson's rule, while C and D belong to set II. Instance 5 has exponentially distributed processing times and was empirically evaluated in Chapter 7.

The instances of each category can be distinguished based on their *consistency*: if placing the jobs in non-decreasing order of the processing times on machine 1 also results in the jobs being ordered in non-decreasing order of the processing times on machine 2, i.e., if $\eta_1(j) \leq$

$\eta_1(l)$ implies $\eta_2(j) \leq \eta_2(l)\ \forall\ j, l$, then we call the corresponding problem instance *consistent*. As shown in Table 9.1, instances 0 and 2 are consistent; the other instances we consider are inconsistent.

Instance	$(\eta_1(A), \eta_2(A))$	$(\eta_1(B), \eta_2(B))$	type	μ_1	μ_2	consistent?
0	$(1, 2)$	$(2, 300)$	1	$\frac{2}{3}$	$\frac{1}{151}$	yes
1	$(1, 300)$	$(2, 2)$	1	$\frac{2}{3}$	$\frac{1}{151}$	no
2	$(300, 2)$	$(2, 1)$	2	$\frac{1}{151}$	$\frac{2}{3}$	yes
3	$(300, 1)$	$(3, 2)$	2	$\frac{2}{303}$	$\frac{2}{3}$	no

Table 9.1: Data for instances 0 to 3.

In our evaluation of the fluid limits, we assume that one unit of fluid is present at machine 1 at time 0, with equal proportions of fluid belonging to each of the fluid types that are part of the instance.

9.1.2.1 Instances of Categories 1 and 2

Instances 0 to 3 consist of jobs belonging to only one set of Johnson's rule, so the *reverse* policy is equivalent to the *makespan* policy, while *J|FCFS* and *reverse|FCFS* are both equivalent to *FCFS*. The fluid limits of work arriving to machine 2 are piecewise linear functions for all policies since all A jobs are identical and all B jobs are identical. The rate at which work arrives at machine 2 is the ratio of the machine 2 and machine 1 processing rates.

Consider instance 0 as an example. The rate at which work arrives at machine 2 is 2 for type A fluid and 150 for type B. Using SPT_{sum} and *makespan* yields the same schedule: all jobs of type A are scheduled before any jobs of type B. The amount of time it takes for machine 1 to process all fluid of this sub-problem is $0.5(1) + 0.5(2) = 1.5$ since the initial fluid level of each type is 0.5 and since the machine is non-idling. Once all work of the current sub-problem has arrived at machine 2, the function $\bar{\mathcal{E}}_2^{\pi}(t)$ remains constant until the end of the sub-problem, which occurs at 151 for instance 0. Assuming the fluid level at machine 2 is initially 0, Equation (9.3) for $\pi =$*makespan* and $\pi = SPT_{sum}$ is the piecewise linear function:

$$\bar{\mathcal{E}}_2^{\pi}(t) = \begin{cases} 2t, & 0 \leq t \leq 0.5, \\ 150t - 74, & 0.5 \leq t \leq 1.5, \\ 151, & 1.5 \leq t \leq 151. \end{cases} \tag{9.7}$$

The *reverseMakespan* policy constructs the opposite sequence, resulting in the function:

$$\bar{\mathcal{E}}_2^{reverseMakespan}(t) = \begin{cases} 150t, & 0 \le t \le 1, \\ 2t + 148, & 1 \le t \le 1.5, \\ 151, & 1.5 \le t \le 151. \end{cases} \tag{9.8}$$

The fluid limits for instances 1 to 3 are calculated similarly. The graphs of the functions corresponding to instances 0 to 3 are presented in Figures 9.1–9.4.

The fluid limits of work present at machine 2 are calculated based on Equations (9.2) and (9.4). For instances of category 1, the arrival rate of work to machine 2 is always greater than or equal to 1, implying that machine 2 is always busy and that the rate at which work leaves machine 2 is 1. As a result, for instances 0 and 1, $\bar{\mathcal{W}}_2^{reverseMakespan}(t) = \bar{\mathcal{E}}_2^{\pi}(t) - t$. For instance 0, the total amount of machine 2 work is $0.5(2) + 0.5(300) = 151$. Since machine 2 is always fully utilized, the sub-problem finishes at time 151 for all policies that we consider. The fluid limit of work present at machine 2 for instance 0 is the following piecewise linear function for $\pi =$ *makespan* and $\pi = SPT_{sum}$:

$$\bar{\mathcal{W}}_2^{\pi}(t) = \begin{cases} t, & 0 \le t \le 0.5, \\ 149t - 74, & 0.5 \le t \le 1.5, \\ 151 - t, & 1.5 \le t \le 151. \end{cases} \tag{9.9}$$

For *reverseMakespan*, the fluid limit is defined by the function:

$$\bar{\mathcal{W}}_2^{reverseMakespan}(t) = \begin{cases} 149t, & 0 \le t \le 1, \\ t + 148, & 1 \le t \le 1.5, \\ 151 - t & 1.5 \le t \le 151. \end{cases} \tag{9.10}$$

For type 2 instances the rate at which work arrives at machine 2 is always less than 1, implying that work leaves machine 2 instantaneously as it arrives. We do not present the graphs of the functions corresponding to the work present at machine 2 for instances 0 to 3 since for type 1 instances the slope of the $\bar{\mathcal{W}}_2^{\pi}(t)$ functions is different from the slope of $\bar{\mathcal{E}}_2^{\pi}(t)$ by -1, while for type 2 instances the fluid level of work present at machine 2 is always 0.

9.1.2.2 Instances of Category 3

Next, we evaluate fluid limits for two instances of category 3, which contain jobs of both set I and set II of Johnson's rule. For instance 4, we obtain piecewise linear functions as for instances 0 to 3, with the exception that there are more "pieces" since there are more job types. In addition, each of the seven policies described earlier results in a different processing

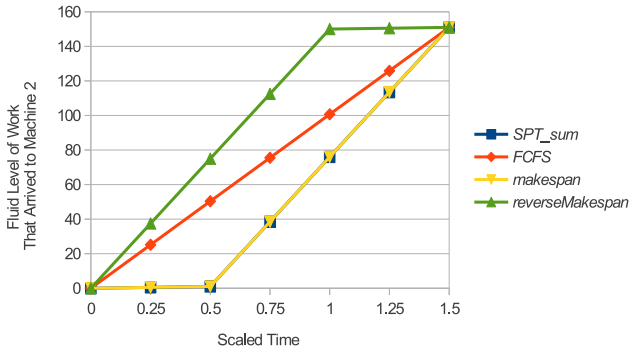

Figure 9.1: Fluid Limits for Instance 0.

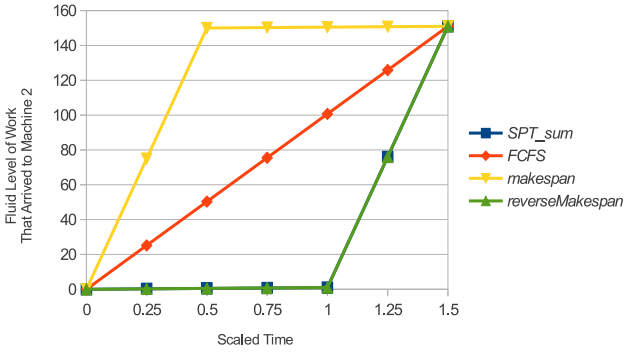

Figure 9.2: Fluid Limits for Instance 1.

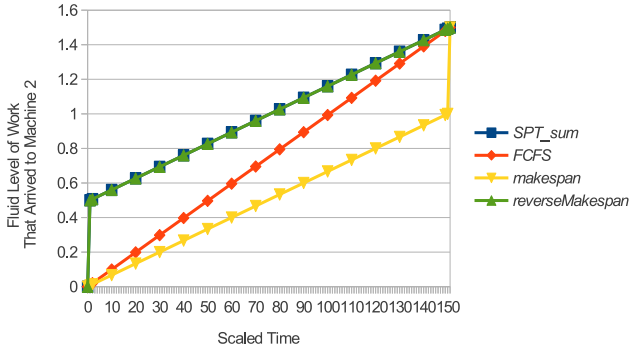

Figure 9.3: Fluid Limits for Instance 2.

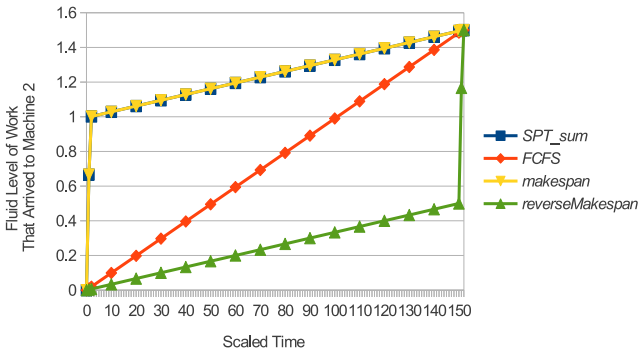

Figure 9.4: Fluid Limits for Instance 3.

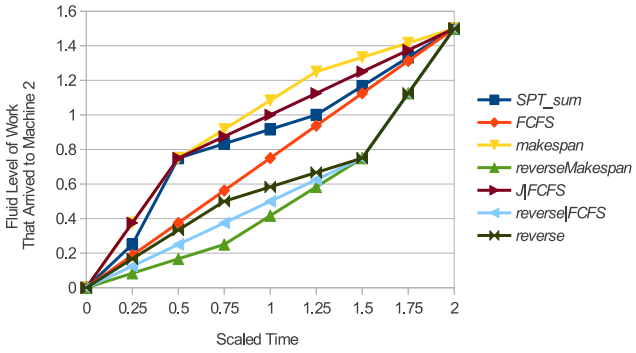

Figure 9.5: Fluid Limits of Work Arrived at Machine 2 for Instance 4.

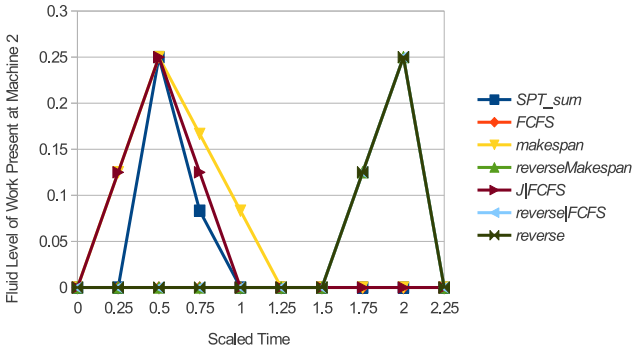

Figure 9.6: Fluid Limits of Work Present at Machine 2 for Instance 4.

sequence and hence different fluid limit functions. These are shown in Figures 9.5 and 9.6.

The next instance we consider has exponentially distributed processing times and was empirically evaluated in Chapter 7. With the exponential assumptions, it is more difficult to analyze the fluid limits of policies that use processing time information than when the processing times are deterministic or determined by a two-point distribution. Therefore, we do not present exact fluid limits for *makespan*, SPT_{sum}, *reverse* and *reverseMakespan*: we leave the evaluation of the exact limits for these policies for future work. However, in Figure 9.7, we show the workload arriving to machine 2 for the initial sub-problem of a particular instance with exponentially distributed processing times and a large number of jobs initially present in the system, which gives an indication of the fluid trajectories for *makespan*, SPT_{sum}, *reverse* and *FCFS*. In Figure 9.8, we show the workload present at machine 2 over time for the same sub-problem.

We now consider the fluid limit for *J|FCFS*, which is easier to evaluate since scheduling the jobs in *FCFS* order in each set implies that we can directly apply the Law of Large Numbers. The probability that job j falls into set I is $p = P(\eta_1(j) \leq \eta_2(j)) = \frac{\mu_1}{\mu_1+\mu_2}$, while the probability that job j will be in set II is $1-p = P(\eta_1(j) > \eta_2(j)) = \frac{\mu_2}{\mu_1+\mu_2}$. Denote the rate of processing for set I jobs as $\tilde{\mu}_1$ for machine 1 and $\tilde{\mu}_2$ for machine 2. Similarly, let the processing rate for set II jobs be $\hat{\mu}_1$ and $\hat{\mu}_2$ for machine 1 and 2, respectively. Equations (9.11)–(9.14) provide definitions for these processing rates in terms of μ_1 and μ_2.

$$E(\eta_1(j)|\eta_1(j) \leq \eta_2(j)) = \frac{1}{\mu_1+\mu_2} = \frac{1}{\tilde{\mu}_1} \tag{9.11}$$

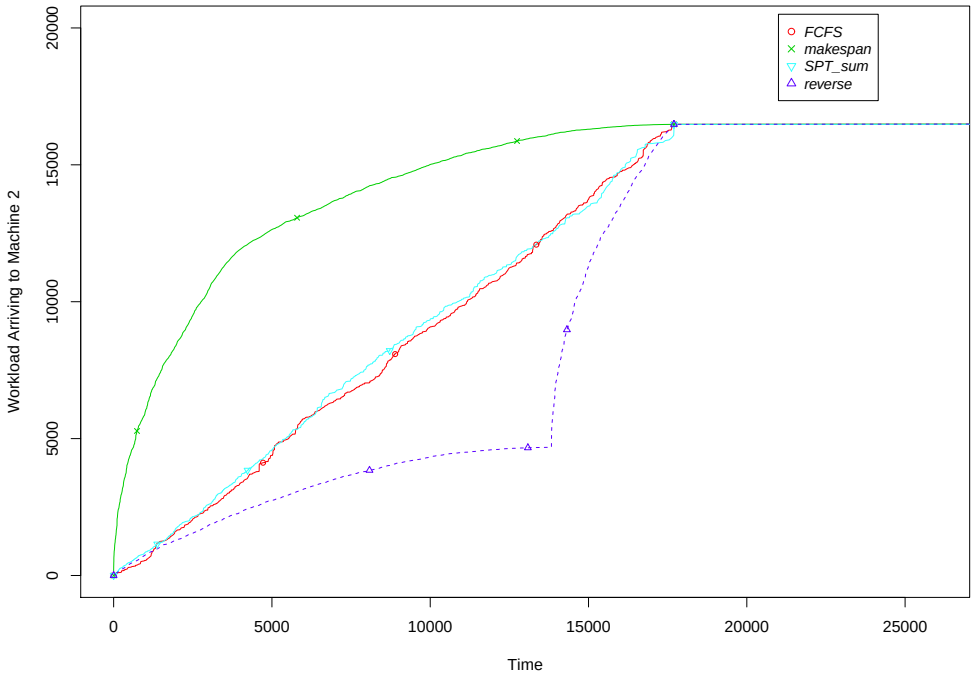

Figure 9.7: Workload Arriving to Machine 2 for Instance 5.

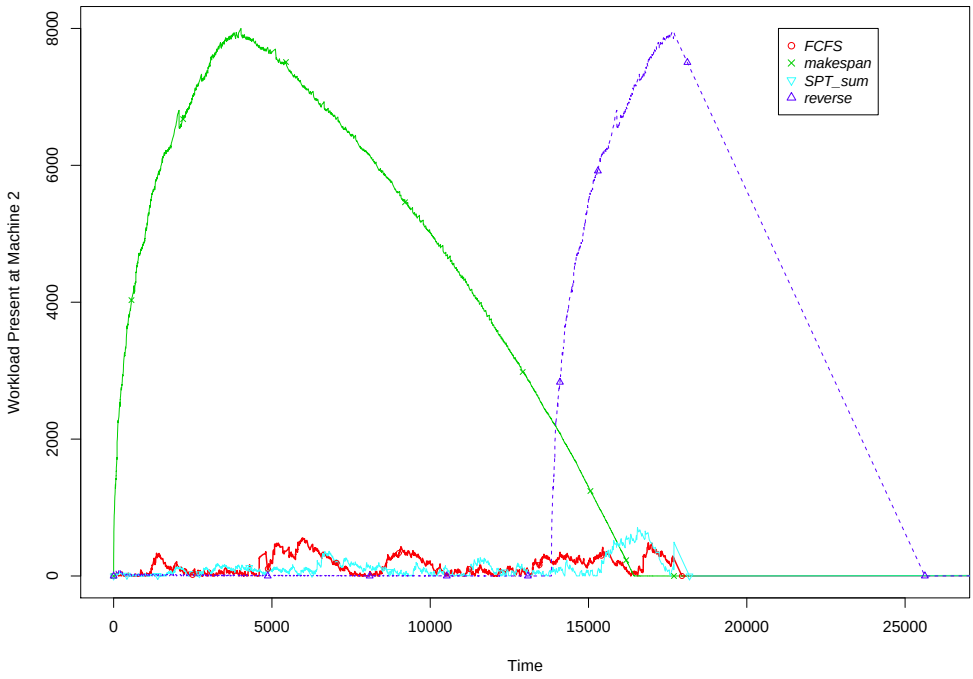

Figure 9.8: Workload Present at Machine 2 for Instance 5.

$$E(\eta_2(j)|\eta_1(j) \leq \eta_2(j)) = \frac{2\mu_2 + \mu_1}{\mu_2(\mu_1 + \mu_2)} = \frac{1}{\tilde{\mu}_2} \tag{9.12}$$

$$E(\eta_1(j)|\eta_1(j) > \eta_2(j)) = \frac{2\mu_1 + \mu_2}{\mu_1(\mu_1 + \mu_2)} = \frac{1}{\hat{\mu}_1} \tag{9.13}$$

$$E(\eta_2(j)|\eta_1(j) > \eta_2(j)) = \frac{1}{\mu_1 + \mu_2} = \frac{1}{\hat{\mu}_2} \tag{9.14}$$

Let $\mathcal{E}_2^I(t)$ and $\mathcal{E}_2^{II}(t)$ be the workload arrival processes to machine 2 for set I and set II, respectively. Then, using the definitions above, we find that their derivatives are $\dot{\mathcal{E}}_2^I(t) = \frac{\tilde{\mu}_1}{\tilde{\mu}_2}$ and $\dot{\mathcal{E}}_2^{II}(t) = \frac{\hat{\mu}_1}{\hat{\mu}_2}$.

In this section, we evaluated the fluid limits of six problem instances with various characteristics. We compare the fluid makespans of different policies in Section 9.1.3 and the fluid limits of work arriving to machine 2 in Section 9.1.4. Our evaluation and comparison serve as the basis for the insights we provide in Section 9.2.

9.1.3 Comparison of Fluid Makespans

One observation we make from the above examples is that the fluid makespan for instances of category 1 and 2 is always the same for all policies we consider. We now state this property in Proposition 9.1.1 and prove it. We denote by $\bar{C}_{max}^{\pi}$ the fluid makespan of a static problem instance under policy π.

Proposition 9.1.1. *If a given static problem instance belongs to category 1 or 2, then $\bar{C}_{max}^{\pi}$ is the same for all non-idling π.*

Proof. If $\eta_1(j) \leq \eta_2(j)$, for all jobs j belonging to a problem instance (i.e., category 1), then the rate at which work arrives at machine 2 is always greater than or equal to 1, regardless of the scheduling policy. Combining this property with the fact that machine 2 is non-idling, we see that work leaves machine 2 at rate 1 regardless of the policy. Thus, on the fluid scale, the length of the sub-problem is equal to the length of the sub-problem on machine 2 for all non-idling policies.

If $\eta_1(j) > \eta_2(j)$, for all j (category 2 instance), then the rate at which work arrives at machine 2 is always less than 1. Hence, irrespective of the policy, machine 2 is not fully utilized, and the length of the sub-problem is defined by the length of time it takes to process all fluid on machine 1. Thus, on the fluid scale, the length of the sub-problem is the same under all non-idling policies. □

Figures 9.6 and 9.8 show that for category 3 instances, the fluid makespans of different non-idling policies may be different. We now formally establish the relationship between the fluid makespans of the seven policies of interest to us.

