Using Queueing Analysis to Guide Combinatorial Scheduling in Dynamic Environments

by

Tony T. Tran

A thesis submitted in conformity with the requirements
for the degree of Master of Applied Science
Graduate Department of Mechanical and Industrial Engineering
University of Toronto

Abstract

Using Queueing Analysis to Guide Combinatorial Scheduling in Dynamic Environments

Tony T. Tran

Master of Applied Science

Graduate Department of Mechanical and Industrial Engineering

University of Toronto

2011

The central thesis of this dissertation is that insight from queueing analysis can effectively guide standard (combinatorial) scheduling algorithms in dynamic environments. Scheduling is generally concerned with complex combinatorial decisions for static problems, whereas queueing theory simplifies the combinatorics and focuses on dynamic systems. We examine a queueing network with flexible servers under queueing and scheduling techniques. Based on the strengths of queueing analysis and scheduling, we develop a hybrid model that guides scheduling with results from the queueing model.

In order to include setup times, we create a logic-based Benders decomposition model for a static representation of the queueing network. Our model is able to find optimal schedules up to 5 orders of magnitude faster than the only other model in the literature. A hybrid model is then developed for the dynamic problem and shown to achieve the best mean flow time while also guaranteeing maximal capacity.

Acknowledgements

There are a number of people that I would like to thank for assisting me during my efforts of writing this dissertation.

Thanks to J. Christopher Beck. His guidance and insights these past two years have been fundamental to the development of my research as well as to me as a researcher. Not only has he been a wonderful mentor, but also a role model which I can use to set the standard for my own work.

Thanks to Daria Terekhov for suggesting graduate studies to me as well as helping me through this journey.

Thanks to my parents and my grandmother for their support, love and for always pushing me to work harder to achieve my goals.

Lastly, thanks to An Nguyen for all her love and support. She has been by my side through the best and worst of times to keep me humble, sane, and happy.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

The central thesis of this dissertation is that insight from queueing analysis can be effectively used to guide standard (combinatorial) scheduling algorithms in a dynamic environment. We illustrate the advantages of combining queueing and scheduling by creating a hybrid approach to a dynamic queueing network from the literature. Individually, the queueing and scheduling approaches exhibit different characteristics which lead to better performance when situations favour one over the other. The integration of the methods produces increased performance by maintaining the respective strengths of each model. Cooperation of queueing and scheduling provides a framework that more accurately models real world problems requiring both the dynamic aspects of queueing theory and the combinatorial optimization techniques of scheduling.

1.1 Motivations

This dissertation is motivated by the goal of solving dynamic scheduling problems better. Scheduling techniques often study static problems and lack the stochastic analysis of queueing theory. We are interested in investigating the integration of these two fields of research. In more detail, the motivations for the work in this dissertation are as follows:

1. **The need for complex combinatorial reasoning in dynamic networks** - Since the in-

ception of queueing theory, many dynamic systems have been analyzed using stochastic models. These models are often limited to simple networks because formal analysis of a system is often beyond the current state of the art when complex combinatorial structure is present. The difficulty of deriving analytical results for complex systems has resulted in queueing theory providing few insights on how to actually schedule jobs in the system. Research which includes scheduling in queueing theory are often comparisons between simple dispatch rules such as first come first serve (FCFS) and shortest processing time first (SPT) [113]. However, many real world problems that are dynamic are also complex where the sequence and/or allocation of jobs has a large impact on system performance.

2. **The inclusion of scheduling quality measures in a dynamic network** - Andradóttir et al. [4] analyze a queueing network with flexible servers. The policy presented in their work guarantees maximization of the capacity for a given system in terms of the arrival rate of jobs. For the operator of the system (e.g., the factory scheduler), maximizing capacity means resources are not wasted and the operator knows when to expand their resources. However, a customer (e.g, the owner of a job) is not interested in the stability of the system. Rather, they are more concerned with performance measures of a job such as the average flow time. A customer will prefer servicing to occur in a timely manner and will judge the system based on how long they had to wait before they could be serviced.