Proposition 9.1.2. *For a given static problem instance,*

$$\bar{C}_{max}^{makespan} = \bar{C}_{max}^{J|FCFS} = \bar{C}_{max}^{FCFS} \leq \bar{C}_{max}^{reverseMakespan} = \bar{C}_{max}^{reverse|FCFS} = \bar{C}_{max}^{reverse}. \quad (9.15)$$

Proof. The fact that $\bar{C}_{max}^{makespan} = \bar{C}_{max}^{FCFS}$ was shown in the proof of Proposition 8.3.2 in the previous chapter. $\bar{C}_{max}^{makespan} = \bar{C}_{max}^{J|FCFS}$ and $\bar{C}_{max}^{reverseMakespan} = \bar{C}_{max}^{reverse|FCFS} = \bar{C}_{max}^{reverse}$ since all of these policies divide jobs into set I and II in the same way and the fluid makespans of each set are the equal by Proposition 9.1.1.

To prove the inequality of Expression (9.15), note that the fluid makespan of the *makespan* policy provides a lower bound on the fluid makespan of any policy, since this policy minimizes the makespan on the original scale. Instance 4 and Figure 9.6 provide an example of a problem instance where the inequality is strict. The inequality becomes an equality for problems of category 1 and 2, as shown by Proposition 9.1.1, and illustrated by instances 0 to 3. □

In fact, the above proposition and its proof imply that non-idling policies that divide jobs into two sets as does Johnson's rule can be placed into two classes based on their fluid makespan. That is, all non-idling policies that schedule set I before set II result in the same fluid makespan as the *makespan* policy, while all policies that schedule set II before set I yield the same fluid makespan as *reverseMakespan*.

The above results have a direct implication for the polling system with a flow shop server. In this system, we know that the arrival process is independent of the scheduling policy and that a new sub-problem is constructed when the previous one completes on machine 2. By Proposition 9.1.2, we know that the first sub-problem completes at the same time for *makespan*, *J|FCFS*, *FCFS* and all other non-idling policies that schedule set I before set II. Thus, at the next review point, exactly the same sub-problem is solved by all these methods. By repeatedly applying Proposition 9.1.2, we see that all policies of this class find exactly the same fluid makespan for every sub-problem. The same is true for *reverseMakespan*, *reverse|FCFS*, *reverse* and all other non-idling policies that schedule set II before set I. The order of jobs within each set has no effect on the fluid makespan.

The above examination of fluid limits is important since it allows us to gain an understanding of the performance of policies that use processing time information that could not have been obtained via standard scheduling analysis tools. In particular, we have identified two classes of policies that, in general, result in different fluid makespans; we have also identified a condition under which the two classes perform identically (i.e., when an instance is of category

1 or 2). Since differences on the fluid scale imply significant differences on the original scale, our results lead to two conclusions. Firstly, without using interchange arguments or doing an experimental evaluation as would have been necessary if a pure scheduling perspective was taken, we establish that policies that schedule set I before set II achieve significantly lower makespans for static two-machine flow shop problems than those that schedule set II before set I. Secondly, by combining the static problem result with the analysis of Section 7.3.4, we find that policies that schedule set I before set II also perform significantly better in terms of the long-run mean flow time in a polling system with a two-machine flow shop server. We discuss additional insights gained from looking at the fluid limits in Section 9.2.

9.1.4 Comparison of Fluid Limits

Next, we investigate the relationship among the fluid limits of different policies. Firstly, by looking at Figures 9.1–9.8, we see that the fluid limits of both the work arriving to machine 2 and the work present at machine 2 are dependent on the underlying distributions and the scheduling policies employed. This observation suggests a further investigation of the relationship between the shapes of the fluid limits and the scheduling policies may be interesting.

Secondly, we conjectured that $\bar{\mathcal{E}}_2^{makespan}(t) \geq \bar{\mathcal{E}}_2^{J|FCFS}(t) \geq \bar{\mathcal{E}}_2^{FCFS}(t) \geq \bar{\mathcal{E}}_2^{reverseMakespan}(t)$, $\forall t$.[3] The examples of the previous section show that our conjectures that $\bar{\mathcal{E}}_2^{makespan}(t) \geq \bar{\mathcal{E}}_2^{J|FCFS}(t)$ and $\bar{\mathcal{E}}_2^{FCFS}(t) \geq \bar{\mathcal{E}}_2^{reverseMakespan}(t)$ do not hold in general. In particular, in Figures 9.1 and 9.3 (instances 0 and 2), we see that the curve for *makespan* is below those of *FCFS* (which, in these cases, is identical to *J|FCFS*) and *reverseMakespan*. However, we can prove part of this relationship under the assumption of exponentially distributed processing times.

Proposition 9.1.3. *Assume that the processing times in either the dynamic flow shop with two machines or the polling system with a two-machine flow shop server are exponentially distributed. Then,* $\bar{\mathcal{E}}_2^{J|FCFS}(t) \geq \bar{\mathcal{E}}_2^{FCFS}(t)$, $\forall t$.

Proof. The initial sub-problem starts at time 0 and is exactly the same under all policies. Let $\bar{s}_0$ be the completion time, on the fluid scale, of set I under *J|FCFS* during this sub-problem. Then, for all t in the interval $[0, \bar{s}_0]$, $\bar{\mathcal{E}}_2^{J|FCFS}(t) = \frac{\tilde{\mu}_1}{\tilde{\mu}_2} t \geq \frac{\mu_1}{\mu_2} t = \bar{\mathcal{E}}_2^{FCFS}(t)$.

Since machine 1 is non-idling, all policies complete the current sub-problem on machine 1 at the same time on the fluid scale, $\bar{t}_0$. In other words, $\bar{\mathcal{E}}_2^{J|FCFS}(\bar{t}_0) = \bar{\mathcal{E}}_2^{FCFS}(\bar{t}_0)$.

Since $\bar{\mathcal{E}}_2^{J|FCFS}(t)$ and $\bar{\mathcal{E}}_2^{FCFS}(t)$ are non-decreasing, their values at $\bar{t}_0$ are equal, and there exists a time point $\bar{s}^*$ within the sub-problem such that $\bar{\mathcal{E}}_2^{J|FCFS}(\bar{s}^*) \geq \bar{\mathcal{E}}_2^{FCFS}(\bar{s}^*)$, then, for

[3] Recall that $\bar{\mathcal{E}}_2^{\pi}(t)$ represents the amount of work that has arrived at machine 2 by time t under policy π, on the fluid scale.

every time point in $[\bar{s}_0, \bar{t}_0]$, the amount of work that has arrived at machine 2 under *J|FCFS* is greater than or equal to the amount of work that has arrived under *FCFS*.

In the polling system, we know, by Proposition 9.1.2, that the sub-problems for *FCFS* and *J|FCFS* are identical on the fluid scale. In the dynamic flow shop, we know that the set of jobs that belong to a particular problem is independent of the scheduling policy; thus, there is also no difference among the policies in the fluid level at the start of a given sub-problem. Since in both systems the amount of work present at the beginning of every sub-problem is the same under *FCFS* and *J|FCFS*, we know that the above argument applies to every sub-problem. We therefore conclude that $\bar{\mathcal{E}}_2^{J|FCFS}(t) \geq \bar{\mathcal{E}}_2^{FCFS}(t),\ \forall t$, in both the polling system with a two-machine flow shop server and a dynamic two-machine flow shop. □

Based on Figure 9.7, we conjecture that the rest of the inequalities hold in the case of exponentially distributed processing times. Investigation of this conjecture is left for future work since it would require the derivation of exact fluid limits for *makespan* and *reverseMakespan*, which is beyond the scope of this dissertation.

Thirdly, notice that for instance 0, the SPT_{sum} and *makespan* curves are the same, while for instance 1, the SPT_{sum} curve is identical to that of *reverseMakespan*. These observations suggest that the shape of the fluid limits is dependent not only on the category of the instance (which is based on the relationship between the magnitudes of the processing times on machine 1 and 2) but also on its consistency. It appears that for category 1 instances, the SPT_{sum} and *makespan* fluid limits are the same when the processing times are consistent. Instances 2 and 3 suggest the opposite behaviour for category 2 instances: the SPT_{sum} and *makespan* curves are the same when the processing times are *inconsistent* and different when the processing times are consistent.

Next, consider the fluid limits of the work present at machine 2. In Figure 9.6, the maximum of $\bar{\mathcal{W}}_2^{\pi}(t)$ occurs at time 0.5 for policies that schedule set I ahead of II and at time 2 for policies that do the opposite. In Figure 9.8, we see a similar pattern in behaviour: the maximum occurs early in the schedule for *makespan*, while for *reverse* it happens at the time when the fluid level under the *makespan* policy has already reached 0. This behaviour reflects the fact that policies that schedule set I before set II maximize the utilization of machine 2 and hence result in smaller makespan values both on the original scale and the fluid scale.

9.2 Insights

Below, we discuss the insights gained from analyzing the behaviour of different policies on the fluid scale. Firstly, we demonstrate how fluid limit analysis can help identify key algorithmic

features. Secondly, we consider the relationship between the shape of the fluid limits and minimization of makespan. Finally, we look at the stability of the non-idling policies we have investigated above.

9.2.1 Identification of A Key Algorithm Feature

On the fluid scale, some details of the underlying system dynamics are not visible; the most significant features, however, become apparent. We have shown in Section 9.1.3 that all non-idling policies that divide jobs into two sets as does Johnson's rule can be categorized into two classes based on their fluid makespans: one consisting of policies that schedule set I before set II (plus *FCFS*), and the other consisting of policies that schedule set II before set I. The order in which the jobs are scheduled within each set has no effect on the fluid makespans. Since any difference on the fluid scale implies a significant difference on the original scale, these results suggest that appropriate division of jobs into two sets based on processing times and the order in which these sets are scheduled have a greater influence on the makespan than does the detailed sequencing within each set.

Identifying the division of jobs as a key feature of Johnson's rule suggests that we can use fluid analysis to determine whether a similar type of division in a system with more than two machines has the same significance for minimizing makespan. If such a division is identified, it can lead to the development of polynomial-time approximation algorithms for problems with more than two machines. It would then be interesting to determine if the difference in performance on the fluid level can allow us to find performance guarantees for these approximations.

Our observation that placing set II before set I can result in a large increase in makespan also has implications for scheduling in the dynamic flow shop. In this system, we know that a new sub-problem starts as soon as the previous sub-problem finishes on machine 1. If we assume that the instance parameters are such that most sub-problems consist of both set I and set II jobs, then we know that when one sub-problem completes and another starts, set II from sub-problem i is scheduled before set I of sub-problem $i+1$. From the perspective of the total makespan over the two sub-problems i and $i+1$, this is equivalent to using one of the reverse policies. Thus, to improve the total makespan over multiple sub-problems, we could adopt the following strategy: review the status of the system at the completion of set I of the current sub-problem; if any new set I jobs are present, these should be scheduled before any set II jobs. In this approach, however, we would need to determine some threshold on the number of set II jobs at which processing of set II should begin. Additionally, these observations imply that an idling policy may actually perform better than a non-idling policy since waiting an extra time unit may imply the arrival of a set I job that could be scheduled ahead of a previously-

arrived set II job. See the discussion following Lemma 8.2.1 for an example. Interestingly, Tran (2011) and Tran et al. (2013) use a similar observation to improve scheduling performance in a dynamic parallel machine environment with setup times.[4] In particular, one of their methods reviews the status of the system earlier than the end of the already-scheduled set of jobs: the status is reviewed when the current schedule prescribes a machine to switch from processing one class of jobs to another; if any of the newly-arrived jobs are of the class that the machine has just been working on, then a setup time can be avoided by scheduling these jobs ahead of those belonging to another class. We conjecture that similar types of rules for reviewing the system earlier can be developed for other dynamic settings.

9.2.2 Predicting Performance of Scheduling Algorithms

In this section, we investigate whether fluid limit analysis can help predict the performance of scheduling algorithms with respect to the makespan objective.

While intuitively it seems that the policy that sends work to machine 2 at the greatest possible rate on the fluid scale should be the one that minimizes makespan, instances 0 and 2 show that this is not necessarily true for category 1 and 2 instances, respectively. For these instances, the actual shape of the curve for fluid arriving to machine 2 has no effect on the fluid makespan since it does not change the proportion of time when machine 2 is busy. However, the shape of the curve does reflect differences in detailed scheduling on the original level, which also affect the makespan values. The discrepancy between intuition and the behaviour seen for instances 0 and 2 can be explained as follows. If we schedule, from set I, the job with the smallest processing time on machine 1 first, then on the original scale we ensure that machine 2 becomes busy as soon as possible, regardless of the job's machine 2 processing time. However, if we attempt to maximize the workload arrival rate from machine 1 to machine 2 on the fluid scale, then we may need to violate the SPT order on machine 1. We conclude that for instances of category 1 and 2, observing equal fluid makespans tell us that the difference in makespans on the original scale is small, but the behaviour of the fluid curves does not give us more information.

For instances of category 3, if we can show, for a particular problem instance, that the inequality in Expression (9.15) is strict, we can infer a significant difference in performance on the original scale between the two classes of non-idling policies that we introduced in Section 9.1.3. Instance 4, which is based on a two-point distribution, and instance 5, which is based on exponentially distributed processing times, provide examples where this property holds (see

[4] We discuss this work in Section 10.3.3.2.

Figures 9.6 and 9.8). In instance 5, the fluid makespan[5] for the *makespan* policy for the first sub-problem (given an initial fluid level of 1) is $\frac{1}{6}$, and for *reverse* is $\frac{1}{4}$. The difference in actual makespans is 17699.7 for *makespan* versus 25625.3 for *reverse*. From Section 7.3.4, we know that in the polling system such differences propagate and result in smaller mean flow times for the *makespan* approach.

Figure 9.8 also suggests that if both set I and II jobs are present, then the graphs of $\bar{\mathcal{W}}_2^{makespan}(t)$ and $\bar{\mathcal{W}}_2^{reverseMakespan}(t)$ provide the two extremes of the fluid shapes that can occur. We conjecture that, given a fluid limit curve between these two extremes, we can predict the performance of the corresponding policy with respect to the makespan objective based on how close the curve is to the fluid limits of *makespan* and *reverseMakespan*. For example, suppose we are given the fluid curves of SPT_{sum} and *FCFS* as in Figure 9.8. Based on the observation we just made, we would predict that *FCFS* performs slightly better than SPT_{sum} since it has somewhat bigger spikes initially (i.e., closer to *makespan*), while SPT_{sum} appears to leave more workload for later, as does *reverse*. We then find that the actual makespan of the sub-problem is 17951.9 for *FCFS* and 18189.8 for SPT_{sum}. Clearly, this example is not strong enough to provide conclusive evidence (recall also that Figure 9.8 does not show that exact fluid limits) and hence the generality of our observation requires further investigation, which we leave for future work.

In future work, we would also like to determine whether analysis of the fluid limits can help us predict the performance of algorithms with respect to the sum of completion times objective.

9.2.3 Stability of Non-Idling Policies in the Polling System

In the previous chapter, we established stability of *FCFS* and *makespan* policies for the polling system with a flow shop server. In this section, we supplement those results and thus provide a more complete picture of stability of non-idling policies that utilize processing time information in a polling system with a two-machine flow shop server.

Firstly, based on the result of Proposition 9.1.2, we know that the two classes of policies that we identified in Section 9.1.3 have identical stability regions.

Corollary 9.2.1. *The stability regions of* reverse*,* reverseMakespan*,* reverse|FCFS *and all other non-idling policies that schedule set II of Johnson's rule before set I are the same. The stability regions of* FCFS*,* makespan*,* makespan|FCFS *and all other non-idling policies that schedule set I of Johnson's rule before set II are the same.*

[5]The fluid makespan is not evident from Figure 9.8 since this figure is based on a particular instance and does not display the true fluid limits. The calculation of fluid limits is evident from Equations (9.16) and (9.21), which are presented in the next section.

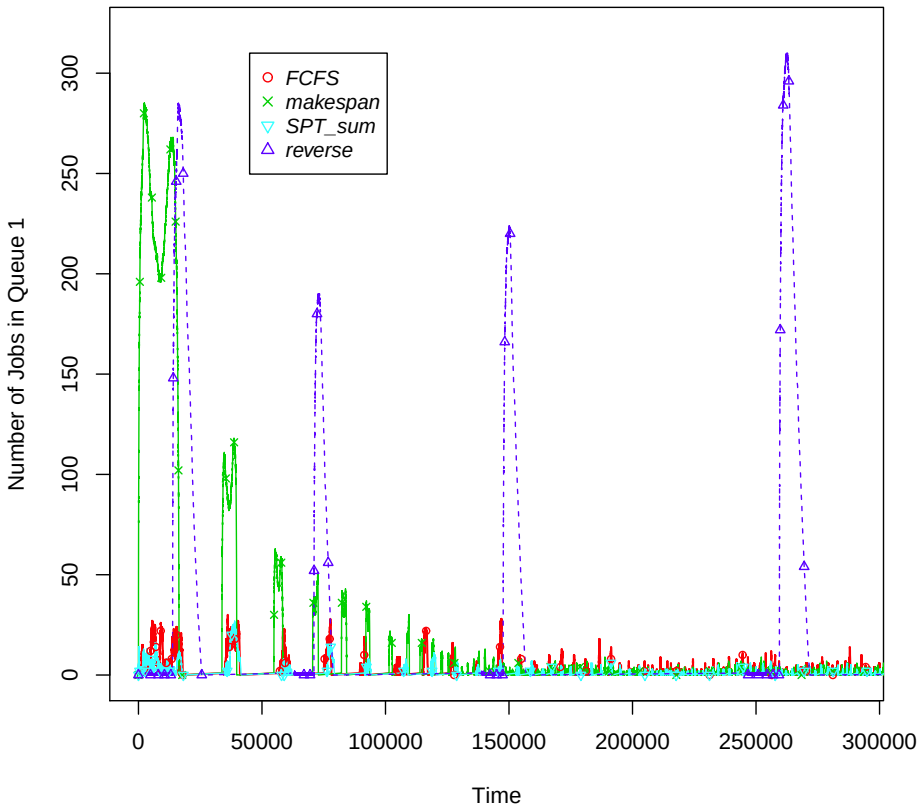

Figure 9.9: Number of Jobs at Machine 2 in Queue 1 Over Time for an Example Instance. This figure also appears as Figure 8.9.

Proof. The corollary follows from Proposition 9.1.2, the fact that all of the considered policies are non-idling, and that the arrival rate to the system is independent of the policy. □

Secondly, in the special case when sub-problems are category 1 or 2 instances, the following proposition is true.

Proposition 9.2.1. *In the polling system with a two-machine flow shop server, if for every sub-problem it holds that either* $\eta_1(j) \leq \eta_2(j)$*, for all* j*, or* $\eta_1(j) > \eta_2(j)$*, for all* j*, then all non-idling policies are stable under the same condition as* FCFS *and* makespan.