3. **The integration of queueing theory and scheduling** - Queueing theory and scheduling are two areas of research that are both concerned with efficient resource management. However, the assumptions, goals, and approaches vary widely between the two. Queueing theory focuses on the probabilistic nature of a system in order to model the dynamic behaviour of a network. In comparison, scheduling deals with the combinatorics of a problem where the system is often static. A link between the two research areas could lead to more accurate representations and better performing schedulers of real world

problems which are both complex and dynamic.

1.2 Overview of Dissertation

The outline of the dissertation is as follows.

Chapter 2 reviews the background information for the work in this dissertation. We start by introducing queueing theory and the optimal design and control of queues. In the following section, different techniques and systems studied in the scheduling literature are presented. Further, we provide a brief overview of Benders decomposition and an extension to the partitioning method. Finally, a queueing network problem studied by Andradóttir et al. [4] is presented. The network provides the basis of the work in this dissertation and is the inspiration of the following chapters.

In Chapter 3, we study the problem presented by Andradóttir et al., but simplify the problem to remove setup times and re-routing. We also concern ourselves with a different, scheduling-based, objective: minimize mean flow time. The flow time performance of the existing queueing policy is compared to two scheduling policies. A hybrid model is presented which uses aspects of the queueing and scheduling policy. Experimental results show that the hybrid model significantly outperforms the queueing policy and scheduling model, resulting in the best overall performance.

We wish to reintroduce setup times to the problem by Andradóttir et al. To do so, a more sophisticated scheduling technique needs to be developed. We simplify the problem in a different way. By making the problem static, we are able to concentrate on the combinatorics of the problem. Chapter 4 focuses on the unrelated parallel machine scheduling problem (PMSP) with sequence and machine dependent setup times. We present a logic-based Benders decomposition model for the PMSP and demonstrate the ability of our model to find optimal solutions for problems four times in size greater than previously possible. On problems where both the previous approach and the Benders decomposition are able to find optimal schedules,

the decomposition is able to find and prove an optimal schedule up to five orders of magnitude faster. A heuristic extension to the model is also developed enabling the method to compete with current state-of-the-art metaheuristic approaches in terms of problem size and solution quality. One significant advantage of this heuristic, however, is the ability to directly quantify the quality of a solution in terms of the optimality gap.

Chapter 5 combines the work in Chapters 3 and 4 to allow us to examine the original problem by Andradóttir et al. by using the logic-based Benders decomposition to find schedules with the minimum makespans for each period. With setup times, the queueing policy is found to always maintain lower mean flow time than a pure scheduling model. We create a hybrid model incorporating some of the advantageous qualities of the queueing policy. The new model is able to experimentally produce the best mean flow time results across all system loads when tested. Further, we formally show that the hybrid model is able to maximize the capacity of the system and guarantee stability whenever possible.

Lastly, in Chapter 6, we conclude and suggest directions for future work based on this dissertation.

Appendix A defines the standard notation for queueing models, while Appendix B and C detail the parameters for experiments in Chapters 3 and 5 respectively.

1.3 Summary of Contributions

The contributions of this dissertation are as follows:

1. The creation of a hybrid queueing and scheduling model that takes advantage of the strengths of both approaches. The hybrid is able to provide formal stability guarantees, while also empirically achieving the best mean flow performance of all methods tested.

2. A complete method using logic-based Benders decomposition for finding minimal makespan schedules to a PMSP that outperforms previous optimal approaches by up to five orders of magnitude in speed.

3. An extension to the logic-based Benders decomposition which abandons optimality to solve very large problems. The improved model is found to be competitive with current state-of-the-art heuristic approaches in terms of solution quality while significantly reducing the CPU time.

4. An investigation of the performance differences between a queueing policy and two scheduling models. Results demonstrated that short term greediness can lead to undesirable long term performance when a system is heavily loaded. However, a conservative strategy is able to perform much better under heavy loads and also maintain competitive or even better performance when a system is lightly loaded.

5. The inclusion of a scheduling quality measure in a queueing network which was previously concerned only with maximizing capacity. The utility of such an extension is to better represent the concerns of customers in the system as well as those of the operator.

Chapter 2

Literature Review

In this chapter, we present a review of three topics. In the first section, queueing theory is introduced. Section 2.2 presents scheduling methods and techniques. Finally, an overview of Benders decomposition, work which is relevant for Chapter 4, follows. At the end of the chapter, the dynamic queueing network with flexible servers problem is presented. The work in this dissertation builds on this problem. The problem considered is a dynamic system which we will use to study the potential for combining scheduling and queueing theory.

2.1 Queueing Theory

Queueing theory is the study of waiting lines in dynamic systems. Jobs arrive to the system and are processed by one or more servers. When server are unable to service a job arriving to the system, the job is added to a queue where it awaits processing.

The queueing theory literature focuses on descriptive and prescriptive models. Descriptive models are concerned with describing current real-world situations, whereas prescriptive models attempt to optimize what the real-world should be [46]. Our review will focus on the prescriptive models developed in queueing theory, also called optimal design and control of queues. For the reader interested in descriptive models, Gross and Harris [46] spends most of their book detailing descriptive models.

In the following two sections, we review some of the work that has been done in both the design and control of queues respectively. For a more in-depth review of work in these areas, Crabill [29] presents a bibliography of queueing work and Tadj and Choudhury [98] survey work done in the optimal design and control of queues.

2.1.1 Optimal Design of Queues

The study of the design of queues addresses the question of finding system parameters that optimize a given queueing system. A variety of parameters are studied such as the maximum arrival rate a system can cope with or the number of servers required for a given arrival rate. Typical solution methods use descriptive queueing results to model the problem (e.g., to derive an objective function) within some optimization framework [46].

The following list summarizes some of the work done on the design of queues.

- One of the first uses of presciprtive queueing design models was used to find the optimal number of clerks to place at service counters in a plant belonging to Boeing Airplane Company [16]. Brigham observed that arrivals followed a Poisson process and service rates were exponential. These observations allowed an M/M/c model representation as notated by Kendall [60], which is the standard notation in the queueing literature and detailed in Appendix A. Using both clerk idle time and customer wait time in a cost function, he presented results showing the optimal number of clerks as a function of the ratio between arrival rate and service rates.

- Morse [78] used differential calculus to find optimal service rates for minimizing the cost of an M/M/1 and M/M/1/K queue by taking the derivative of the cost function with respect to the service rate and then setting to zero. He also analyzed an M/M/c/c system where no queue exists and found that for given arrival and service rates, an optimal number of servers is calculable.

- Some work has also focused on determining the number of active servers in a system

based on the number of customers. The rules developed depend only on the current state of a system and are called stationary policies. Winston [114] uses Markov Decision processes to derive conditions where the number of servers is a non-decreasing function of the numbers of customers. An M/M/2 system is studied by Bell [12] and an optimal policy is developed which defines at which system states one should increase or decrease the number of active servers.

- Reneging and balking, a representation of customer impatience to exit the system prematurely if the queues or wait times are too long, are considered when machines are allowed to fail [59]. In this work, a linear cost function is created based on waiting times, service completion, idle servers, broken servers, and reneged or balked jobs. Using the long run expected performance of the system, the optimum number of servers are calculated.

2.1.2 Optimal Control of Queues

Optimal queue control is the study that decides on the admission, servicing, routing and scheduling of jobs in a system to achieve an objective. Control is often done through use of policies which map a system state into the actions to be performed. Research primarily focuses on determining when certain policies are optimal, however, some are concerned with finding the optimal values of the control parameters [46].