Proof. From Proposition 9.1.1, we know that if a sub-problem consists only of set I or only of set II jobs, then on the fluid scale, the length of the sub-problem is the same under all non-idling policies. We also know that the arrival process is not dependent on the scheduling policy. Therefore, if the length of the first sub-problem on the fluid scale is the same under all non-idling policies, then it follows that the subsequent sub-problem has the same length on the fluid

scale as well. Thus, assuming that every sub-problem consists either of set I jobs only or of set II jobs only, we know that all non-idling policies encounter exactly the same sub-problems, and that their lengths, on the fluid scale, are identical to those of *FCFS* and *makespan*. Thus, when all sub-problems consist exclusively of jobs from set I or jobs from set II, then all non-idling policies are stable under the same condition as *makespan* and *FCFS*. □

Figure 8.9 of the previous chapter, repeated here as Figure 9.9, suggests that for category 3 instances, not all non-idling policies are stable under the same condition. We now formally prove that the system depicted in Figure 9.9, and other systems with similar parameters, are indeed unstable under the *reverse* policy.

The system depicted in Figure 9.9 is a symmetric polling system with a two-machine flow shop server and with processing times that are exponentially distributed with equal means on machines 1 and 2 ($1/\mu_1 = 1/\mu_2$). Since the system is symmetric (i.e., the processing and arrival rates are not queue dependent), we use notation that is slightly different (and simpler) than that used in the previous chapter. We number the sub-problems starting from 0. At the start of sub-problem i on the fluid scale, the fluid level under policy π is denoted by $\bar{Q}_i^{\pi}$ and the fluid makespan of the corresponding sub-problem is equal to $\bar{C}_{max_i}^{\pi}$. We denote the fluid level present in set I of sub-problem i by $\bar{Q}_i^{\pi,I}$ and in set II by $\bar{Q}_i^{\pi,II}$. If *FCFS* is employed, then the fluid makespan of sub-problem i is

$$\bar{C}_{max_i}^{FCFS} = \frac{\bar{Q}_i^{FCFS}}{\mu_1}. \tag{9.16}$$

Since $\mu_1 = \mu_2$, by using Equations (9.11)–(9.14) we find that

$$\frac{1}{\tilde{\mu_2}} = \frac{1}{\hat{\mu_1}} = \frac{3}{2\mu_1}. \tag{9.17}$$

If *reverse* is employed, the fluid makespan of sub-problem i is

$$\begin{aligned}
\bar{C}_{max_i}^{reverse} &= \frac{\bar{Q}_i^{reverse,I}}{\tilde{\mu_2}} + \frac{\bar{Q}_i^{reverse,II}}{\hat{\mu_1}} && (9.18)\\
&= \frac{3\bar{Q}_i^{reverse,I}}{2\mu_1} + \frac{3\bar{Q}_i^{reverse,II}}{2\mu_1} && (9.19)\\
&= \frac{3}{2\mu_1}(\bar{Q}_i^{reverse,I} + \bar{Q}_i^{reverse,II}) && (9.20)\\
&= \frac{3}{2\mu_1}\bar{Q}_i^{reverse}. && (9.21)
\end{aligned}$$

We now state and prove the instability result of interest to us.

Theorem 9.2.1. *Consider a symmetric polling system with B queues and a two-machine flow shop server. The arrival rates to the queues are general with rate λ and the processing times are exponentially distributed with rate μ_1 on both machines. If $\frac{3B\lambda}{2\mu_1} > 1$ then the system is unstable under the* reverse *policy.*

Proof. We compare the given system to an equivalent polling system with arrival rate $\lambda^{FCFS} = \frac{3\lambda}{2}$ (the rest of the parameters remain the same) operating under *FCFS*. We show that the amount of fluid present at the start of sub-problem i, for all i, is the same for the original system with arrival rate λ under *reverse* and the equivalent system with arrival rate λ^{FCFS} under *FCFS*, i.e., $\bar{Q}_i^{reverse} = \bar{\mathcal{Q}}_i^{FCFS}$, where the notation $\bar{\mathcal{Q}}_i^{\pi}$ represents the fluid level at the start of sub-problem i under policy π in the equivalent system with the adjusted arrival rate.[6] We denote the length of sub-problem i under policy π in the equivalent system by $\bar{\mathcal{C}}_{max_i}^{\pi}$.

Assume that, at time 0, there is one unit of fluid in the original system and in the equivalent system with adjusted arrival rate. We show by induction that $\bar{Q}_i^{reverse} = \bar{\mathcal{Q}}_i^{FCFS}$ for all i.
Base Case: $\bar{Q}_0^{reverse} = \bar{\mathcal{Q}}_0^{FCFS} = 1$ and $\bar{Q}_s^{reverse} = \bar{\mathcal{Q}}_s^{FCFS} = 0$ for all $s < 0$.
Inductive Hypothesis: Assume that $\bar{Q}_k^{reverse} = \bar{\mathcal{Q}}_k^{FCFS}$ for all $k < i$.
Inductive Step:

$$\begin{aligned} \bar{\mathcal{Q}}_i^{FCFS} &= \frac{3\lambda}{2}(\bar{\mathcal{C}}_{max_{i-1}}^{FCFS} + \bar{\mathcal{C}}_{max_{i-2}}^{FCFS} + \bar{\mathcal{C}}_{max_{i-3}}^{FCFS} + \bar{\mathcal{C}}_{max_{i-4}}^{FCFS} + \bar{\mathcal{C}}_{max_{i-5}}^{FCFS}) && (9.22)\\ &= \frac{3\lambda}{2}\frac{1}{\mu_1}(\bar{\mathcal{Q}}_{i-1}^{FCFS} + \bar{\mathcal{Q}}_{i-2}^{FCFS} + \bar{\mathcal{Q}}_{i-3}^{FCFS} + \bar{\mathcal{Q}}_{i-4}^{FCFS} + \bar{\mathcal{Q}}_{i-5}^{FCFS}) && (9.23) \end{aligned}$$

where the second equality is based on the calculation of fluid makespan of Equation (9.16).

Similarly,

$$\begin{aligned} \bar{Q}_i^{reverse} &= \lambda(\bar{C}_{max_{i-1}}^{reverse} + \bar{C}_{max_{i-2}}^{reverse} + \bar{C}_{max_{i-3}}^{reverse} + \bar{C}_{max_{i-4}}^{reverse} + \bar{C}_{max_{i-5}}^{reverse}) && (9.24)\\ &= \lambda\frac{3}{2\mu_1}(\bar{Q}_{i-1}^{reverse} + \bar{Q}_{i-2}^{reverse} + \bar{Q}_{i-3}^{reverse} + \bar{Q}_{i-4}^{reverse} + \bar{Q}_{i-5}^{reverse}). && (9.25) \end{aligned}$$

It follows that $\bar{\mathcal{Q}}_i^{FCFS} = \bar{Q}_i^{reverse}$.

Therefore, the fluid level at the start of every sub-problem in the two systems is the same. As a consequence, we know that if the equivalent system with adjusted arrival rate under *FCFS* is unstable, then so is the original system under *reverse*. It follows from Proposition 8.3.1 that the equivalent system is unstable under *FCFS* if $B\frac{\lambda^{FCFS}}{\mu_1} = \frac{3B\lambda}{2\mu_1} > 1$. Hence, the original system under *reverse* is unstable if this condition holds also. □

In the example of Figure 9.9, $B = 5$, $\lambda = 1$, $\mu_1 = 6$ and so $\frac{3B\lambda}{2\mu_1} = \frac{15}{12} > 1$, implying instability of the system if the *reverse* policy is used. As a consequence of Corollary 9.2.1, the

[6] This notation is necessary since $\bar{Q}_i^{FCFS} \neq \bar{\mathcal{Q}}_i^{FCFS}$.

system is also unstable under *reverseMakespan*, *reverse|FCFS* and all other non-idling policies that schedule set II before set I.

In summary, the results proven in this section define stability properties of a large group of policies that are dependent on processing time information, which has not previously been done. Our work also demonstrates two general approaches for proving stability or instability of periodic scheduling methods: compare, on the fluid scale, either the sub-problem length or the amount of fluid present at the start of each sub-problem for the scheduling method of interest and a simpler policy known to be stable or unstable. These approaches are important for developing a further understanding of stability of periodic scheduling policies that utilize processing time information. One of the next steps in the study of stability of periodic scheduling methods is investigating the stability of the *completionTime* policy, which does not have a specific structure that can be utilized nor a property that makes it easily comparable to *FCFS*.

9.3 Conclusion

Continuing the integration of scheduling and queueing theory on the theoretical level, in this chapter we proposed to use fluid limit analysis as a tool for investigation of scheduling algorithm performance. Specifically, we studied the fluid limits of work arriving to machine 2 and present at machine 2 in a two-machine flow shop under six different assumptions regarding processing times. These fluid limits helped us to gain several insights. First, we showed that division of jobs into two sets based on processing times is a key component of Johnson's rule, having a significant effect on makespan. Second, we made observations regarding the usefulness of fluid limit analysis for prediction of policy performance with respect to the makespan objective. Third, a better understanding of fluid limits allowed us to prove the stability of various non-idling periodic scheduling methods that utilize processing time information.

Thus, this chapter provided additional evidence of the potential of applying queueing theory methodologies to scheduling. In the next chapter, we present future work on the integration of queueing theory and scheduling.

Chapter 10

Scheduling and Queueing Theory Future Work

In this dissertation, we have adopted the viewpoint that integration of queueing and scheduling can occur on three different levels: conceptual, theoretical and algorithmic. The current chapter outlines directions for future work on each of these levels.

Some ideas for integration of queueing and scheduling were discussed in the literature review chapter (Chapter 6). In the current chapter, we classify these ideas according to the three-level framework but do not repeat the details. Additionally, we present extensions resulting from the work presented in Chapters 7, 8 and 9, as well as other ideas for integration that fit into the three-level framework and have not been mentioned in the literature review. We discuss the algorithmic level of integration in more detail than the other levels, outlining a general method for creating hybrid models, and discussing two examples of how the method can be applied, one of which has already appeared.

10.1 Conceptual Level

The conceptual level of integration of queueing and scheduling focuses on combining problem settings, concepts and assumptions from the two areas. Further work in this direction can result in a richer framework for modelling real-world problems and can lead to new insights about dynamic scheduling.

10.1.1 Summary of Ideas from Chapter 6

The following ideas for future work on the conceptual level were proposed in the literature review chapter:

- continuation of the work by Harchol-Balter (2011), Nuyens et al. (2008) and Wierman et al. (2005) on using queueing analysis to derive theoretical performance guarantees for dispatching rules under particular distributional assumptions (Section 6.1.2.2.1);

- development of new priority queueing models based on dispatching rules (e.g., composite dispatching rules such as the Apparent Tardiness Cost heuristic (Pinedo, 2009)) and their analysis (Section 6.1.2.2.1);

- development of a general framework for integrating queueing and scheduling based on the observation that *any* scheduling algorithm can be represented as a function of M, P and A, where M is the decision mode, P is a priority function and A is an arbitration rule (Jaiswal, 1982; Ruschitzka and Fabry, 1977) (Section 6.1.2.2.1);

- study of problems that require optimization of polling order together with optimization of within-queue sequencing (e.g., simultaneously minimizing the expected length of a production cycle and the total job tardiness in a manufacturing facility) (Section 6.1.2.2.2);

- investigation of how polling models can represent systems with different objectives at different decision-making levels (Section 6.1.2.2.2);

- investigation of polling systems under the *limited-k* discipline where the k jobs are selected using rules other than *FCFS* (Section 6.1.2.2.2);

- improvement of polling system performance by employing scheduling approaches to optimize within-queue scheduling (Section 6.1.2.2.2);

- further investigation of the link between lot-sizing and polling systems (Winands, 2007), and the use of lot-sizing or batch scheduling methods for optimization of polling order within polling systems (Section 6.1.2.2.2);

- investigation of the use of polling models with precedence constraints (Khamisy et al., 1992) for modelling scheduling problems with precedences and the task management problem (Myers et al., 2007) (Section 6.1.2.2.2);

- study of the relationship between the restless bandit problem and stochastic variations of resource constrained project scheduling discussed by Mercier and Van Hentenryck (2008) (Section 6.1.2.2.4);

- comparison of the results from the online scheduling literature and the study of priority queues (Section 6.2);

- investigation of the links between queueing design and control models in which the goal is to find optimal values for controllable queue parameters other than the order of service (see Section 6.1.1.3) and scheduling (Section 6.1.2).

10.1.2 Extensions of Chapter 7

Extensions of the work in Chapter 7 are based on combining problem settings and assumptions from queueing and scheduling.

10.1.2.1 Combination of Problem Settings

One direction for future work is to explore variations of the system introduced in Chapter 7, which is a hybrid of a classical polling system from the queueing literature and a flow shop, which has received significant attention in scheduling.

As indicated in Section 6.1.2.2.2, polling systems can be controlled by choosing the polling order, the queue service discipline and the service order within a queue. In our work, we have assumed that both the polling order and the queue service discipline are fixed, and focused on the optimization of the service order within each queue. It would be interesting to relax this assumption and study optimization of the polling order, the queue service discipline as well as combinations of the three types of decisions in a polling system with a flow shop server.

Additionally, we could study a polling system with a more complex machine structure within the server, such as a job shop. We conjecture that if the gated discipline is used, then minimizing the makespan of each sub-problem would lead to smaller overall mean flow times than minimizing the flow time of each sub-problem or using queueing policies, regardless of the combinatorial structure of the server. As a result, the trade-off between short-run and long-run performance measures discussed in Section 7.3 would also be observed for a polling system with a server that is combinatorially more complex than a flow shop. Research on polling systems with such servers has several drawbacks, however. Firstly, finding the optimal makespan for each sub-problem may become impractical since an increase in a sub-problem's complexity could lead to significantly longer run times. Secondly, based on the above conjecture, it seems unlikely that we could obtain any new significant insights from the study of polling systems with more complex servers. It would, however, be fruitful to identify practical applications of polling systems with complex servers (e.g., one potential application[1] may be in polymerization plants (Terrazas-Moreno et al., 2008)), determine whether the insights gained from abstract models would be useful for these applications, and investigate any related questions.

[1]This suggestion is due to Dr. Mark Boddy.

10.1.2.2 Combination of Assumptions

In Chapter 7 we used a combination of assumptions from queueing and scheduling: we assumed knowledge of inter-arrival and processing time distributions, as in queueing theory, and we assumed that the processing time of a job becomes known with certainty upon its arrival to the system, as in scheduling. This combination of assumptions is of both theoretical and practical significance. From the theoretical perspective, there has not been much previous work on this "middle ground" between queueing and scheduling assumptions, although there has been a significant amount of work on the extremes (i.e., knowledge of distributions only, or knowledge of exact processing times only). From the practical perspective, there exist applications (e.g., in manufacturing and computer processing) with dynamic arrivals and well-estimated processing times. Thus, it would be interesting to examine a wider range of problems under a combination of queueing and scheduling assumptions.

One example of such a problem is from the paper by Atkins and Chen (1995). In this problem, three machines serve six job classes: machine 1 serves classes 1 and 2, machine 2 serves classes 3 and 4, and machine 3 serves classes 5 and 6. Jobs arrive to the different classes according to stochastic processes with known inter-arrival rates. When a job of class k completes processing, it turns into class l with a given probability p_{kl}. Each class is processed by the corresponding machine, with processing times being drawn from known distributions with class-dependent processing rates. If, in addition, we assume that once a job arrives its exact route through the system and its processing times on all machines become known with certainty, then we obtain a hybrid of a queueing network and a job shop with recirculation.[2] We can study three variations of this problem:

1. All jobs have the same weight, regardless of their class membership. The objective is to optimize, over the long run, the mean flow time of jobs.

2. All jobs belonging to the same class have the same weight. The objective is to optimize, over the long run, the mean weighted flow time of jobs.

3. Each job has a weight that is not dependent on its class membership and which becomes known upon its arrival to the system. The objective is to minimize the mean weighted flow time of jobs over a long time horizon.

This problem setting borrows several assumptions from queueing theory that are usually not made in classical scheduling models:

- knowledge of inter-arrival and processing time distributions,

[2] In scheduling, *recirculation* means that a job may be processed by a particular machine more than once (Pinedo, 2003).

- division of jobs into classes,

- knowledge of the routing matrix, which states the probability with which a job of class k turns into a job of class l.

In addition, it uses a classical scheduling assumption that is not typically made in queueing models: knowledge of a job's route through the system and its exact processing times on each machine become known upon the job's arrival. Thus, from the scheduling perspective this problem setting is the dynamic version of a job shop with recirculation and with the additional distributional knowledge. From the queueing perspective, it is an open queueing network with realizations of all stochastic job characteristics becoming known at the arrival of the job, rather than at its departure. As far as we are aware, such problems have not been addressed in the literature.

10.1.3 Other Ideas for Conceptual Integration

In this section, we present other conceptual differences between queueing and scheduling, and discuss how these differences can be leveraged for obtaining new insights about dynamic scheduling.

10.1.3.1 Modelling Waiting

Both queueing and scheduling models represent environments in which tasks need to be processed by a set of resources. Classical scheduling models focus on how the resources are utilized by the jobs and do not include an explicit representation of the queue these jobs form while waiting for processing. Queueing theory, on the contrary, reasons about the waiting jobs and makes sequencing decisions based on the queue's state. For example, it may prescribe a machine to switch from one class of jobs to another when the length of the queue of the second class exceeds some threshold value. We could therefore investigate hybrid models with an explicit representation of both the state of the queue and the detailed sequencing decisions on the resource. Such models would require scheduling approaches that would be able to reason about both aspects of the problem.

10.1.3.2 Bottleneck Sub-network

In most environments studied by queueing and scheduling, there are machines which are more critical than others, either because they are in greater demand by the jobs or because they are

slower.[3] In both queueing and scheduling, such machines are typically referred to as *bottleneck* machines. In scheduling, it is well-known that an environment with a complicated machine setup but a single bottleneck may be modelled as a single machine scheduling problem. There also exist scheduling methods based on the bottleneck concept (Adams et al., 1988; Beck and Fox, 2000). Similarly, in queueing theory, bottlenecks have been used as part of decomposition approaches such as the Sequential Bottleneck Decomposition method of Dai et al. (1994).

Bottleneck machines are exactly the ones at which large queues tend to form, where the majority of waiting time is incurred, and where scheduling should have the biggest impact (Ravikumar and Narahari, 1994). This observation suggests dealing with the bottleneck sub-network using scheduling models, while representing the rest of the system using queueing. This hybrid representation of a system has several advantages. Firstly, by focusing on the combinatorics of the bottlenecks, we are likely to get significant improvements in performance. Secondly, detailed scheduling of a sub-network should require less computational effort than detailed scheduling of the whole network.

We could argue, on the contrary, that queueing theory should be used to derive a policy for controlling the bottleneck sub-network while the rest of the system can be represented via scheduling models. Such an approach may be necessary because in practice it may be impossible to use detailed scheduling models of the bottleneck sub-network due to time constraints.[4] In addition, queueing theory could be used to analyze the stability of the bottleneck sub-network.

In both of these hybrid models some method of communication between the queueing and the scheduling sub-models is necessary for obtaining a representation of the overall system. For example, we can use queueing theory to determine the arrival process of jobs from the non-bottleneck part of the system to the bottleneck sub-network; this arrival process can then be used by the scheduling model as input.

10.2 Theoretical Level

The theoretical level of integration focuses on combining theoretical notions from queueing and scheduling. In Chapter 8 we showed that the queueing notion of stability is an important and interesting performance measure for dynamic scheduling problems, and that queueing methodologies can be applied to prove stability of periodic scheduling methods. In Chapter 9, we analyzed different periodic scheduling approaches using fluid limits. It would be interesting to pursue both a further study of stability and fluid limits in dynamic scheduling, and an

[3] We can relate this notion of being critical to the queueing notion of offered work load rate, which is the ratio of the arrival rate to the processing rate (Gross and Harris, 1998).