Much of the early work on queue control was on stationary policies that prescribe the same action whenever a system is in a specific state. Romani [88] examined an M/M/c model and considered a policy that added servers if the queue length is at a critical size and new jobs arrive. Following the policy guarantees that the queue never exceeds the critical size as new servers are added accordingly. New servers continue to serve jobs and are only removed when the queue has emptied. Moder and Phillips [75] study a similar problem except that the system has a certain number of servers always available and a maximum number of servers that can be added. Moder and Phillips derive steady-state probabilities of witnessing a certain number

of customers during different states and some measures of effectiveness such as idle time, expected queue lengths and expected waiting times.

One well-studied family of policies are so-called *threshold* policies which are defined as follows: given a non-decreasing set of predetermined queue sizes, a threshold policy will increase the number of servers as values in the set are realized and maintain these servers until the system is once again empty or below the threshold. Optimality of a threshold policy for various systems has been shown by a number of researchers [31, 30, 34, 85].

The work in queueing theory has for the most part not considered complex scheduling of queues. The scheduling approaches used are often simple dispatch policies: first come first serve (FCFS), last in first out (LIFO), shortest processing time first (SPT), longest processing time first (LPT), and random selection for service (RSS). A well known rule for scheduling queues where customers have priorities is the $c\mu$ rule which chooses the customer with the largest $c\mu$ index to service. The parameter c represents a holding cost attributed to a class and μ is the service rate of the class. This rule has been shown to be optimal for an M/M/1 system with infinite buffer size [18, 62, 110]. However, Kim and Oyen [62] show that simple rules, such as the $c\mu$ rule, are not optimal for a more complex systems where the buffer sizes are finite and jobs must be rejected.

Job priorities under different scheduling policies for an M/G/1 queue are examined by Wierman and Harchol-Balter [112]. Various policies are analyzed and compared based on the *fairness* to jobs where a policy with higher variation in response times to different jobs is considered less fair. Previous researchers concerned themselves with the mean response times which lead to the $c\mu$ rule. However, Wierman and Harchol-Balter show that policies that bias towards small jobs to minimize response time may end up being unfair to larger jobs by delaying service. Their results showed that some policies, like preemptive shortest job first (PSJF) and shortest remaining processing time first (SRPT), have similar characteristics and behaviour to each other but lead to different fairness classifications.

Andradóttir et al. [4] examine a queueing network with flexible servers. Jobs of different

classes enter the system and are processed on one or more servers. Processing times vary depending on the server and class. The authors present a policy that is able to maximize the capacity of any given system. We present a more detailed summary of the work by Andradóttir et al. in Section 2.4.

Queueing theory provides the analytical tools to understand a dynamic system. Specifically, the control of queues gives insight into the long run performance of the decisions made by an operator. Understanding the long run performance can help to make better decisions. However, what is currently missing from the queueing literature is a focus on systems requiring complex combinatorial decisions.

2.2 Scheduling

We review work that has been done in scheduling research based on the two books *Scheduling* [84] and *Online Stochastic Combinatorial Optimization* [105]. The *Scheduling* book defines scheduling as the allocation of scarce resources to tasks over time with the goal of optimizing one or more objectives. This definition of scheduling is remarkably similar to that of queueing theory. In general, both are concerned with optimizing a system with limited resources. The similarity of definition between scheduling and queueing may be misleading when one views the two different research areas. The assumptions and approaches in scheduling differ greatly from that of queueing theory. While queueing theory looks at uncertain and dynamic systems, the majority of work in scheduling focuses on deterministic and static problems. Some research in scheduling has looked at uncertainty and dynamism [7, 50, 72, 83, 107, 108], but the body of research is heavily focused on the deterministic models.

The following section outlines static scheduling, the topic of the *Scheduling* [84] book. Static scheduling can further be categorized into deterministic and stochastic scheduling based on whether complete information is available or not. We focus on the deterministic problems where there exists no uncertainty in the following section. Section 2.2.2 reviews dy-

namic scheduling problems where unpredictable events cause the system to change significantly enough that a static approach would lead to poor performance.

2.2.1 Static Scheduling

In static (deterministic) scheduling, a set of jobs are to be processed by a set of machines to optimize some objective. These problems have complete information regarding job data such as: release date, due date, processing time, weight, and cost of jobs. Further, the machines are classified as:

- (1) *Single* machine - one machine to service jobs.
- (P) *Identical parallel* machines - multiple machines each processing jobs at the same speed.
- (Q) *Uniform parallel* machines - multiple machines that process jobs with linearly related speeds.
- (R) *Unrelated parallel* machines - multiple machines that process a job with unrelated speeds on different machine.
- (F) *Flow shop* - each job contains a chain of tasks that are to be processed on each machine in the same order.
- (J) *Job shop* - each job contains a unique chain of tasks that are to be processed on a number of machines. Unlike with flow shops, the order of each job in a job shop problem may differ.
- (O) *Open shop* - each job contains multiple tasks that must be processed on a number of machines. The ordering that a job is processed on the machines is not specified.

We make use of the 3-field notation, $\alpha|\beta|\gamma$ as defined by Graham et al. [43] to classify scheduling problems. The first field, α denotes the machine configuration of the system. β clas-

sifies the processing restrictions and constraints such as: preemption, resource requirements, and precedence relations. The last field represents the performance measure to optimize. For example, $1|r_j|\sum C_j$ represents a scheduling problem with 1 machine where jobs have release dates and must be sequenced in such a way to minimize the sum of their total completion time.

In Section 2.2.1.1, single machine scheduling problems are discussed. The following section outlines work on parallel machines. Finally, the research on *flow shop*, *job shop*, and *open shop* problems are presented. For an in depth review on deterministic scheduling, see surveys by Graham et al. [43], Graves [44], and Lee et al. [68].

2.2.1.1 Single Machine Problems

The single machine problem has been the subject of a lot of research since the work of Jackson [56]. He modeled a production line as a $1||L_{max}$ problem where the objective is to minimize the maximum lateness. The earliest due date first (EDD) dispatch rule, also known as Jackson's rule, is developed for this particular problem to sequence jobs in increasing order of their due dates. A generalization to the model studied by Jackson is the $1|r_j|L_{max}$ problem where jobs have release dates. Many researchers looked at branch and bound methods for the $1|r_j|L_{max}$ problem [19, 71, 82]. Lenstra et al. [69] show that this problem is strongly NP-hard.

Another early study in scheduling of single machine systems is a $1||\sum w_jC_j$ problem by Smith [96]. He proves the optimality of the weighted shortest processing time first (WSPT) for this problem. Extensions of the weighted completion time problem have been studied with added precedence constraints [66, 76, 77, 94] or preemption and release dates [64, 90]. The problem with preemption and release date is solvable with an extension of Smith's work for cases where the objective is to minimize the total completion time, but in the case of a weighted completion time, it is unary NP-hard [64].

2.2.1.2 Parallel Machine Problems

Parallel scheduling problems tend to be more difficult than the single machine counterpart. Simple dispatch rules obtain the optimal schedules for both $1||C_{max}$ and $1||\sum w_jC_j$. Specifically, any non-idling schedule will produce the minimum makespan of a schedule while, as stated earlier, WSPT provides the optimality for the weighted completion time problem. Increasing the number of machines to two identical servers, $P2||C_{max}$ and $P2||\sum w_jC_j$, makes the problem binary NP-hard [17, 69].

The difficulty of parallel machine scheduling leads most of the work towards heuristic methods. Dispatch policies are a popular approach which are often compared by their worst case bound performance [25, 37, 42]. Min and Chen [74] solve a large scale $P||C_{max}$ problem using a genetic algorithm. They compare their algorithm to the LPT heuristic and simulated annealing to find that genetic algorithm finds better solutions than the LPT rule and equal or better solutions to the simulated annealing in less time. Specifically, the genetic algorithm was found to scale better to larger problems than simulated annealing.