[4] The idea of looking at the bottleneck sub-network using one approach, and at the rest of the system using another was suggested by my thesis committee during the qualifying exam meeting.

examination of other theoretical notions from the two areas.

10.2.1 Summary of Ideas from Chapter 6

The following ideas for future work on the theoretical level have been proposed in the literature review chapter:

- investigation of the steady-state performance of composite dispatching rules (Section 6.1.2.2.1);
- application of descriptive results from polling systems to the development of bounds on optimal schedules and within scheduling algorithms (Section 6.1.2.2.2).

10.2.2 Extensions of Chapters 8 and 9

Below, we discuss future work arising from Chapters 8 and 9.

10.2.2.1 Stability

One direct next step from the work presented in Chapter 8 is the investigation of our conjecture that Theorem 8.2.2 can be shown true for the case with more than two machines using a fluid model approach. Stability analysis of SPT_{sum} and *completionTime*, the remaining two methods presented in Chapter 7, is also of interest. Proving stability of the *completionTime* approach may prove to be especially challenging since it possesses neither a specific structure that can be utilized in the proof nor a property that makes it easily comparable to *FCFS*.

Conceptual integration could motivate additional stability questions. For example, we could investigate stability of the *FCFS* and *makespan* policies assuming that a *limited-*k, rather than a gated, queue service discipline is adopted in the polling system. Specifically of interest is the question of whether the stability conditions would be identical for the two policies. In the case of the limited-1 discipline, the *FCFS* and *makespan* policies are equivalent, and so, trivially, they would be stable under the same condition. As soon as k is increased to two, however, it becomes possible for *FCFS* and *makespan* to encounter different sub-problems, which eliminates the possibility of proving strong sample paths results of the type presented in Lemma 8.2.1. We conjecture that the equivalent of Lemma 8.2.1 holds if one of the following two conditions for ensuring that the two methods will solve exactly the same sub-problems is true: (1) the arrival rate is such that exactly k jobs are always available at the end of every sub-problem, or (2) there are k or more jobs available at the end of every sub-problem and the two methods select, out of the available jobs, exactly the same k jobs.

More generally, we hope this dissertation will serve as a motivation for stability analysis of more complex scheduling methods based on processing times (e.g., a method which minimizes flow time within each static sub-problem using a mixed-integer or a constraint programming approach). Such analysis will provide important long-run guarantees for the periodic approach to dynamic scheduling problems. If a given periodic scheduling technique is shown to be unstable, it then becomes interesting to determine if augmenting the technique with reasoning based on queueing theory could lead to a stable method.

10.2.2.2 Fluid Limit Analysis

In Chapter 9, we looked at the fluid limits of work arriving to machine 2 and work present at machine 2 for various policies. While our analysis yielded some insights and showed potential for fluid analysis to be used as a tool in evaluation of scheduling algorithms, several parts of our work are not conclusive. In particular, in future work we would like to obtain stronger results regarding how well the fluid limits can predict the performance of methods with respect to makespan and total completion time objectives. Additionally, we would like to: further investigate the relationship between the fluid limits of work arriving to machine 2 that is discussed in Section 9.1.4; derive exact limits of policies dependent on processing times; determine whether the fluid limits of SPT_{sum} and *completionTime* models are similar; determine whether our analysis can be extended to problems with more than two machines and whether it can lead to the development of polynomial-time approximation algorithms.

10.2.3 Other Ideas for Theoretical Integration

Another direction for further research is the investigation of other theoretical queueing notions that could be borrowed by scheduling and of theoretical notions existing in the scheduling literature that may be of interest in queueing. The goal of such an investigation would be to formalize dynamic scheduling methodologies and results, and to develop a theoretical understanding of when various predictive-reactive methods would perform better than queueing-type priority rules, or vice versa, and why.

10.3 Algorithmic Level

As discussed in Chapter 6, scheduling algorithms developed within queueing theory are typically concerned with the allocation of resources to job classes rather than with detailed sequencing decisions. They are likely to optimize long-run objectives but may perform poorly

for a given short time horizon. Scheduling methods, in contrast, focus on sequencing and optimization of short-run performance but may be myopic. Thus, combining algorithmic components from queueing theory and scheduling could lead to the development of hybrid algorithms for effectively addressing dynamic scheduling problems.

10.3.1 Summary of Ideas from Chapter 6

The following ideas for integration of queueing and scheduling on the algorithmic level have been proposed in the literature review chapter:

- empirical comparison of the performance of queueing policies (e.g., priority policies, fluid model-based heuristics), predictive-reactive scheduling methods, online stochastic combinatorial optimization methods and hybrid algorithms, similar in style to the algorithmic work of Tran (2011) (Sections 6.1.2.2.2, 6.1.2.3.1, 6.1.2.3.2 and 6.2);

- examination of the MDP model as a meta-framework for the construction of queueing/scheduling hybrids: if a queueing algorithm and a scheduling algorithm are chosen, and the state of the problem is compactly represented, then we can define an action as the choice to follow the queueing algorithm or the choice to follow the scheduling algorithm until the next decision time point (Section 6.1.2.1);

- investigation of whether it is useful to model part or all of a dynamic scheduling problem as a bandit problem and to incorporate Gittins indices into the constraints or the objective function of the models used to construct predictive schedules (Section 6.1.2.2.4);

- investigation of the applicability of the achievable region approach in scheduling (e.g., for derivation of performance bounds) (Section 6.1.2.3.1);

- derivation of new constraints for the achievable region approach based on techniques from scheduling (Section 6.1.2.3.1);

- investigation of operational-level scheduling policies that can also respect the tactical allocations provided by queueing methods (e.g., Markov Decision Process approaches, fluid models, Brownian models) (Sections 6.1.2.1 and 6.1.2.3.1), similar to the investigation of Aramon Bajestani et al. (2012) which combines higher-level maintenance decisions with lower-level operational ones;

- investigation of translation mechanisms from the queueing approximations and abstractions to implementable policies (this is closely related to the direction mentioned in the previous point) (Section 6.1.2.3.2);

- investigation of the idea of tracking for scheduling: firstly, predictive-reactive methods could be developed which aim to track the solution of the fluid model as do the trajectory-tracking policies in queueing, and, secondly, for static problems, there may be problem relaxations, such as preemptive versions of non-preemptive problems, that can be tracked (Section 6.1.2.3.2);

- use of a model that specifies high-level resource allocations as part of a problem decomposition method such as the logic-based Benders method of Hooker (2005) (Section 6.1.2.3.2); the work of Aramon Bajestani and Beck (2011, 2013) on the use of Benders decomposition in problems with maintenance and scheduling decisions would be useful in the study of this direction.

Another avenue for future work is to continue the empirical study of Chapter 7, comparing our proposed periodic scheduling methods to dispatching rules discussed by Sarper and Henry (1996) and investigating the performance of periodic methods with multiple objectives (see Section 7.4.1).

10.3.2 A General Framework for Algorithmic Integration

One general framework for developing hybrid queueing/scheduling algorithms is based on the progressive scheduling approach of Bidot (2005). Hybrids created using this framework build the schedule incrementally: the status of the system is reviewed at various points in time and a static scheduling problem consisting of all or a subset of the unscheduled jobs is solved at each review point. This framework can be described in terms of three steps:

1. Use one of the queueing theory approaches described in Chapter 6. Typically, queueing methodologies for scheduling result either in an implementable policy (e.g., a priority rule such as shortest processing time first) or in a high-level solution that cannot be directly implemented (e.g., proportions of time that each machine should spend on each job class). Both the policy and the high-level solution can be used as input data for the hybrid method.

2. Define an approach for progressively[5] constructing the schedule, which includes

   (a) Determining the *reasoning* horizon for selecting the data on which to reason.

   (b) Determining a rule for choosing the subset of jobs that will be scheduled at each review point. Some possible approaches are:

[5]The progressive scheduling approach was proposed by Bidot et al. (2009). It is presented here in a slightly different manner than in the work by Bidot (2005) and Bidot et al. (2009).

- take all of the unscheduled jobs that have arrived to the system by the current review time point. This is, essentially, the equivalent of the *gated* queue service discipline used in polling systems (see Section 6.1.2.2.2 for a discussion of policies in polling systems).
- select, according to some rule (e.g., *FCFS*), a subset of k jobs from the set of all unscheduled jobs. This is the equivalent of the limited-k queue service discipline studied in the polling literature.
- if the queueing approach of Step 1 provides the proportions of time that should be allocated to each class, then select as many unscheduled jobs of a given class as is allowable by the corresponding proportion. The jobs may be selected according to *FCFS* or according to a more complicated rule. For example, we may select the jobs using the *weighted shortest processing time* rule with the "weight" of a job j being $t - r_j$ where t is the end of the current scheduling period and r_j is the arrival time (release date) of job j. This way of defining the weight of a job implies that jobs that have been waiting for processing longer than others will tend to be selected earlier.

(c) Determining when decisions should be made. The decisions can be made at equally-spaced points in time (e.g., every hour), whenever the uncertainty level of the available information becomes low enough or whenever the *anticipation* horizon becomes small enough. The anticipation horizon is defined by Bidot et al. (2009) as the time period between the current time and the time of the last scheduled activity. The approach that reviews the status of the system at equally-spaced points in time does not require an execution-monitoring system, whereas the other two approaches do. Queueing information (e.g., from fluid models) can also be used for determining when the status of the system should be reviewed.

(d) Defining the *commitment horizon*, which includes choosing the set of jobs for which the decisions made will not be reconsidered during execution (Bidot et al., 2009). We can do so using the rules outlined in Step 2(b).

3. Create the sub-problem to be solved at each review point of the progressive scheduling approach by defining the objective and the constraints. The structure of both is dependent on the short-run and long-run performance goals, and on the type of information provided by the queueing approach of Step 1.

Below, we discuss two examples of how this framework can be applied. The first example is from the dissertation by Tran (2011), and is also part of a working journal paper (Terekhov

et al., 2012d) and a conference paper (Tran et al., 2013), and deals with scheduling in a dynamic parallel machine setting.[6] The second one concerns the dynamic job shop with recirculation.

10.3.3 Example 1: Dynamic Parallel Machine Setting

Work in the direction of algorithmic integration that was done concurrently with the work for the present dissertation focused on the dynamic parallel machine scheduling problem (Tran, 2011; Tran et al., 2013; Terekhov et al., 2012d), which can alternatively be referred to as the problem of routing in a queueing network with flexible servers. In this problem, jobs from different classes arrive over time and have to be assigned to one of a set of parallel machines. The processing times of these jobs are dependent both on their class and the machine they are assigned to. As in Chapter 7, it is assumed that the processing times of a job on all machines become known upon its arrival to the system. The goal of the problem is to assign jobs to machines in order to minimize the mean flow time. Tran (2011) empirically evaluates pure scheduling, pure queueing and queueing/scheduling methods for solving this problem without and with setup times.

10.3.3.1 Without Setup Times

Consider the problem discussed above and assume that no setup times are incurred when two jobs of different classes have to be processed on the same machine.

Pure Scheduling Approaches The first scheduling approach considered by Tran (2011) is a greedy heuristic which assigns an arriving job to the machine that would result in the smallest possible flow time for this job given all the jobs already scheduled.

The second method, referred to as *MinMksp*, is a periodic scheduling approach similar to the *makespan* method presented in Section 7.2: the status of the system is reviewed periodically and a static scheduling problem with the makespan objective is solved. However, in this case minimizing the makespan, C_{max}, of each sub-problem requires the solution of a mixed integer program (MIP). If J' denotes the set of jobs belonging to the current sub-problem, and M is the set of parallel machines, then this MIP is stated as follows:

[6] This material is included with permission of the author.

$$\min \quad C_{max} \tag{10.1}$$

$$\text{s.t.} \quad \sum_{j=1}^{|J'|} x_{ij}p_{ij} + v_i \leq C_{max}, \quad i \in M \tag{10.2}$$

$$\sum_{i=1}^{|M|} x_{ij} = 1, \quad j \in J' \tag{10.3}$$

$$x_{ij} \in \{0,1\}, j \in J', i \in M, \tag{10.4}$$

where p_{ij} is the processing time of job j on machine i, and v_i is the remaining processing time on machine i corresponding to jobs from the previous sub-problem. The decision variable in the problem is x_{ij}, which equals 1 if job j is assigned to machine i and 0 otherwise. Once the jobs are assigned based on the solution of this MIP, they are sequenced in *FCFS* order on each machine.

Pure Queueing Approaches The pure queueing approach that Tran (2011) utilizes is due to the paper by Andradóttir et al. (2003), which uses a fluid model-based linear program (LP) to determine the proportion of time, δ_{ik}, that a machine i should spend on a class k so as to maximize the total arrival rate that the system can handle while remaining stable.[7] Based on these proportions, Andradóttir et al. (2003) define a round-robin policy for assigning the jobs to machines, while Al-Azzoni and Down (2008) define an LP-based affinity scheduling (LPAS) heuristic.

Queueing/Scheduling Hybrid The hybrid queueing/scheduling approach proposed by Tran (2011) utilizes both the optimal δ_{ik} values, and the MIP in Equations (10.1)–(10.4). Specifically, the hybrid is a periodic scheduling approach which solves, at each review time point, the following MIP:

$$\min \quad \alpha C_{max} + (1-\alpha)\sum_{i=1}^{|M|}\sum_{k=1}^{|K|} c_{ik} \tag{10.5}$$

$$\text{s.t.} \quad \text{Constraints (10.2) to (10.4)} \tag{10.6}$$

$$\sum_{j \in S_k} x_{ij}p_{ij} - \delta^*_{ik}\sum_{j=1}^{|J'|} x_{ij}p_{ij} \leq c_{ik}, k \in K, i \in M, \tag{10.7}$$

$$c_{ik} \geq 0, k \in K, i \in M, \tag{10.8}$$

[7] See Section 6.1.2.3.1 for a discussion of fluid models and stability, and Chapters 8 and 9 for more detailed examples of the use of fluid models.

where c_{ik} is the deviation between the assignment of class k to machine i and δ^*_{ik}, α is a parameter that represents the importance of deviation from δ^*_{ik} versus the importance of makespan minimization, S_k is the set of jobs belonging to class k and K is the set of all classes. The objective function quantifies the trade-off between what the scheduling algorithm suggests in order to reach the short-run goal of minimizing C_{max} and what the queueing approach suggests in order to ensure stability in the long run. Constraint (10.7) states that c_{ik} is greater than or equal to the difference between the actual amount of time assigned to class k and the amount of time that queueing suggests should be devoted to this class on machine i.

The hybrid algorithm proposed by Tran (2011) fits into the framework of Section 10.3.2 as follows. In Step 1, the fluid model-based LP is chosen as the queueing approach. As mentioned earlier, solving this LP provides the proportions of time that should be spent on each class in order to maximize the arrival rate that can be handled by the system. In Step 2, a periodic approach similar to the one used for scheduling in the dynamic flow shop of Chapter 7 is employed. Tran (2011) defines a scheduling period as the time that elapses from when the scheduler evaluates the system until any of the machines is once again available. Thus, the anticipation horizon is the time window between the earliest time point when one of the machines becomes available and the end time of the last scheduled job from the same sub-problem. The data that is used for reasoning consists of the set of unscheduled jobs and the jobs of the previous sub-problem that are still in progress at the given review point. Decisions are made about all the unscheduled jobs that are present in the system. The reasoning and the commitment horizons are both equal to the temporal window between a given review time point and the completion time of the last scheduled job from the set considered at that review point. In Step 3, the sub-problem is defined via Equations (10.5)–(10.8). In this case, the objective focuses on the balance between minimizing C_{max} and the deviation from the queueing solution, and a constraint is needed to define the deviation.

Tran's experiments showed that the hybrid model performs better than the pure queueing and pure scheduling approaches described above. At heavy loads, the hybrid model finds mean flow times that are up to 50% lower than the next best model. We refer the reader to the dissertation by Tran (2011) and the working paper by Terekhov et al. (2012d) for detailed results.

10.3.3.2 With Setup Times

Tran (2011) and Tran et al. (2013) address the dynamic parallel machine scheduling problem with setup times that are dependent on both the resource and the job sequence. As for the problem without setup times, they compare the round-robin policy developed by Andradóttir et al. (2003) with a scheduling approach based on periodic makespan minimization.

Tran (2011) also proposes four variations of a hybrid approach, referred to as *Tracking*, *Restricted*, *Reschedule* and *Restricted + Reschedule*. The *Tracking* approach is exactly the same as the hybrid approach used in the problem without setups, described in Section 10.3.3.1. The *Restricted* model operates similarly, but allows a job to be assigned to a resource only if the corresponding proportion δ^*_{ik} is non-zero. From the perspective of the framework of Section 10.3.2, the only difference between *Tracking* and *Restricted* is in how the information from the queueing approach of Step 1 is incorporated into the sub-problem developed in Step 3 (i.e., another constraint is added in the *Restricted* case).

The *Reschedule* method redefines the end of a scheduling period to be the earliest of the times that any of the resources become idle or the time when the resource is scheduled to switch to another class. To avoid the situation where a job class would not receive processing for a long time, the maximum number of jobs from class k that can be scheduled on machine i consecutively is set to l_{ik}, which is a parameter of the round-robin policy of Andradóttir et al. (2003). Thus, compared to the *Tracking* hybrid, the *Reschedule* one uses different anticipation and commitment horizons, and has an extra constraint on the number of consecutive jobs from the same class. Another characteristic that distinguishes the *Reschedule* method from all of the hybrids described above is that it uses both the proportions of time generated by the fluid model-based LP and the parameters of a queueing policy based on the same LP. The *Restricted + Reschedule* method is a combination of the *Restricted* and *Reschedule* hybrids.

Experiments showed that all four versions of the hybrid approach perform better than the pure scheduling method. However, the *Tracking*, *Restricted* and *Reschedule* methods are outperformed by the pure queueing approach, the round-robin policy of Andradóttir et al. (2003). Nevertheless, using the *Restricted + Reschedule* approach results in up to a 60% improvement in mean flow time compared to either the pure scheduling or the pure queueing approach. These results demonstrate that queueuing/scheduling hybrids can perform better than pure queueing or pure scheduling methods. However, creating effective hybrid queueing/scheduling methods is non-trivial and requires analysis and experimentation with the specific problem of interest.

10.3.4 Example 2: Dynamic Job Shop with Recirculation

The work of Tran (2011) provides a basis for integrating queueing and scheduling algorithms. However, the problem addressed in that work deals with the assignment of arriving jobs to machines, rather than with sequencing of jobs on a machine,[8] and studies the parallel machine environment, with each job consisting of just one activity (i.e., there are no precedence

[8]Tran (2011) uses *FCFS* to sequence jobs on each machine.

constraints). In order to more fully understand the interaction of queueing and scheduling algorithms, and examine the performance of queueing and scheduling hybrids, we propose an empirical evaluation of these three types of methods in the dynamic job shop with recirculation discussed in Section 10.1.2.2, which involves sequencing decisions and jobs consisting of more than one activity.

Pure Scheduling Approach A periodic scheduling approach similar to the one described in Section 7.2 can be used. Specifically, this approach would be based on reviewing the status of the system at the earliest time point when one of the machines completes a sub-problem.