Focusing on the objective of minimizing the total completion time when there are identical servers, $P||\sum C_j$, Conway et al. [26] present an $O(n \log n)$ algorithm based on SPT. Their analysis illustrate that, without weighted costs, the problem has a polynomial solvable algorithm. Horowitz and Sahni [55] generalize results from Conway et al. on the identical machine problem to uniform machines, $Q||\sum C_j$. Their procedure matches the largest jobs to latest positions on fast machines in $O(n \log n)$ time to acquire optimal schedules. Horn [54] and Bruno et al. [17] further extend the problem to unrelated parallel machines, $R||\sum C_j$, and represent the model as a transportation problem. They create a mixed integer program (MIP) model which can be solved in $O(n^3)$ time.

2.2.1.3 Flow Shop Scheduling

In many manufacturing facilities, a number of tasks must be performed on every job. The tasks often have to be done in the same order for all jobs and require different machines for each

task. The environment described is considered a *flow shop*.

Johnson [58] presented a solution for the $F2||C_{max}$ problem known as Johnson's rule. He shows that an optimal solution can be found in $O(n \log n)$ time. The algorithm arranges the jobs such that a job j precedes a job k if $\min(p_{1j}, p_{2k}) \leq \min(p_{2j}, p_{1k})$. Further, Johnson proves that there exists an optimal schedule for the two machine *flow shop* that is a permutation schedule, i.e., one with the same processing order on both machines.

The $F3||C_{max}$ problem is proven to be NP-hard [38]. However, Johnson [58] shows that if all processing times on the first machine are greater than all jobs on the second machine or if all machines on the third machine are greater than all those on the second, optimality can be achieved using a modification of his algorithm for the two machine case.

Wagner [109] formulates an integer program for $F||C_{max}$. The difficulty of solving a flow shop problem make heuristics more popular. Taillard [99] show a tabu search that can outperform heuristic dispatching policies. Other tabu search algorithms have also been presented for the flow shop problem that perform well when finding schedules for large problems. Nowicki and Smutnicki [80] and Grabowski and Wodecki [41] make use of *block properties* in a tabu search algorithm to find feasible solutions for problems of size 500 jobs and 20 machines.

2.2.1.4 Job Shop Scheduling

Job shop, like *flow shop*, is an environment that deals with jobs containing multiple tasks with fixed routes. However, unlike the *flow shop* environment, the routes are not necessarily the same for each job.

Jackson [57] extends Johnson's algorithm for the two machine *flow shop* problem to the two machine *job shop* problem. Jackson places jobs into four groups: jobs only requiring machine 1, jobs only requiring machine 2, jobs requiring machine 1 first then machine 2, and jobs requiring machine 2 first then machine 1. To minimize the makespan of the problem, jobs are to be ordered in the same method as Johnson's rule with the addition of priorities of jobs based on the group that a job belongs to. Priority on a machine is given to jobs that

require the machine first, while jobs that require a machine second are given lower priority on that machine. Jobs that are only to be scheduled on one machine are inserted between the two different priority groups. Jackson's results are only obtainable for systems where jobs only have at most two tasks. If a job has three or more tasks, the problem becomes NP-hard [38, 69].

A large body of work has examined branch and bound techniques to minimize the makespan in a *job shop*. Carlier and Pinson [20] present a branch and bound algorithm based on the single machine scheduling problem. However, complete approaches are very limited to smaller problems. Therefore, most of the work in the past 15 years has focused on approximate algorithms because of the intractable nature of *job shop* scheduling.

Adams et al. [1] present the Shifting Bottleneck heuristic used for the $J||C_{max}$ problem. The heuristic schedules the machines one by one based on the machine identified as the bottleneck. After a machine is sequenced, the already scheduled machines are optimized again based on the addition of the newly sequenced machine.

Local search methods have shown the most promise for the $J||C_{max}$ problem. Simulated annealing approaches are proposed by Van Laarhoven et al. [106] and Yamada and Nakano [118]. Yamada and Nakano [117] also present a genetic algorithm approach to deal with the *job shop* scheduling problem. These approaches have tested problem sizes of up to 30 jobs and 10 machines or 15 jobs and 15 machines.

Of the numerous local search methods, tabu search has been shown to work very well for the $J||C_{max}$ problem. The concept of a tabu search is to create a tabu list of the most recent moves or states that the search has made or visited. Subsequent moves to neighbouring solutions are forbidden if it is on the tabu list unless the move would produce a solution that is better than any previously found schedule. By avoiding previous moves or states and restarting the search from a previously found solution after a certain number of iterations lead to no improvement, tabu search attempts to avoid getting stuck in a local minima. Taillard [100] show that a tabu search implementation is able to out perform the Shifting Bottleneck heuristic and simulated annealing approach. Nowicki and Smutnicki [79, 81] also present a tabu search algorithm to

minimize the makespan called TSAB and the follow-up algorithm i-TSAB. The tabu search methods are tested for problems of sizes up to 100 jobs and 20 machines.

Constraint programming (CP) has shown promise as a constructive search for scheduling job shop problems. Beck [9] developed the solution-guided search and showed that it significantly outperformed other constructive search techniques. However, CP was unable to compete with state-of-the-art local search methods. Using the knowledge gained from solution-guided search, Beck et al. [10] created a hybrid model based on CP and tabu search. Their results showed that the inference capabilities of CP was able to significantly improve the performance of the tabu search algorithm and that the hybrid model could compete with the state-of-the-art local search techniques. A particular advantage is that the hybrid model is a complete approach.

2.2.1.5 Open Shop Scheduling

In the open shop scheduling problem, jobs require processing on a number of machines. The order in which a job is processed on machines does not matter. However, a job may only be processed by a single machine at a time.

Similar to *flow shop* and *job shop* scheduling, a polynomial algorithm exists for the two machine case when minimizing makespan. Gonzalez and Sahni [40] present an $O(n)$ algorithm that divides jobs into two categories; one category for jobs that have longer processing times on the first machine and one with longer processing times on the second machine. The schedule is then built from the middle such that all jobs with longer processing times on the first machine are scheduled after the middle and the rest are scheduled before the middle point. Gonzalez and Sahni show that an optimal solution is achievable by following this method. They also show that $O3||C_{max}$ is binary NP-hard.

Sevastianov and Woeginger [91] present a polynomial time approximation scheme for the $O||C_{max}$ problem. They initially create an optimal schedule using only a subset of jobs that have characteristics of a *big* job type. The schedule induces gaps which are then greedily filled by *small* and *tiny* jobs. This algorithm can be performed in $O(n \log n)$ time if parameters

are chosen such that the number of large jobs are limited. However, the algorithm may be impractical to perform if parameters are chosen such that finding an optimal schedule for the *big* jobs is intractable.

If the *open shop* problem is allowed to have preemption, Lawler and Labetoulle [67] provide a polynomial time algorithm to minimize the makespan of a schedule. Given a matrix of processing times $P = p_{ij}$, they show that it is possible to create a schedule with makespan,

$$C_{max} = \max\{\max_j\{\sum_i p_{ij}\}, \{\max_i\{\sum_j p_{ij}\}\}.$$

Lawler and Labetoulle show that such a schedule is optimal.

2.2.2 Dynamic Scheduling

Traditionally, scheduling focuses on static problems and does not account for changes that may happen. However, many real world problems can witness unpredictable events which change the original problem. Manufacturing facilities may create a schedule to meet the demand of current customer orders, but the request of new arrivals could drastically change the optimal schedule. Further, machine failures can occur during execution where a previously feasible schedule may become infeasible. One can also see application in emergency room scheduling in hospitals because at any moment, a patient may enter with a critical condition which has higher priority than others and requires immediate attention. In an ever changing environment, static abstractions are unable to properly accommodate for these real time changes [28, 93]. Recent trends in scheduling has seen a need for scheduling models that are robust in these dynamic environments [83, 97, 116].