Pure Queueing Approaches As indicated in Chapter 6, there is a variety of queueing approaches that can be used to obtain scheduling policies. As a first step, we propose to focus on fluid models. For the problem described in Section 10.1.2.2, Atkins and Chen (1995) solve the fluid model-based LP stated in Equations (6.30)–(6.33) of Section 6.1.2.3.1 for each of 2^6 states of the system. The state is defined as a six-dimensional vector, where each entry is either a 0, corresponding to an empty class queue, or a $+$, representng a positive number of jobs in the respective class queue. The solution of each LP is a vector $\mathbf{x} = (x_1, \ldots, x_k)$, where x_k represents the proportion of capacity that station $\sigma(k)$ should devote to processing class k fluids. To create an implementable policy from this fluid model solution, we can consider four approaches:

- The solution can be interpreted as a priority rule. In some cases, doing so is trivial as the solution suggests that all of the server's capacity should be allocated to exactly one class. For example, when there are jobs present in all six classes, the optimal solution of the above-mentioned LP is $\mathbf{x} = (+, 0, +, 0, +, 0)$, where $+$ represents a positive number. This solution suggests that classes 1, 3 and 5 should receive greater priority at stations 1, 2 and 3, respectively, than the other classes those stations are serving (Atkins and Chen, 1995). In fact, by similarly interpreting the solution of the LP for all possible states of the system, the following policy is obtained: at machine 1, class 1 is always given priority over class 2; at machine 3, class 5 always has priority over class 6; and at machine 2, priority is given to class 3 if the queue of class 5 is not empty and to class 4 otherwise (Atkins and Chen, 1995).

  In the general case, the solution of the LP may suggest that some amount of processing time should be spent on more than one class. In this case, priority could be given to the class with the largest proportion or the choice could be made probabilistically.

- The solution can be transformed into a policy using discrete-review trajectory tracking:

a goal state is set and capacity is allocated in such a way as to reach this state (Maglaras, 2000). The status of the system is reviewed periodically. Specifically, at a time point when the decision to process a job has to be taken, the first job from a class that has the greatest current deviation from the fluid solution can be chosen.

- A "roulette-wheel" approach can be used: given the fluid solution specifying the probability of serving each class, the job should be chosen probabilistically at every decision time point (e.g., completion of a job on a machine).

Queueing/Scheduling Hybrids To develop hybrid approaches, we can follow the general framework proposed in Section 10.3.2, using any of the above pure queueing approaches for Step 1 and using the periodic scheduling method for Step 2. In Step 3, we can use a variety of ways for incorporating queueing information, which depend on the choice made in Step 1. Some options include:

- applying a queueing approach to obtain start times for all jobs and then defining, for every job, the deviation between the start time assigned by the queueing solution and the start time suggested by the scheduling approach. This deviation term can then be utilized as part of the objective function in the same way as the class allocation deviations are used by Tran (2011).

- determining the pure queueing and the pure scheduling solutions and then combining the two to form a better schedule for the sub-problem. Various methods from the metaheuristic literature, where there has been work on combining different solutions to derive a new one should be investigated (e.g., path relinking (Glover et al., 2000)).

10.4 Conclusions

As stated in Chapter 2, real scheduling problems are combinatorial, dynamic and uncertain, and related to other decision-making problems. In Part III of this dissertation, we provided a new framework for addressing problems that are combinatorial and dynamic. This framework is based on the integration of classical scheduling and queueing theory on three levels, conceptual, theoretical and algorithmic. The conceptual level is examined in the context of flow shops in Chapter 7, while examples of theoretical integration are given in Chapters 8 and 9. In the current chapter, we discussed ideas for future work on both the conceptual and the theoretical levels. For the algorithmic level, we presented one possible framework for the creation of hybrid queueing/scheduling algorithms, and two examples of the use of this framework. Further

work in the direction of combining queueing and scheduling could be motivated by specific applications, such as the ones arising in healthcare or manufacturing.

In the next part of the dissertation, we propose future work ideas that combine the developments of Parts II and III, summarize the contributions of our work and conclude the dissertation.

Part IV

Conclusion

Chapter 11

Conclusions and Future Work

In this final chapter, we summarize the work presented in previous chapters, re-state the major contributions of this dissertation and state directions for future work that are based on combinations of ideas from Parts II and III.

11.1 Summary and Contributions

Real-world scheduling problems are combinatorial, dynamic and uncertain, and closely related to other decision-making processes of their environment. However, classical scheduling has focused on solving isolated, static, deterministic versions of these problems. In this dissertation, we showed that integration of scheduling with related fields of study provides a means to address scheduling problems under more realistic assumptions. Firstly, by combining scheduling with ideas from inventory management, we were able to more accurately model a specific supply chain setting, and to partially deal with the need for scheduling to take into account related decision-making processes. Secondly, by combining concepts, ideas and methodologies from queueing theory and scheduling, we gained a better understanding of the dynamic aspects of real scheduling problems.

11.1.1 Integration of Scheduling and Inventory Management

Part II of this dissertation addressed, within a small but realistic supply chain setting, the necessity to model the relationship between combinatorial scheduling problems and other decision-making processes, and took a step toward developing a framework for integrating scheduling and inventory decisions.

11.1.1.1 Modelling and Solving a Realistic Supply Chain Scheduling Problem

In Chapter 4, we provided three mixed-integer programming (MIP) models and a constraint programming (CP) model for a small supply chain with two manufacturing facilities that produce distinct products, and a merge-in-transit facility that packages these products together according to customer order specifications. Since manufacturing requires component parts, we explicitly modelled their availability. Specifically, we assumed that a fixed periodic replenishment policy is known and added constraints to our models to ensure that production of an item will not start until all components become available. These constraints model three types of components: those required by a particular product at a particular manufacturer, those shared among multiple products at a particular manufacturer, and those shared between the two manufacturers.

Our models were implemented using commercially available software and extensively empirically evaluated. We therefore provided a baseline on how well the given problem can be solved without resorting to problem-specific algorithms. The empirical evaluation of Chapter 4 showed that, in terms of finding a feasible solution within one hour, the CP model is the best, with the MIP model based on time-indexed variables being a close second. When there are no components shared among the two manufacturers and the processing times are short, this MIP model is the best performer in terms of proving optimality; with a larger range in processing times, the CP model is the best. For problems with shared components, the time-indexed MIP is the best-performing model for proving optimality, although the performance of all models deteriorates significantly, indicating a need, in future work, to investigate more sophisticated approaches.

11.1.1.2 Basis for a General Framework

We created a basis for a general modelling framework for environments where it is necessary to determine the timing and quantity of component replenishments and the schedule of jobs that utilize these components. Models of such environments need to explicitly represent the influence of inventory decisions on scheduling and vice versa. In Part II of this dissertation, we demonstrated how to incorporate the effect of inventory availability on scheduling into mixed-integer programming and constraint programming models. We assumed a fixed replenishment policy that is not dependent on the inventory position and optimized the scheduling objective of total weighted tardiness. In Chapter 10, we outlined further steps in the development of a general framework, including investigation of problems whose goal is to optimize a combination of procurement, holding and weighted tardiness costs or to minimize the inventory costs subject to a constraint ensuring that all jobs are completed on time, under increasingly general

assumptions regarding the inventory policy.

11.1.2 Integration of Scheduling and Queueing Theory

In Part III of this dissertation, we developed a three-level framework for integration of queueing and scheduling, and demonstrated, on two of these levels, the benefits of such integration for dealing with dynamic aspects of real-world scheduling problems. This framework was built on a thorough review of the queueing literature. We concluded Part III with an outline of future work directions for each of the three levels of integration.

11.1.2.1 Review of Queueing Theory Methods for Scheduling

In Chapter 6, we reviewed the queueing theory literature that deals with scheduling problems. Our review classified the queueing work on scheduling into three categories: direct approaches based on Markov decision processes, models possessing special structure, and approximations/abstractions. Throughout the review, we discussed similarities and differences between queueing and scheduling approaches, bridging the gap between two areas that have developed independently. We also indicated specific ways in which the developments of queueing can be useful to scheduling and vice versa, which should be explored in future work. Thus, our review built a strong foundation for integrating queueing theory and scheduling.

11.1.2.2 Conceptual Level of Integration

We demonstrated that integrating queueing and scheduling on the conceptual level leads to novel insights about scheduling in dynamic environments. To investigate this level, we considered two related environments that are dynamic and possess an underlying combinatorial structure. In addition to studying the classical dynamic flow shop, we proposed and investigated a novel scheduling environment: a polling system with a gated, cyclic discipline and a server that is a two-machine flow shop. In both systems we made the classical queueing assumption of knowing the distribution of processing times prior to scheduling, and the classical scheduling assumption that a job's processing time becomes known with certainty upon its arrival to the system.

In both settings, we empirically evaluated the performance of four periodic scheduling approaches for the problem of minimizing mean flow time. Our experiments showed that in the polling system, the method that optimizes the makespan of each sub-problem is the best, while in the dynamic flow shop, an approach based on minimizing the mean flow time directly at each decision point is the best performer. These results demonstrated that in the polling system, there is a conflict between short-run and long-run objectives: minimization

of the makespan leads to sub-optimal sum of completion time values for each sub-problem, but results in significant overall mean flow time improvements. In dynamic two-machine flow shops there is no such conflict: minimizing the total completion time of each sub-problem leads to better long-run mean flow time than does minimizing makespan. These results demonstrated the importance of considering both short-run and long-run objectives in dynamic scheduling.

Subsequently, we formally analyzed the performance differences between the method that periodically optimizes makespan and the method that periodically optimizes the sum of completion times under the assumption that all sub-problems consist of the same number of jobs. We formally showed that the theoretical *completionTime* model always performs better than the *makespan* model with respect to the sum of completion times in the dynamic flow shop. In the polling system, assuming that both models encounter the same sub-problems and some other mild conditions, we showed that the opposite occurs: the *makespan* model outperforms the theoretical *completionTime* model with respect to total completion time as the number of sub-problems goes to infinity. We then stated how the results generalize if the various assumptions made in our analysis are removed.

We observed that the difference in the performance of different methods in the two systems is induced by the change in the review time point. Therefore, we concluded that the design of dynamic scheduling algorithms needs to take into account that changes in both the system structure and the parameters of the approach can have a significant effect on how the trade-off between short-run and long-run objectives affects system performance.

Additionally, we showed that periodic scheduling methods can perform better than queueing-based dispatching rules for optimization of long-run performance. However, it may not always be obvious how to choose the objective function for each sub-problem so that the long-run objective could be met.

11.1.2.3 Theoretical Level of Integration

On the theoretical level of integration, we introduced the notions of stability and fluid limit analysis to the combinatorial scheduling literature. In particular, we analyzed stability of the two-machine flow shop and the polling system with a flow shop server. In the dynamic flow shop, we used a sample path argument to show that a scheduling approach that periodically solves static deterministic sub-problems and optimizes their makespan is stable under the same condition as first-come, first-served (*FCFS*). In the polling system, we proved stability of both *FCFS* and a periodic scheduling method that optimizes the makespan of every sub-problem using the fluid model methodology of Dai (1995). Thus, by proving stability of a scheduling method that is based on the traditional scheduling literature and utilizes processing time information to make sequencing decisions in both systems, we showed that theoretical integra-

tion of queueing and scheduling can lead to long-run performance guarantees for scheduling algorithms that have previously been proved only for queueing policies.

Additionally, we showed that the sample path argument does not extend to a dynamic flow shop with more than two machines. For the polling system, we extended the proofs of stability of *FCFS* and *makespan* to the case when the server is an M-machine flow shop or a d-stage flexible flow shop with M machines at each stage.

Furthermore, we proposed fluid limit analysis as a tool for investigation of the performance of scheduling algorithms. We demonstrated that for policies based on processing time information, the fluid limits of work arriving to machine 2 and work present at machine 2 retain more information about the schedule and the distribution of processing times than those of *FCFS*. Our analysis allowed us to obtain three insights. Firstly, we identified the division of jobs into two sets based on processing times and the order in which these sets are sequenced as being key for optimization of makespan in a two-machine flow shop. Secondly, we made observations regarding the relation between fluid limits and a method's ability to optimize the makespan and the sum of completion times. Finally, for the polling system with a two-machine flow shop server, we established stability of several policies and showed that not all non-idling policies are stable under the stability conditions of *FCFS* and *makespan*.

11.1.2.4 Other Contributions

In Chapter 10 we outlined numerous directions for future work on integrating queueing and scheduling. Most importantly, we introduced one possible framework for the creation of hybrid queueing and scheduling algorithms, establishing a general basis for the algorithmic level of integration. We illustrated this framework in a parallel machine setting from the work by Tran (2011) and in a dynamic job shop with recirculation.

11.2 Future Work

In order to further progress in the development of models of real scheduling problems, in future work it is necessary to study

1. problems with uncertain[1] characteristics as well as complex combinatorics,

2. the relationship between scheduling and decision-making processes other than inventory management,

[1]Recall that we employ a very general definition of the term *uncertainty* in this dissertation. It describes environments that have at least one problem parameter that is not known with certainty at the time of scheduling, and includes cases with both known and unknown distributions for values of these parameters.

3. all major characteristics of real scheduling problems within one framework.

The reader is also referred to Chapter 5 for future work directions stemming from Part II of this dissertation and to Chapter 10 for future work resulting from Part III.

11.2.1 Modelling Uncertainty

In dynamic problems, the environment changes as jobs arrive, are processed and then sent to a different facility or to the customer. The sources of uncertainty include delays in the arrival of component parts, machine breakdowns, adjustments in customer order specifications, and changes in due dates, among others. In addition, in some applications, the processing times of jobs may not be known with certainty prior to their completion. For example, Manitz (2008) states that, from the point of view of the machine, configurable products correspond to stochastic processing times, since each configuration takes a slightly different amount of time to process.

In both Part II and Part III, we assumed exact knowledge of all job parameters at the time of scheduling. Thus, we see as the next step the investigation of the problems we studied in this dissertation under the assumption that some characteristics of the jobs are not known with certainty at the time of the job's arrival but their distributions are available. Assembly-type scheduling environments with stochastic characteristics but without inventory have been considered in the literature. For example, see the work on assembly-like queues (Harrison, 1973), fork-join systems (Ko and Serfozo, 2004), in-forests (Rothblum and Sethuraman, 2008; Liu and Towsley, 1994), parallel bulk-service queues with correlated arrivals (Iravani et al., 2004), synchronized queues (Gallien and Wein, 2001), assembly lines with a synchronization constraint (Manitz, 2008) and stochastic assembly/disassembly systems (Righter, 1997). There has also been work on inventory management in assemble-to-order settings (Doğru et al., 2010). A challenge for future work, then, is to model the effect of inventory availability on scheduling problems with stochastic characteristics. Secondly, it would be interesting to determine whether it would be useful to create periodic scheduling methods that solve a stochastic rather than a deterministic problem at each review time point.

The same problems could be investigated under the assumption that the distribution of job characteristics (e.g., processing times) is not known, but that there is a lower and an upper bound on the parameter values (Sotskov et al., 2010). Work from the artificial intelligence literature focusing on simple temporal networks with uncertainty (Dechter et al., 1991; Cesta and Oddi, 1996; Morris et al., 2001) could be useful in such an investigation as well.

11.2.2 Investigating the Relationship Between Scheduling and Other Decision-Making Processes

In Part II, we made an initial step toward creating models that take into account the relationship between scheduling and inventory. As mentioned in Chapter 2, there are other decision-making processes that have an impact on scheduling, especially in supply chain settings (see Figure 1 of the paper by Moin and Salhi (2006)). In future work, we would therefore like to investigate the relationship between scheduling and other decision-making processes, such as staffing and logistics.

These other decision-making processes are themselves inter-related. For example, there has been a substantial interest in problems that involve both routing and inventory decisions (Dror and Ball, 1987; Moin and Salhi, 2006).[2] In future work, it would be interesting to incorporate scheduling into models that already take into account more than one decision-making process (e.g., integrate scheduling with inventory routing models). In addition, inventory routing work can provide general insights about integrating several decision-making processes within supply chain contexts. For instance, the paper by Dror and Ball (1987) considers both short-run and long-run costs in inventory routing models; it would therefore be interesting to contrast their insights and results with those we obtained in Chapter 7.

11.2.3 Addressing Three Major Characteristics of Scheduling Problems Together

Likely the most unrealistic assumption of the assembly scheduling problem of Chapter 4 is that of considering a fixed set of n orders. Although this might be a reasonable simplification in some cases, a more realistic problem representation would take into account the fact that orders arrive at random points in time and that the number and types of products required are not known until the order's arrival. This is exactly the problem we would like to investigate in future work. Addressing this problem would require combining the work we have presented in Parts II and III of this dissertation, as well as investigating work on fork-join queues (Ko and Serfozo, 2004) and order scheduling in an online setting (Garg et al., 2007). It would also be interesting to study the dynamic version of the assembly scheduling problem in which there could be changes in the inventory replenishment policy over time. These problems would possess three major characteristics of real scheduling problems: complex combinatorics, dynamism and relation to another decision-making process. Thus, investigation of such problems would lead to further progress being made toward the goal of effectively addressing realistic

[2]Thank you to Dr. Michael Trick for bringing the inventory routing work to my attention.

scheduling problems.

11.3 Conclusions

The central thesis of this dissertation is that by combining classical scheduling methodologies with those of inventory management and queueing theory we can better model, understand and solve complex real-world scheduling problems. In the first part of this dissertation, we provided models of a realistic supply chain scheduling problem that captured its combinatorial nature as well as its dependence on inventory availability. We presented an extensive empirical evaluation of how well implementations of these models in commercially available software solve the problem. In the second part of this dissertation, we demonstrated that integrating queueing and scheduling leads to novel insights about, and a better understanding of, problems that have a combinatorial structure and are dynamic. To our knowledge, this dissertation is the first work that builds a framework for integrating queueing theory and scheduling. Motivated by characteristics of real problems, this dissertation took a step toward extending scheduling research beyond traditional assumptions and addressing more realistic scheduling problems.

List of Symbols

Bibliography

Adams, J., E. Balas, D. Zawack. 1988. The shifting bottleneck procedure for job shop scheduling. *Management Science* **34** 391–401.

Adan, I., J. Resing. 2002. *Queueing Theory*. Department of Mathematics and Computing Science, Eindhoven University of Technology. Available online at http://www.win.tue.nl/~iadan/queueing.pdf.

Ahmadi, R., U. Bagchi, T. A. Roemer. 2005. Coordinated scheduling of customer orders for quick response. *Naval Research Logistics* **52** 493–512.

Ahmadi, R. H., U. Bagchi. 1990. Scheduling of multi-job customer orders in multi-machine environments. *ORSA/TIMS* .

Ahn, H. S., I. Duenyas, M. E. Lewis. 2002. Optimal control of a two-stage tandem queuing system with flexible servers. *Probability in the Engineering and Informational Sciences* **16** 453–469.

Akturk, M. S., D. Ozdemir. 2000. An exact approach to minimizing total weighted tardiness with release dates. *IIE Transactions* **32** 1091–1101.

Al-Azzoni, I., D. G. Down. 2008. Linear programming-based affinity scheduling of independent tasks on heterogeneous computing systems. *IEEE Transactions on Parallel and Distributed Systems* **19** 1671–1682.

Allahverdi, A., F. S. Al-Anzi. 2006. A PSO and Tabu search heuristics for the assembly scheduling problem of the two-stage distributed database application. *Computers and Operations Research* **33** 1056–1080.

Altman, E., U. Yechiali. 1993. Cyclic Bernoulli polling. *Mathematical Methods of Operations Research* **38** 55–76.

Andradóttir, S., H. Ayhan. 2005. Throughput maximization for tandem lines with two stations and flexible servers. *Operations Research* **53** 516–531.

Andradóttir, S., H. Ayhan, D. G. Down. 2003. Dynamic server allocation for queueing networks with flexible servers. *Operations Research* **51** 952–968.

Andrews, M., K. Kumaran, K. Ramanan, A. Stolyar, R. Vijayakumar, P. Whiting. 2004. Scheduling in a queueing system with asynchronously varying service rates. *Probability in the Engineering and Informational Sciences* **18** 191–217.

Antunes, N., C. Fricker, J. Roberts. 2011. Stability of multi-server polling system with server limits. *Queueing Systems* **11** 229–235.

Aramon Bajestani, M., J. C. Beck. 2011. Scheduling an aircraft repair shop. *Proceedings of the twenty-first International Conference on Automated Planning and Scheduling (ICAPS'11)*. 10–17.

Aramon Bajestani, M., J. C. Beck. 2013. Scheduling a dynamic aircraft repair shop with limited repair resources. *Journal of Artificial Intelligence Research* To Appear.

Aramon Bajestani, M., J. C. Beck, D. Banjevic. 2012. Integrated maintenance planning and production scheduling under deteriorating machine conditions. Working Paper.

Atkins, D., H. Chen. 1995. Performance evaluation of scheduling control of queueing networks: Fluid model heuristics. *Queueing Systems* **21** 391–413.

Axsäter, S. 2006. *Inventory Control*. 2nd ed. Springer.

Aytug, H., M. Lawley, K. McKay, S. Mohan, R. Uzsoy. 2005. Executing production schedules in the face of uncertainties: A review and future directions. *European Journal of Operational Research* **161** 86–110.

Baccelli, F., Z. Liu. 1990. On the execution of parallel programs on multiprocessor systems – a queuing theory approach. *Journal of the Association for Computing Machinery* **37** 373–414.

Baccelli, F., W. A. Massey, D. Towsley. 1989. Acyclic fork-join queuing networks. *Journal of the Association for Computing Machinery* **36** 615–642.

Baker, K. R., D. Trietsch. 2009. *Principles of Sequencing and Scheduling*. John Wiley & Sons.

Baptiste, P., P. Laborie, C. Le Pape, W. Nuijten. 2006. Constraint-based scheduling and planning. F. Rossi, P. van Beek, T. Walsh, eds., *Handbook of Constraint Programming*, chap. 22. Elsevier, 761–799.

Basnet, C., J. H. Mize. 1994. Scheduling and control of flexible manufacturing systems: a critical review. *International Journal of Computer Integrated Manufacturing* **7** 340–355.

Bäuerle, N. 2000. Asymptotic optimality of tracking policies in stochastic networks. *Annals of Applied Probability* **10** 1065–1083.

Bean, J. C., J. R. Birge, J. Mittenthal, C. E. Noon. 1991. Match-up scheduling with multiple resources, release dates and disruptions. *Operations Research* **39** 470–483.

Beck, J. C. 2002. Heuristics for scheduling with inventory: Dynamic focus via constraint criticality. *Journal of Scheduling* **5** 43–69.

Beck, J. C., A. J. Davenport, E. D. Davis, M. S. Fox. 1998. The ODO project: Toward a unified basis for constraint-directed scheduling. *Journal of Scheduling* **1** 89–125.

Beck, J. C., M. S. Fox. 2000. Dynamic problem structure analysis as a basis for constraint-directed scheduling heuristics. *Artificial Intelligence* **117** 31–81.

Beck, J. C., N. Wilson. 2007. Proactive algorithms for job shop scheduling with probabilistic durations. *Journal of Artificial Intelligence Research* **28** 183–232.

Beekhuizen, P., D. Denteneer, J. Resing. 2008. Reduction of a polling network to a single node. *Queueing Systems* **58** 303–319.

Bent, R., P. Van Hentenryck. 2004. Regrets only! Online stochastic optimization under time constraints. *Proceedings of the Nineteenth National Conference on Artificial Intelligence (AAAI'04)*. 501–506.

Berthold, T., S. Heinz, M. E. Lübbecke, R. H. Möhring, J. Schulz. 2010. A constraint integer programming approach for resource-constrained project scheduling. A. Lodi, M. Milano, P. Toth, eds., *Proceedings of the Seventh International Conference on Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems (CPAIOR'10)*. 313–317.

Bertsekas, D. P. 2005. *Dynamic Programming and Optimal Control: Volume I*. 3rd ed. Athena Scientific.

Bertsekas, D. P. 2007. *Dynamic Programming and Optimal Control: Volume II*. 3rd ed. Athena Scientific.

Bertsimas, D. 1995. The achievable region method in the optimal control of queueing systems: Formulations, bounds, policies. *Queueing Systems* **21** 337–389.

Bertsimas, D., D. Gamarnik. 1999. Asymptotically optimal algorithms for job shop scheduling and packet routing. *Journal of Algorithms* **33** 296–318.

Bertsimas, D., D. Gamarnik, J. Sethuraman. 2003. From fluid relaxations to practical algorithms for high-multiplicity job-shop scheduling: the holding cost objective. *Operations Research* **51** 798–813.

Bertsimas, D., J. Niño-Mora. 1996. Conservation laws, extended polymatroids and multiarmed bandit problems; a polyhedral approach to indexable systems. *Mathematics of Operations Research* **21** 257–306.

Bertsimas, D., I. C. Paschalidis, J. N. Tsitsiklis. 1994. Optimization of multiclass queueing networks: Polyhedral and nonlinear characterizations of achievable performance. *The Annals of Applied Probability* **4** 43–75.

Bertsimas, D., I. C. Paschalidis, J. N. Tsitsiklis. 1995. Branching bandits and Klimov's problem: Achievable region and side constraints. *IEEE Transactions on Automatic Control* **40** 2063–2075.

Bertsimas, D., J. Sethuraman. 2002. From fluid relaxations to practical algorithms for job shop scheduling: the makespan objective. *Mathematical Programming* **92** 61–102.

Beyer, D., F. Cheng, S. P. Sethi, M. Taksar. 2010. *Markovian demand inventory models*, *International Series in Operations Research and Management Science*, vol. 108. Springer Verlag.

Bidot, J. 2005. A general framework integrating techniques for scheduling under uncertainty. Ph.D. thesis, Ecole Nationale d'Ingénieurs de Tarbes.

Bidot, J., T. Vidal, P. Laborie, J. C. Beck. 2009. A theoretic and practical framework for scheduling in a stochastic environment. *Journal of Scheduling* **12** 315–344.

Bierwirth, C., D. C. Mattfeld. 1999. Production scheduling and rescheduling with genetic algorithms. *Evolutionary computation* **7** 1–17.

Blanc, J. P. C., R. D. van der Mei. 1995. Optimization of polling systems with Bernoulli schedules. *Performance Evaluation* **22** 139–158.

Blocher, J. D., D. Chhajed. 2008. Minimizing customer order lead-time in a two-stage assembly supply chain. *Annals of Operations Research* **161** 25–52.

Bock, S., M. Pinedo. 2010. A decomposition scheme for single stage scheduling problems. *Journal of Scheduling* **13** 203–212.

Bookbinder, J. H., J. Y. Tan. 1988. Strategies for the probabilistic lot-sizing problem with service-level constraints. *Management Science* **34** 1096–1108.

Boon, M. A. A., I. J. B. F. Adan. 2009. Mixed gated/exhaustive service in a polling model with priorities. *Queueing Systems* **63** 383–399.

Boon, M. A. A., R. D. Van der Mei, E. M. M. Winands. 2011. Applications of polling systems. *Surveys in Operations Research and Management Science* **16** 67–82.

Borst, S. C. 1995. Polling systems with multiple coupled servers. *Queueing systems* **20** 369–393.

Borst, S. C., O. J. Boxma, J. H. A. Harink, G. B. Huitema. 1994. Optimization of fixed time polling schemes. *Telecommunication Systems* **3** 31–59.

Bose, S. K. 2002. *An Introduction to Queueing Systems*. Springer.

Boudoukh, T., M. Penn, G. Weiss. 2001. Scheduling jobshops with some identical or similar jobs. *Journal of Scheduling* **4** 177–199.

Boxma, O., J. Van Der Wal, U. Yechiali. 2008. Polling with batch service. *Stochastic Models* **24** 604–625.

Boxma, O. J., W. P. Groenendijk. 1987. Pseudo-conservation laws in cyclic-service systems. *Journal of Applied Probability* **24** 949–964.

Boxma, O. J., H. Levy, J. A. Weststrate. 1989. Optimization of polling systems. Tech. Rep. BS-R8932, Department of Operations Research, Statistics, and System Theory, CWI Centre for Mathematics and Computer Science, Amsterdam, The Netherlands.

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

Boysen, N., M. Fliedner, A. Scholl. 2009. Sequencing mixed-model assembly lines: Survey, classification and model critique. *European Journal of Operational Research* **192** 349–373.

Bramson, M. 1994. Instability of FIFO queueing networks. *The Annals of Applied Probability* 414–431.

Bramson, M. 2008. Stability of queueing networks. *Probability Surveys* **5** 169–345.

Branke, J., D. C. Mattfeld. 2002. Anticipatory scheduling for dynamic job shop problems. *Proceedings of the ICAPS'02 Workshop on On-line Planning and Scheduling*. 3–10.

Branke, J., D. C. Mattfeld. 2005. Anticipation and flexibility in dynamic scheduling. *International Journal of Production Research* **43** 3103–3129.

Briskorn, D. 2010. Variable very large neighborhood algorithms for truck sequencing at transshipment terminals. *Working Paper, Instituten für Betriebswirtschaftslehre der Universität Kiel* .

Briskorn, D., B.-C. Choi, K. Lee, J. Leung, M. Pinedo. 2009. Genetic algorithms for inventory constrained scheduling on a single machine. Tech. Rep. 649, Instituten für Betriebswirtschaftslehre der Universität Kiel. Available at http://www.bwl.uni-kiel.de/bwlinstitute/Prod/team/briskorn/ga2009.pdf.

Briskorn, D., B.-C. Choi, K. Lee, J. Leung, M. Pinedo. 2010. Complexity of single machine scheduling subject to nonnegative inventory constraints. *European Journal of Operational Research* **207** 605–619.

Briskorn, D., F. Jaehn, E. Pesch. 2012. Exact algorithms for inventory constrained scheduling on a single machine. *Journal of Scheduling* To Appear, available online at http://www.springerlink.com/content/9w02v25070241840/fulltext.pdf.

Briskorn, D., J. Leung. 2010. Branch and bound algorithms for minimizing maximum lateness of trucks at a transshipment terminal. *Working Paper, Instituten für Betriebswirtschaftslehre der Universität Kiel* .

Browne, S., G. Weiss. 1992. Dynamic priority rules when polling with multiple parallel servers. *Operations Research Letters* **12** 129–137.

Browne, S., U. Yechiali. 1989a. Dynamic priority rules for cyclic-type queues. *Advances in Applied Probability* **21** 432–450.

Browne, S., U. Yechiali. 1989b. Dynamic routing in polling systems. *Teletraffic Science for New Cost-Effective Systems, Networks and Services, ITC-12* 1455–1466.

Brucker, P., A. Drexl, R. Möhring, K. Neumann, E. Pesch. 1999. Resource-constrained project scheduling: Notation, classification, models, and methods. *European Journal of Operational Research* **112** 3–41.

Burns, A. 1991. Scheduling hard real-time systems: a review. *Software Engineering Journal* **6** 116–128.

Buzacott, J. A., J. A. Shanthikumar. 1993. *Stochastic Models of Manufacturing Systems*. Prentice Hall.

Buzacott, J. A., J. G. Shanthikumar. 1985. On approximate queueing models of dynamic job shops. *Management Science* 870–887.

Buzacott, J. A., D. D. Yao. 1986. Flexible manufacturing systems: a review of analytical models. *Management Science* **32** 890–905.

Cadoli, M., F. Patrizi. 2009. On the separability of subproblems in Benders decompositions. *Annals of Operations Research* **171** 27–43.

Cai, X., X. Zhou. 2004. Deterministic and stochastic scheduling with teamwork tasks. *Naval Research Logistics* **51** 818–840.

Cai, X. Q., J. Chen, Y. B. Xiao, X. L. Xu. 2008. Product selection, machine time allocation, and scheduling decisions for manufacturing perishable products subject to a deadline. *Computers and Operations Research* **35** 1671–1683.

Cesta, A., A. Oddi. 1996. Gaining efficiency and flexibility in the simple temporal problem. L. Chittaro, S. Goodwin, H. Hamilton, A. Montanari, eds., *Proceedings of the Third International Workshop on Temporal Representation and Reasoning (TIME-96)*. IEEE Computer Society Press, Los Alamitos, CA.

Chang, H. S., R. Givan, E. K. P. Chong. 2000. On-line scheduling via sampling. *The 5th International Conference on Artificial Intelligence Planning and Scheduling Systems (AIPS'00)* 62–71.

Chao, X., Y. Q. Zhao. 1998. Analysis of multi-server queues with station and server vacations. *European Journal of Operational Research* **110** 392–406.

Chen, F. Y., D. Krass. 2001. Inventory models with minimal service level constraints. *European Journal of Operational Research* **134** 120–140.

Chen, H., P. Yang, D. D. Yao. 1994. Control and scheduling in a two-station queueing network: Optimal policies and heuristics. *Queueing Systems* **18** 301–332.

Chen, H., D. D. Yao. 1993. Dynamic scheduling of a multiclass fluid network. *Operations Research* **41** 1104–1115.

Chen, H., D. D. Yao, eds. 2001. *Fundamentals of Queueing Networks*. Springer.

Chen, K., P. Ji. 2007. A mixed integer programming model for advanced planning and scheduling (APS). *European Journal of Operational Research* **181** 515–522.

Chen, R.-R., S. Meyn. 1999. Value iteration and optimization of multiclass queueing networks. *Queueing Systems* **32** 65–97.

Chrétienne, P., E. G. Coffman Jr., J. K. Lenstra, Z. Liu, eds. 1995. John Wiley & Sons Ltd.

Cicirello, V. A., S. F. Smith. 2005. The max K-armed bandit: A new model of exploration applied to search heuristic selection. *Proceedings of Twentieth National Conference on Artificial Intelligence (AAAI'05)*. 1355–1361.

Coban, E., J. N. Hooker. 2010. Single-facility scheduling over long time horizons by logic-based Benders decomposition. A. Lodi, M. Milano, P. Toth, eds., *Proceedings of the Seventh International Conference on Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems (CPAIOR'10)*. 87–91.

Cohen, M. A., P. R. Kleindorfer, H. L. Lee. 1988. Service constrained *(s, S)* inventory systems with priority demand classes and lost sales. *Management Science* **34** 482–499.

Conway, R. W., W. L. Maxwell, L. W. Miller. 1967. *Theory of Scheduling*. Addison-Wesley.

Cox, D. R., W. L. Smith. 1961. *Queues*. Methuen.

Crabill, T., D. Gross, M. Magazine. 1977. A classified bibliography of research on optimal design and control of queues. *Operations Research* **25** 219–232.

Dacre, M., K. Glazebrook, J. Niño-Mora. 1999. The achievable region approach to the optimal control of stochastic systems. *Journal of the Royal Statistical Society: Series B* **61** 747–791.

Dai, J. G. 1995. On positive harris recurrence of multiclass queueing networks: A unified approach via fluid limit models. *The Annals of Applied Probability* **5** 49–77.

Dai, J. G., W. Lin. 2005. Maximum pressure policies in stochastic processing networks. *Operations Research* **53** 197–218.

Dai, J. G., S. P. Meyn. 1995. Stability and convergence of moments for multiclass queueing networks via fluid limit models. *IEEE Transactions on Automatic Control* **40** 1889–1904.

Dai, J. G., V. Nguyen, M. I. Reiman. 1994. Sequential bottleneck decomposition: an approximation method for generalized Jackson networks. *Operations Research* **42** 119–136.

Dai, J. G., B. Prabhakar. 2000. The throughput of data switches with and without speedup. *Proceedings of the 19th Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM'00)*, vol. 2. IEEE, 556–564.

Dai, J. G., G. Weiss. 1996. Stability and instability of fluid models for reentrant lines. *Mathematics of Operations Research* **21** 115–134.

Dai, J. G., G. Weiss. 2002. A fluid heuristic for minimizing makespan in job shops. *Operations Research* **50** 692–707.

Davenport, A. J., J. C. Beck. 2000. A survey of techniques for scheduling with uncertainty. Tech. rep., University of Toronto. Available at: http://www.tidel.mie.utoronto.ca/publications.php.

Davenport, A. J., C. Gefflot, J. C. Beck. 2001. Slack-based techniques for robust schedules. *Proceedings of the Sixth European Conference on Planning (ECP-2001)*.

Dechter, R., I. Meiri, J. Pearl. 1991. Temporal constraint networks. *Artificial Intelligence* **49** 61–95.

Della Croce, F., V. Narayan, R. Tadei. 1996. The two-machine total completion time flow shop problem. *European Journal of Operational Research* **90** 227–237.

Derman, C. 1970. *Finite State Markovian Decision Processes*. Academic Press.

Doshi, B. 1986. Queueing systems with vacations – a survey. *Queueing Systems* **1** 29–66.

Doğru, M. K., M. I. Reiman, Q. Wang. 2010. A stochastic programming based inventory policy for assemble-to-order systems with application to the W model. *Operations Research* **58** 849–864.

Down, D. 1998. On the stability of polling models with multiple servers. *Journal of Applied Probability* **35** 925–935.

Drekic, S., D. A. Stanford. 2000. Threshold-based interventions to optimize performance in preemptive priority queues. *Queueing Systems* **35** 289–315.

Dror, M., M. Ball. 1987. Inventory/routing: Reduction from an annual to a short-period problem. *Naval Research Logistics* **34** 891–905.

Du, J., J. Y.-T. Leung. 1990. Minimizing total tardiness on one machine is NP-hard. *Mathematics of Operations Research* **15** 483–495.

Duan, L., M. K. Doğru, U. Özen, J. C. Beck. 2012. A negotiation framework for linked combinatorial optimization problems. *Journal of Autonomous Agents and Multi-Agent Systems* **25** 158–182.

Dubois, D., H. Fargier, H. Prade. 1996. Possibility theory in constraint satisfaction problems. *Applied Intelligence* **6** 287–309.

Dudek, R. A., S. S. Panwalkar, M. L. Smith. 1992. The lessons of flowshop scheduling research. *Operations Research* **40** 7–13.

Dumitriu, I., P. Tetali, P. Winkler. 2003. On playing golf with two balls. *SIAM Journal on Discrete Mathematics* **16** 604–615.

El-Bouri, A., S. Balakrishnan, N. Popplewell. 2008. Cooperative dispatching for minimizing mean flowtime in a dynamic flowshop. *International Journal of Production Economics* **113** 819–833.

El Sakkout, H., M. Wallace. 2000. Probe backtrack search for minimal perturbation in dynamic scheduling. *Constraints* **5** 359–388.

Ephremides, A., P. Varaiya, J. Walrand. 1980. A simple dynamic routing problem. *Automatic Control, IEEE Transactions on* **25** 690–693.

Fan-Orzechowski, X., E. A. Feinberg. 2007. Optimality of randomized trunk reservation for a problem with multiple constraints. *Probability in the Engineering and Informational Sciences* **21** 189–200.

Fanti, M. P., G. Stecco, W. Ukovich. 2010. Scheduling the internal operations in distribution centers with buffer constraints. *6th Annual IEEE Conference on Automation Science and Engineering (CASE)*. IEEE, 75–80.

Federgruen, A., H. Groenevelt. 1988. Characterization and optimization of achievable performance in general queueing systems. *Operations Research* **36** 733–741.

Fournier, L., Z. Rosberg. 1991. Expected waiting times in polling systems under priority disciplines. *Queueing Systems* **9** 419–440.

Fox, M. S. 1983. Constraint-directed search: A case study of job-shop scheduling. Ph.D. thesis, Carnegie Mellon University, Intelligent Systems Laboratory, The Robotics Institute, Pittsburgh, PA. CMU-RI-TR-85-7.

Framinan, J. M., R. Leisten, R. Ruiz-Usano. 2005. Comparison of heuristics for flowtime minimisation in permutation flowshops. *Computers and Operations Research* **32** 1237–1254.

Fromherz, M. P. J. 2001. Constraint-based scheduling. *Proceedings of the 2001 American Control Conference (ACC'01)*, vol. 4. IEEE, 3231–3244.

Fuhrmann, S. W. 1984. A note on the M/G/1 queue with server vacations. *Operations Research* **32** 1368–1373.

Gagliolo, M., J. Schmidhuber. 2007. Learning restart strategies. *Proceedings of the of the Twenty First International Joint Conference on Artificial Intelligence (IJCAI'07)*. 792–797.

Gallien, J., L. M. Wein. 2001. A simple and effective component procurement policy for stochastic assembly systems. *Queueing Systems* **38** 221–248.

Garg, N., A. Kumar, V. Pandit. 2007. Order scheduling models: Hardness and algorithms. *FSTTCS 2007*. Springer-Verlag, 96–107.

Gaujal, B., A. Hordijk, D. van der Laan. 2007. On the optimal open-loop control policy for deterministic and exponential polling systems. *Probability in the Engineering and Informational Sciences* **21** 157–187.

Ghallab, M., D. Nau, P. Traverso. 2004. *Automated Planning: Theory and Practice*. Morgan Kaufman.

Gittins, J. C. 1979. Bandit processes and dynamic allocation indices. *Journal of the Royal Statistical Society: Series B (Methodological)* **41** 148–177.

Glasserman, P., D. D. Yao. 1994. Monotone optimal control of permutable GSMPs. *Mathematics of Operations Research* **19** 449–476.

Glazebrook, K. D., R. W. Owen. 1995. Gittins-index heuristics for research planning. *Naval Research Logistics* **42** 1041–1062.

Glover, F., M. Laguna, R. Martí. 2000. Fundamentals of scatter search and path relinking. *Control and Cybernetics* **29** 653–684.

Goldberg, H. M. 1977. Analysis of the earliest due date scheduling rule in queueing systems. *Mathematics of Operations Research* **2** 145–154.

Gourgand, M., N. Grangeon, S. Norre. 2005. Markovian analysis for performance evaluation and scheduling in m machine stochastic flow-shop with buffers of any capacity. *European Journal of Operational Research* **161** 126–147.

Govil, M. K., M. C. Fu. 1999. Queueing theory in manufacturing: A survey. *Journal of Manufacturing Systems* **18** 214–240.

Graham, R. L., D. E. Knuth, O. Patashnik. 1988. *Concrete Mathematics: A Foundation for Computer Science*. 1st ed. Addison-Wesley Publishing Co.

Graham, R. L., E. L. Lawler, J. K. Lenstra, A. H. G. Rinnooy Kan. 1979. Optimization and approximation in deterministic sequencing and scheduling: a survey. *Annals of Discrete Mathematics* **5** 287–326.

Grassmann, W., X. Chen, B. R. K. Kashyap. 2001. Optimal service rates for the state-dependent M/G/1 queues in steady-state. *Operations Research Letters* **29** 57–63.

Grassmann, W. K. 1977. Transient solutions in Markovian queueing systems. *Computers & Operations Research* **4** 47–53.

Grigoriev, A., M. Holthuijsen, J. van de Klundert. 2005. Basic scheduling problems with raw material constraints. *Naval Research Logistics* **52** 527–535.

Gross, D., C. Harris. 1998. *Fundamentals of Queueing Theory*. John Wiley & Sons, Inc.

Gupta, A. K., A. I. Sivakumar. 2006. Job shop scheduling techniques in semiconductor manufacturing. *The International Journal of Advanced Manufacturing Technology* **27** 1163–1169.

Gupta, J. N. D., E. F. Stafford Jr. 2006. Flowshop scheduling research after five decades. *European Journal of Operational Research* **169** 699–711.

Gurvich, I., W. Whitt. 2009. Scheduling flexible servers with convex delay costs in many-server service systems. *Manufacturing & Service Operations Management* **11** 237–253.

Hajek, B. 1984. Optimal control of two interacting service stations. *IEEE Transactions on Automatic Control* **29** 491–499.

Hall, L. A. 1997. Approximation algorithms for scheduling. D. S. Hochbaum, ed., *Approximation algorithms for NP-hard problems*, chap. 1. PWS Publishing Company, 1–45.

Harchol-Balter, M. 2011. Queueing disciplines. *Wiley Encyclopedia of Operations Research and Management Science* .

Hariri, A. M. A., C. N. Potts. 1997. A branch and bound algorithm for the two-stage assembly scheduling problem. *European Journal of Operational Research* **103** 547–556.

Harrison, J. M. 1973. Assembly-like queues. *Journal of Applied Probability* **10** 354–367.

Harrison, J. M. 1975. Dynamic scheduling of a multiclass queue: Discount optimality. *Operations Research* **23** 270–282.

Harrison, J. M. 1988. Brownian models of queueing networks with heterogeneous customer populations. W. Fleming, P. L. Lions, eds., *Stochastic Differential Systems, Stochastic Control Theory and Applications*. Springer-Verlag, 147–186.

Harrison, J. M. 1996. The BIGSTEP approach to flow management in stochastic processing networks. F. P. Kelly, S. Zachary, I. Ziedins, eds., *Stochastic Networks: Theory and Applications*, chap. 4. Oxford University Press, 57–90.

Harrison, J. M. 1998. Heavy traffic analysis of a system with parallel servers: Asymptotic optimality of discrete-review policies. *The Annals of Applied Probability* **8** 822–848.

Harrison, J. M. 2003. A broader view of Brownian networks. *The Annals of Applied Probability* **13** 1119–1150.

Harrison, J. M., V. Nguyen. 1993. Brownian models of multiclass queueing networks: current status and open problems. *Queueing Systems* **13** 5–40.

Harrison, J. M., L. M. Wein. 1989. Scheduling networks of queues: Heavy traffic analysis of a simple open network. *Queueing Systems* **5** 265–280.

Hartmann, S., D. Briskorn. 2010. A survey of variants and extensions of the resource-constrained project scheduling problem. *European Journal of Operational Research* **207** 1–14.

Hassin, R., J. Puerto, F. R. Fernández. 2009. The use of relative priorities in optimizing the performance of a queueing system. *European Journal of Operational Research* **193** 476–483.

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

Hejazi, S. R., S. Saghafian. 2005. Flowshop-scheduling problems with makespan criterion: a review. *International Journal of Production Research* **43** 2895–2929.

Herroelen, W., B. De Reyck, E. Demeulemeester. 1998. Resource-constrained project scheduling: a survey of recent developments. *Computers and Operations Research* **25** 279–302.

Herroelen, W., R. Leus. 2005. Project scheduling under uncertainty: Survey and research potentials. *European Journal of Operational Research* **165** 289–306.

Hofri, M., Z. Rosberg. 1987. Packet delay under the golden ratio weighted TDM policy in a multiple-access channel. *IEEE Transactions on Information Theory* **33** 341–349.

Hooker, J. N. 2005. A Hybrid Method for Planning and Scheduling. *Constraints* **10** 385–401.

Hooker, J. N., G. Ottosson. 2003. Logic-based Benders' decomposition. *Mathematical Programming* **96** 33–60.

Huang, H. L., B. M. T. Lin. 2007. Concurrent openshop problem to minimize the weighted number of late jobs. Eugene Levner, ed., *Multiprocessor Scheduling, Theory and Applications*, chap. 12. InTech, 215.

Iravani, S. M. R., K. L. Luangkesorn, D. Simchi-Levi. 2004. A general decomposition algorithm for parallel queues with correlated arrivals. *Queueing Systems* **47** 313–344.

Jackson, J. R. 1963. Jobshop-like queueing systems. *Management Science* **10** 131–142.

Jaiswal, N. K. 1968. *Priority Queues*. Academic Press.

Jaiswal, N. K. 1982. Performance evaluation studies for time-sharing computer systems. *Performance Evaluation* **2** 223–236.

Johri, P. K., M. N. Katehakis. 1988. Scheduling service in tandem queues attended by a single server. *Stochastic Analysis and Applications* **6** 279–288.

Kaczynski, W. H., L. M. Leemis, J. H. Drew. 2011. Transient queueing analysis. *INFORMS Journal on Computing* To Appear.

Kang, W., F. P. Kelly, N. H. Lee, R. J. Williams. 2004. Fluid and Brownian approximations for an internet congestion control model. *43rd IEEE Conference on Decision and Control*, vol. 4. 3938–3943.

Kao, E. P. C., K. S. Narayanan. 1991. Analyses of an M/M/N queue with servers' vacations. *European Journal of Operational Research* **54** 256–266.

Katayama, T. 1992. Performance analysis and optimization of a cyclic-service tandem queueing system with multi-class customers. *Computers and Mathematics with Applications* **24** 25–33.

Katta, A.-K., J. Sethuraman. 2005. A note on bandits with a twist. *SIAM Journal on Discrete Mathematics* **18** 110–113.

Keha, A. B., K. Khowala, J. W. Fowler. 2009. Mixed integer programming formulations for single machine scheduling problems. *Computers & Industrial Engineering* **56** 357–367.

Kelly, F. P., C. N. Laws. 1993. Dynamic routing in open queueing networks: Brownian models, cut constraints and resource pooling. *Queueing Systems* **13** 47–86.

Kelly, F. P., S. Zachary, I. Ziedins, eds. 1996. *Stochastic Networks: Theory and Applications*. Oxford University Press.

Khamisy, A., E. Altman, M. Sidi. 1992. Polling systems with synchronization constraints. *Annals of Operations Research* **35** 231–267.

Khintchine, A. Y. 1932. Mathematical theory of stationary queues. *Matematicheskii Sbornik* **39** 73–84. In Russian.

Kim, Y. D. 1993. A new branch and bound algorithm for minimizing mean tardiness in two-machine flowshops. *Computers and Operations Research* **20** 391–401.

Kitaev, M. Y., V. V. Rykov. 1995. *Controlled Queueing Systems*. 1st ed. CRC-Press.

Kleinrock, L. 1976. *Queueing Systems, Volume 1 and 2*. Wiley.

Klimov, G. P. 1975. Time-sharing service systems I. *Theory of Probability and its Applications* **19** 532–551. Translated from Russian.

Ko, S.-S., R. F. Serfozo. 2004. Response times in M/M/s fork-join networks. *Advances in Applied Probability* **36** 854–871.

Kolisch, R. 2000. Integrated scheduling, assembly area- and part-assignment for large-scale, make-to-order assemblies. *International Journal of Production Economics* **64** 127–142.

Kolisch, R., K. Hess. 2000. Efficient methods for scheduling make-to-order assemblies under resource, assembly area and part availability constraints. *International Journal of Production Research* **38** 207–228.

Kovács, A., J. C. Beck. 2011. A global constraint for total weighted completion time for unary resources. *Constraints* **16** 100–123.

Kubiak, W. 2004. Fair sequences. J. Y.-T. Leung, ed., *Handbook of Scheduling: Algorithms, Models and Performance Analysis*, chap. 19. CRC Press, 19–1–19–21.

Kumar, P. R. 1994. Scheduling semiconductor manufacturing plants. *IEEE Control Systems Magazine* **14** 33–40.

Kumar, P. R., S. P. Meyn. 1995. Stability of queueing networks and scheduling policies. *IEEE Transactions on Automatic Control* **40** 251–260.

Kumar, V. 1992. Algorithms for constraint satisfaction problems: A survey. *AI Magazine* **13** 32–44.

Lee, C.-Y., T. C. E. Cheng, B. M. T. Lin. 1993. Minimizing the makespan in the 3-machine assembly-type flowshop scheduling problem. *Management Science* **39** 616–625.

Lehoczky, J. P. 1996. Real-time queueing theory. *Proceedings of the 17th IEEE Real-Time Systems Symposium*. 186–195.

Lenstra, J. K., D. B. Shmoys. 1995. Computing near-optimal schedules. P. Chrétienne, E. G. Coffman Jr., J. K. Lenstra, Z. Liu, eds., *Scheduling Theory and Its Applications*, chap. 1. John Wiley & Sons Ltd, 1–14.

Leon, V. J., S. D. Wu, R. H. Storer. 1994. Robustness measures and robust scheduling for job shops. *IIE Transactions* **26** 32–43.

Leung, J. Y.-T., ed. 2004. *Handbook of Scheduling: Algorithms, Models, and Performance Analysis*. CRC Press.

Leung, J. Y.-T., H. Li, M. Pinedo. 2005a. Order scheduling in an environment with dedicated resources in parallel. *Journal of Scheduling* **8** 355–386.

Leung, J. Y.-T., H. Li, M. Pinedo. 2005b. Order scheduling models: An overview. G. Kendall, E. K. Burke, S. Petrovic, M. Gendreau, eds., *Multidisciplinary Scheduling: Theory and Applications*. Springer, 37–53.

Leung, J. Y.-T., H. Li, M. Pinedo. 2007. Scheduling orders for multiple product types to minimize total weighted completion time. *Discrete Applied Mathematics* **155** 945–970.

Leung, J. Y.-T., H. Li, M. Pinedo, C. Sriskandarajah. 2005c. Open shops with jobs overlap–revisited. *European Journal of Operational Research* **163** 569–571.

Levy, H. 1991. Binomial-gated service: A method for effective operation and optimization of polling systems. *IEEE Transactions on Communications* **39** 1341–1350.

Levy, H., M. Sidi. 1990. Polling systems: Applications, modeling, and optimization. *IEEE Transactions on Communications* **38** 150–1760.

Li, C.-L., G. Vairaktarakis. 2007. Coordinating production and distribution of jobs with bundling operations. *IIE Transactions* **39** 203–215.

Li, Z., M. Ierapetritou. 2008. Process scheduling under uncertainty: Review and challenges. *Computers & Chemical Engineering* **32** 715–727.

Lin, B. M. T., A. V. Kononov. 2007. Customer order scheduling to minimize the number of late jobs. *European Journal of Operational Research* **183** 944–948.

Liu, Z., D. Towsley. 1994. Stochastic scheduling in in-forest networks. *Advances in Applied Probability* **26** 222–241.

Lu, S. C. H., D. Ramaswamy, P. R. Kumar. 1994. Efficient scheduling policies to reduce mean and variance of cycle-time in semiconductor manufacturing plants. *IEEE Transactions on Semiconductor Manufacturing* **7** 374–388.

Lu, S. H., P. R. Kumar. 1991. Distributed scheduling based on due dates and buffer priorities. *IEEE Transactions on Automatic Control* **36** 1406–1416.

Maglaras, C. 1999. Dynamic scheduling in multiclass queueing networks: Stability under discrete-review policies. *Queueing Systems* **31** 171–206.

Maglaras, C. 2000. Discrete-review policies for scheduling stochastic networks: Trajectory tracking and fluid-scale asymptotic optimality. *The Annals of Applied Probability* **10** 897–929.

Maglaras, C. 2003. Continuous-review tracking policies for dynamic control of stochastic networks. *Queueing Systems* **43** 43–80.

Mahajan, A., D. Teneketzis. 2007. Multi-armed bandit problems. A. O. Hero III, D. A. Castañón, D. Cochran, K. Kastella, eds., *Foundations and Applications of Sensor Management*. Springer, 121–151.

Manitz, M. 2008. Queueing-model based analysis of assembly lines with finite buffers and general service times. *Computers and Operations Research* **35** 2520–2536.

Maravelias, C. T., C. Sung. 2009. Integration of production planning and scheduling: Overview, challenges and opportunities. *Computers & Chemical Engineering* **33** 1919–1930.

Martins, L. F., S. E. Shreve, H. M. Soner. 1996. Heavy traffic convergence of a controlled, multiclass queueing system. *SIAM Journal on Control and Optimization* **34** 2133–2171.

Masin, M., M. O. Pasaogullari, S. Joshi. 2007. Dynamic scheduling of production-assembly networks in a distributed environment. *IIE Transactions* **39** 395–409.

Massey, W. A., W. Whitt. 1998. Uniform acceleration expansions for Markov chains with time-varying rates. *Annals of Applied Probability* **8** 1130–1155.

Mastrolilli, M., M. Queyranne, A. S. Schulz, O. Svensson, N. A. Uhan. 2010. Minimizing the sum of weighted completion times in a concurrent open shop. *Operations Research Letters* **38** 390–395.

McKoy, D. H. C., P. J. Egbelu. 1998. Minimizing production flow time in a process and assembly job shop. *International Journal of Production Research* **36** 2315–2332.

Meilijson, I., U. Yechiali. 1977. On optimal right-of-way policies at a single-server station when insertion of idle times is permitted. *Stochastic Processes and Their Applications* **6** 25–32.

Mercier, L., P. Van Hentenryck. 2008. Amsaa: A multistep anticipatory algorithm for online stochastic combinatorial optimization. *Proceedings of the 5th International Conference on Integration of AI and OR techniques in constraint programming for combinatorial optimization problems (CPAIOR'08)*. 173–187.

Meyn, S. P. 2001. Sequencing and routing in multiclass queueing networks. Part I: Feedback regulation. *SIAM Journal on Control and Optimization* **40** 741–776.

Meyn, S. P. 2008. *Control Techniques for Complex Networks*. Cambridge University Press.

Moin, N. H., S. Salhi. 2006. Inventory routing problems: a logistical overview. *Journal of the Operational Research Society* **58** 1185–1194.

Monahan, G. E. 1982. A survey of partially observable Markov decision processes: Theory, models, and algorithms. *Management Science* **28** 1–16.

Montana, D. 2005. A comparison of combinatorial optimization and dispatch rules for online scheduling. *Proceedings of the 2nd Multidisciplinary International Conference on Scheduling: Theory and Applications (MISTA'05)*. Citeseer, 353–362.

Morris, P., N. Muscettola, T. Vidal. 2001. Dynamic control of plans with temporal uncertainty. *Proceedings of the Seventeenth International Joint Conference on Artificial Intelligence (IJCAI'01)*.

Myers, K., P. Berry, J. Blythe, K. Conley, M. Gervasio, D. McGuinness, D. Morley, Pfeffer A., M. Pollack, M. Tambe. 2007. An intelligent personal assistant for task and time management. *AI Magazine* **28** 47–61.

Nahmias, S., ed. 1993. *Production and Operations Management*. Irwin.

Nash, P., R. R. Weber. 1982. Sequential open-loop scheduling strategies. M. A. H. Dempster, Lenstra J. K., A. H. G. Rinnooy Kan, eds., *Deterministic and Stochastic Scheduling: proceedings of the NATO Advanced Study and Research Institute on Theoretical Approaches to Scheduling Problems*. D. Reidel Publishing Company, 385–397. Available online at http://www.statslab.cam.ac.uk/˜rrw1/publications/Nash%20-%20Weber%201982%20Sequential%20open-loop%20scheduling%20strategies.pdf.

Nazarathy, Y., G. Weiss. 2009. Near optimal control of queueing networks over a finite time horizon. *Annals of Operations Research* **170** 233–249.

Nazarathy, Y., G. Weiss. 2010. A fluid approach to large volume job shop scheduling. *Journal of Scheduling* **13** 509–529.

Neumann, K., C. Schwindt. 2002. Project scheduling with inventory constraints. *Mathematical Methods of Operations Research* **56** 513–533.

Niño-Mora, J. 2007. Dynamic priority allocation via restless bandit marginal productivity indices. *Top* **15** 161–198.

Nuyens, M., A. Wierman, A. P. Zwart. 2008. Preventing large sojourn times with SMART scheduling. *Operations Research* **56** 88–101.

Oliver, R. M., G. Pestalozzi. 1965. On a problem of optimum priority classification. *Journal of the Society for Industrial and Applied Mathematics* **13** 890–901.

Ouelhadj, D., S. Petrovic. 2009. A survey of dynamic scheduling in manufacturing systems. *Journal of Scheduling* **12** 417–431.

Özdamar, L., G. Ulusoy. 1995. A survey on the resource-constrained project scheduling problem. *IIE Transactions* **27** 574–586.

Panwalkar, S. S., W. Iskander. 1977. A survey of scheduling rules. *Operations Research* **25** 45–61.

Pardo, M. J., D. de la Fuente. 2007. Optimizing a priority-discipline queueing model using fuzzy set theory. *Computers and Mathematics with Applications* **54** 267–281.

Park, Y. B. 1988. An evaluation of static flowshop scheduling heuristics in dynamic flowshop models via a computer simulation. *Computers & Industrial Engineering* **14** 103–112.

Park, Y. B., C. D. Pegden, E. E. Enscore. 1984. A survey and evaluation of static flowshop scheduling heuristics. *The International Journal of Production Research* **22** 127–141.

Paschalidis, I. C., C. Su, M. C. Caramanis. 2003. Target-pursuing policies for open multiclass queueing networks. *Proceedings of the Twenty-Second Annual Joint Conference of the IEEE Computer and Communications (INFOCOM'03)*, vol. 1. 196–206.

Paschalidis, I. C., C. Su, M. C. Caramanis. 2004. Target-pursuing scheduling and routing policies for multiclass queueing networks. *IEEE Transactions on Automatic Control* **49** 1709–1722.

Pathumnakul, S., P. J. Egbelu. 2006. An algorithm for minimizing weighted earliness penalty in assembly job shops. *International Journal of Production Economics* **103** 230–245.

Patrick, J., M. L. Puterman, M. Queyranne. 2008. Dynamic multipriority patient scheduling for a diagnostic resource. *Operations Research* **56** 1507–1525.

Pegden, C. D., M. Rosenshine. 1990. Scheduling arrivals to queues. *Computers and Operations Research* **17** 343–348.

Philipoom, P. R., R. S. Russell, T. D. Fry. 1991. A preliminary investigation of multi-attribute based sequencing rules for assembly shops. *International Journal of Production Research* **29** 739–753.

Phipps Jr., T. E. 1956. Machine repair as a priority waiting-line problem. *Operations Research* **4** 76–85.

Pinedo, M. L. 2003. *Scheduling: Theory, Algorithms, and Systems*. 2nd ed. Prentice-Hall.

Pinedo, M. L. 2009. *Planning and Scheduling in Manufacturing and Services*. Springer.

Pirkwieser, S. 2006. A Lagrangian decomposition approach combined with metaheuristics for the knapsack constrained maximum spanning tree problem. Master's thesis, Institute of Computer Graphics and Algorithms, Vienna University of Technology.

Pollaczek, F. 1932. Lösung eines geometrischen wahrscheinlichkeitsproblems. *Mathematische Zeitschrift* **35** 230–278. In German.

Potts, C. N., S. V. Sevast'janov, V. A. Strusevich, L. N. Van Wassenhove, C. M. Zwaneveld. 1995. The two-stage assembly scheduling problem: complexity and approximation. *Operations Research* **43** 346–355.

Pruhs, K. 2007. Competitive online scheduling for server systems. *ACM SIGMETRICS Performance Evaluation Review* **34** 52–58.

Pruhs, K., J. Sgall, E. Torng. 2004. Online scheduling. J. Y.-T. Leung, ed., *Handbook of Scheduling: Algorithms, Models and Performance Analysis*, chap. 15. CRC Press, 15–1 – 15–43.

Puterman, M. L. 1994. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. John Wiley & Sons, Inc.

Radlinski, F., R. Kleinberg, J. Thorsten. 2005. Learning diverse rankings with multi-armed bandits. *Proceedings of Twenty Fifth International Conference on Machine Learning (ICML'08)*. 1355–1361.

Ravikumar, K., Y. Narahari. 1994. Dynamic scheduling in manufacturing systems using Brownian approximations. *Sadhana (Academy Proceedings in Engineering Sciences)* **19** 891–939.

Raviv, T. 2003. Fluid approximation and other methods for hard combinatorial optimization problems. Ph.D. thesis, Technion-Israel Institute of Technology, Haifa, Israel.

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

Reiman, M. I., L. M. Wein. 1999. Heavy traffic analysis of polling systems in tandem. *Operations Research* **47** 524–534.

Righter, R. 1994. Scheduling. M. Shaked, J. G. Shanthikumar, eds., *Stochastic Orders and Their Applications*. Academic Press, 381–432.

Righter, R. 1997. A generalized Johnson's rule for stochastic assembly systems. *Naval Research Logistics* **44** 211–220.

Robinson, D. 1978. Optimization of priority queues – a semi-Markov decision chain approach. *Management Science* **24** 545–553.

Roemer, T. 2006. A note on the complexity of the concurrent open shop problem. *Journal of Scheduling* **9** 389–396.

Rosa-Hatko, W., E. A. Gunn. 1997. Queues with switchover – a review and critique. *Annals of Operations Research* **69** 299–322.

Ross, S. M. 2003. *Introduction to Probability Models*, chap. 6 – Continuous-Time Markov Chains. Academic Press, 349–399.

Rothblum, U. G., J. Sethuraman. 2008. Stochastic scheduling in an in-forest. *Discrete Optimization* **5** 457–466.

Rothkopf, M. H. 1966. Scheduling with random service times. *Management Science* **12** 707–713.

Ruschitzka, M., R. S. Fabry. 1977. A unifying approach to scheduling. *Communications of the ACM* **20** 469–477.

Sabucuoglu, I., M. Bayiz. 2000. Analysis of reactive scheduling problems in a job-shop environment. *European Journal of Operational Research* **126** 567–586.

Sarper, H., M. C. Henry. 1996. Combinatorial evaluation of six dispatching rules in a dynamic two-machine flow shop. *Omega* **24** 73–81.

Scheduler. 2009. *IBM ILOG Scheduler 6.7 User's Manual and Reference Manual*. IBM ILOG.

Schrage, L. 1968. A proof of the optimality of the shortest remaining processing time discipline. *Operations Research* **16** 687–690.

Schrage, L. 1970. An alternative proof of a conservation law for the queue G/G/1. *Operations Research* **18** 185–187.

Schuurman, P., G. J. Woeginger. 1999. Polynomial time approximation algorithms for machine scheduling: Ten open problems. *Journal of Scheduling* **2** 203–213.

Seidman, T. I. 1994. "First come, first served" can be unstable! *IEEE Transactions on Automatic Control* **39** 2166–2171.

Sennott, L. I. 1999. *Stochastic Dynamic Programming and the Control of Queueing Systems*. John Wiley & Sons, Inc.

Sha, L., T. Abdelzaher, K.-E. Årzén, A. Cervin, T. Baker, A. Burns, G. Buttazzo, M. Caccamo, J. Lehoczky, A. K. Mok. 2004. Real time scheduling theory: A historical perspective. *Real-time systems* **28** 101–155.

Shanthikumar, J. G. 1982. On reducing time spent in M/G/1 systems. *European Journal of Operational Research* **9** 286–294.

Shanthikumar, J. G., S. Ding, M. T. Zhang. 2007. Queueing theory for semiconductor manufacturing systems: A survey and open problems. *IEEE Transactions on Automation Science and Engineering* **4** 513–522.

Sharafali, M., H. C. Co, M. Goh. 2004. Production scheduling in a flexible manufacturing system under random demand. *European Journal of Operational Research* **158** 89–102.

Silver, E. A., D. F. Pyke, R. Peterson. 1998. *Inventory Management and Production Planning and Scheduling*. Wiley.

Skinner, W. 1974. The focused factory. *Harvard Business Review* **52** 113–121.

Smith, D. R. 1978. A new proof of the optimality of the shortest remaining processing time discipline. *Operations Research* **26** 197–199.

Smith, S. F. 1994. OPIS: A methodology and architecture for reactive scheduling. M. Zweben, M. S. Fox, eds., *Intelligent Scheduling*, chap. 2. Morgan Kaufmann Publishers, San Francisco, 29–66.

Smith, W. E. 1956. Various optimizers for single-stage production. *Naval Research Logistics Quarterly* **3** 59–66.

Sotskov, Yu. N., N. Yu. Sotskova, T.-C. Lai, F. Werner. 2010. *Scheduling Under Uncertainty: Theory and Algorithms*. Belorussian Science. Available from http://www.math.uni-magdeburg.de/w̃erner/sot-sot-lai-werner.pdf.

Souza, G. C., H. M. Wagner, D. C. Whybark. 2001. Evaluating focused factory benefits with queuing theory. *European Journal of Operational Research* **128** 597–610.

Sparaggis, P. D., C. G. Cassandras, D. Towsley. 1993. On the duality between routing and scheduling systems with finite buffer space. *IEEE Transactions on Automatic Control* **38** 1440–1446.

Stewart, J. 1999. *Calculus*. Brooks/Cole Publishing Company.

Stidham, Jr. S. 1985. Optimal Control of Admission to a Queueing System. *IEEE Transactions on Automatic Control* **AC-30** 705–713.

Stidham, Jr. S., R. Weber. 1993. A Survey of Markov decision models for control of networks of queues. *Queueing Systems* **13** 291–314.

Stidham, S. S. 2009. *Optimal Design of Queueing Systems*. CRC-Press.

Stidham Jr., S. 2002. Analysis, design, and control of queueing systems. *Operations Research* **50** 197–216.

Stolyar, A. L. 2004. Maxweight scheduling in a generalized switch: State space collapse and workload minimization in heavy traffic. *The Annals of Applied Probability* **14** 1–53.

Streeter, M. 2007. Using online algorithms to solve NP-hard problems more efficiently in practice. Ph.D. thesis, Carnegie Mellon University, Pittsburgh, PA.

Sun, X., K. Morizawa, H. Nagasawa. 2003. Powerful heuristics to minimize makespan in fixed, 3-machine, assembly-type flowshop scheduling. *European Journal of Operational Research* **146** 498–516.

Sung, C. S., S. H. Yoon. 1998. Minimizing total weighted completion time at a pre-assembly stage composed of two feeding machines. *International Journal of Production Economics* **54** 247–255.

Suresh, V., D. Chaudhuri. 1993. Dynamic scheduling – a survey of research. *International Journal of Production Economics* **32** 53–63.

Tadj, L., G. Choudhury. 2005. Optimal design and control of queues. *Sociedad de Estadìstica e Investigación Operativa, Top* **13** 359–412.

Takagi, H. 1986. *Analysis of Polling Systems*. MIT Press.

Takagi, H. 1988. Queueing analysis of polling models. *ACM Computing Surveys* **20** 5–28.

Takagi, H. 2000. Analysis and application of polling models. *Performance Evaluation: Origins and Directions*. Lecture Notes in Computer Science, Springer, 423–442.

Tassiulas, L., P. P. Bhattacharya. 2000. Allocation of interdependent resources for maximal throughput. *Stochastic Models* **16** 27–48.

Tassiulas, L., A. Ephremides. 1992. Stability properties of constrained queueing systems and scheduling policies for maximum throughput in multihop radio networks. *IEEE Transactions on Automatic Control* **37** 1936–1948.

Teh, Y. C. 2000. Dynamic scheduling for queueing networks derived from discrete-review policies. D. R. McDonald, S. R. E. Turner, eds., *Analysis of Communication Networks: Call Centres, Traffic, and Performance*. AMS Bookstore, 73–95.

Terekhov, D., J. C. Beck, Kenneth N. Brown. 2009. A constraint programming approach for solving a queueing design and control problem. *INFORMS Journal on Computing* Published online in Articles in Advance.

Terekhov, D., M. K. Doğru, U. Özen, J. C. Beck. 2012a. Solving two-machine assembly scheduling problems with inventory constraints. *Computers and Industrial Engineering* **63** 120–134.

Terekhov, D., D. G. Down, J. C. Beck. 2012b. Stability of a polling system with a flow-shop server. Tech. Rep. MIE-OR-TR2012-01, Department of Mechanical and Industrial Engineering, University of Toronto. Available from http://www.mie.utoronto.ca/labs/ORTechReps/.

Terekhov, D., T. T. Tran, J. C. Beck. 2010. Investigating two-machine dynamic flow shops based on queueing and scheduling. *Proceedings of ICAPS'10 Workshop on Planning and Scheduling Under Uncertainty*.

Terekhov, D., T. T. Tran, D. G. Down, J. C. Beck. 2012c. Long-run stability in dynamic scheduling problems. *Proceedings of the 22nd International Conference on Automated Planning and Scheduling (ICAPS'12)*.

Terekhov, D., T. T. Tran, D. G. Down, J. C. Beck. 2012d. A three-level framework for integration of queueing theory and scheduling. *Working Paper* .

Terrazas-Moreno, S., A. Flores-Tlacuahuac, I. E. Grossmann. 2008. Simultaneous design, scheduling, and optimal control of a methyl-methacrylate continuous polymerization reactor. *AIChE Journal* **54** 3160–3170.

Thiagarajan, S., C. Rajendran. 2005. Scheduling in dynamic assembly job-shops to minimize the sum of weighted earliness, weighted tardiness and weighted flowtime of jobs. *Computers and Industrial Engineering* **49** 463–503.

Thomas, M., H. Szczerbicka. 2007. Evaluating online scheduling techniques in uncertain environments. *Proceedings of the 3rd Multidisciplinary International Scheduling Conference (MISTA'07)*.

Tian, N., Z. G. Zhang. 2006. *Vacation Queueing Models: Theory and Applications*. Springer.

Toomey, J. W. 2000. *Inventory Management: Principles, Concepts and Techniques*. Kluwer Academic Publishers.

Towsley, D., F. Baccelli. 1991. Comparisons of service disciplines in a tandem queueing network with real time constraints. *Operations Research Letters* **10** 49–55.

Tran, T. T. 2011. Using queueing analysis to guide combinatorial scheduling in dynamic environments. Master's thesis, Department of Mechanical and Industrial Engineering, University of Toronto.

Tran, T. T., D. Terekhov, D. G. Down, J. C. Beck. 2013. Hybrid queueing theory and scheduling models for dynamic environments with sequence-dependent setup times. *Proceedings of the 23rd International Conference on Automated Planning and Scheduling (ICAPS'13)*.

Van Der Wal, J., U. Yechiali. 2003. Dynamic visit-order rules for batch-service polling. *Probability in the Engineering and Informational Sciences* **17** 351–367.

Van Hentenryck, P., R. Bent. 2006. *Online Stochastic Combinatorial Optimization*. MIT Press.

van Hoeve, W.-J., I. Katriel. 2006. Global constraints. F. Rossi, P. van Beek, T. Walsh, eds., *Handbook of Constraint Programming*, chap. 6. Elsevier, 169–208.

van Wijk, A. C. C., I. Adan, O. J. Boxma, A. Wierman. 2010. Fairness and efficiency for polling models with the κ-gated service discipline. *Working Paper* .

Varaiya, P., J. Walrand, C. Buyukkoc. 1985. Extensions of the multiarmed bandit problem: the discounted case. *IEEE Transactions on Automatic Control* **30** 426–439.

Veatch, M. H., L. M. Wein. 1992. Monotone control of queueing networks. *Queueing Systems* **12** 391–408.

Vermeulen, I. B., S. M. Bohte, S. G. Elkhuizen, H. Lameris, P. J. M. Bakker, H. L. Poutré. 2009. Adaptive resource allocation for efficient patient scheduling. *Artificial Intelligence in Medicine* **46** 67–80.

Vermorel, J., M. Mohri. 2005. Multi-armed bandit algorithms and empirical evaluation. *Proceedings of the 16th European Conference on Machine Learning (ECML'05)*. 437–448.

Vishnevskii, V. M., O. V. Semenova. 2006. Mathematical methods to study the polling systems. *Automation and Remote Control* **67** 173–220.

Weber, R. R., S. Stidham Jr. 1987. Optimal control of service rates in networks of queues. *Advances in Applied Probability* **19** 202–218.

Weber, R. R., G. Weiss. 1990. On an index policy for restless bandits. *Journal of Applied Probability* 637–648.

Wein, L. M., P. B. Chevalier. 1992. A broader view of the job-shop scheduling problem. *Management Science* **38** 1018–1033.

Weiss, G. 1988. Branching bandit processes. *Probability in the Engineering and Informational Sciences* **2** 269–278.

Whittle, P. 1988. Restless bandits: Activity allocation in a changing world. *Journal of Applied Probability* **25A** 287–298.

Wierman, A., M. Harchol-Balter, T. Osogami. 2005. Nearly insensitive bounds on SMART scheduling. *ACM SIGMETRICS Performance Evaluation Review* **33** 205–216.

Wierman, A., E. Winands, O. Boxma. 2007. Scheduling in polling systems. *Performance Evaluation* **64** 1009–1028.

Williams, R. J. 1996. On the approximation of queueing networks in heavy traffic. F. P. Kelly, S. Zachary, I. Ziedins, eds., *Stochastic Networks: Theory and Applications*, chap. 3. Oxford University Press, 35–56.

Williams, R. J. 1998. Diffusion approximations for open multiclass queueing networks: sufficient conditions involving state space collapse. *Queueing Systems* **30** 27–88.

Winands, E. M. M. 2007. Polling, production and priorities. Ph.D. thesis, Technische Universiteit Eindhoven.

Wolff, R. W. 1989. *Stochastic modeling and the theory of queues*. Prentice Hall.

Xia, C. H., J. G. Shanthikumar, P. W. Glynn. 2000. On the asymptotic optimality of the SPT rule for the flow shop average completion time problem. *Operations Research* **48** 615–622.

Yang, J., M. E. Posner. 2005. Scheduling parallel machines for the customer order problem. *Journal of Scheduling* **8** 49–74.

Yechiali, U. 1991. Optimal dynamic control of polling systems. *Queueing, Performance and Control in ATM (ITC-13)* 205–217.

Yechiali, U. 1993. Analysis and control of polling systems. *Performance Evaluation of Computer and Communication Systems* 630–650.

Yeow, W.-L., C.-K. Tham, W.-C. Wong. 2006. Hard constrained semi-Markov decision processes. *Proceedings of the Twenty-First National Conference on Artificial Intelligence (AAAI'06)*. 549–555.

Yokoyama, M. 2008. Flow shop scheduling with setup and assembly operations. *European Journal of Operational Research* **187** 1184–1195.

Zonderland, M. E., R. J. Boucherie, N. Litvak, C. L. A. M. Vleggeert-Lankamp. 2010. Planning and scheduling of semi-urgent surgeries. *Health Care Management Science* **13** 256–267.