The dynamic scheduling literature has studied a significant number of events in a dynamic environment that exhibit a large impact on the performance of a schedule. The events studied have been classified into two categories by Ouelhadj and and Petrovic [83]:

- **Resource-related:** machine breakdown, operator illness, unavailability or tool failures, loading limits, delay in the arrival or shortage of materials, defective material (material

with wrong specification), etc.

- **Job-related:** rush jobs, job cancellation, due date changes, early or late arrival of jobs, change in job priority, change in job processing time, etc.

The methodologies of dynamic scheduling have been categorized as completely reactive scheduling, predictive-reactive scheduling, and robust proactive scheduling [7, 50, 72, 83, 107, 108].

2.2.2.1 Completely Reactive Scheduling

A schedule is never generated in advance for completely reactive scheduling. Commonly, priority dispatching policies are used to make decisions on demand. Dispatch rules are often easy and fast to implement while usually following some greedy short term goals. Simple rules could be to randomly assign a job to a machine when it is made available or select the job that has been waiting longest. More complex rules also exist that decide which job to schedule based on multiple criteria.

Blackstone et al. [15] and Haupt [48] provide a comprehensive survey of dispatching rules and policies. Blackstone et al. list 34 dispatch rules in their survey and provide analytical approaches, simulation techniques, bias of estimates produced by simulation, sample size and evaluation criteria for a number of different rules.

2.2.2.2 Predictive-Reactive Scheduling

Predictive-reactive scheduling is the most common method used for dynamic scheduling environments [83]. Predictive-reactive scheduling is the process where one initially creates a schedule and then reschedules at a later time based on occurrences of real-time events. A predictive schedule (baseline schedule, pre-schedule, or proactive schedule) attempts to create solutions based on knowledge of the system before processing takes place. Reactive scheduling occurs during execution and makes use of up-to-date information to revise the predictive

schedule. Along side efficiency goals, the scheduler often wishes to have a stable reactive schedule that does not deviate too far from the predictive schedule.

Wu et al. [115] present three heuristic rescheduling algorithms for a single machine system with the criteria of makespan minimization and schedule stability. Stability is measured in two ways, the absolute deviation of job start times and sequence changes from the original schedule. In their study, they consider machine disruptions to model machine failures, processing time uncertainty when jobs take longer than initially planned, and unforeseen arrivals of jobs that require immediate processing. Rescheduling occurs directly after any disruptions occur and a machine is once again made available. They show experimentally the effectiveness of robust rescheduling due to the fact that significant increases to stability can be obtained with little or no effects on the makespan.

An issue that any rescheduling algorithm must address is when to react to events. Wu et al. [115] considered only rescheduling after a disrupted machine is again available, but the literature provides many more options. The question of when to reschedule has been categorized into three policies in the literature [89, 108]: periodic, event driven, and hybrid.

Church and Uzsoy [24] examine the performance of different rescheduling procedures under single and parallel machine systems. Their study looks at two extremes in the context of when to reschedule: continuous rescheduling which creates a new schedule every time a new event occurs and periodic rescheduling which defines some time interval between performing rescheduling. Based on computational results, they conclude that a periodic approach is viable when system utilization is very high or very low, or when due dates are very tight. Furthermore, Church and Uzsoy show that event driven rescheduling can improve performance. Here, optimization happens only when a significant event which is recognized to potentially cause significant disruption to the system occurs. However, event driven rescheduling was shown to have rapidly diminishing returns. Therefore, they believe event driven and periodic reschedulers to provide excellent quality solutions with less computationally expensive requirements than continuous rescheduling methods.

Matsuura et al. [70] provide a different rescheduling approach to conventional predictive-reactive scheduling. In their approach, a predictive schedule is created periodically. The schedule will be executed unless realization deviates sufficiently from the prediction in which case a dispatch policy takes over for the remainder of the period. They show that predictive-reactive approaches are better for periods of low disruptions, whereas dispatching rules perform better as the number of disruptions increase.

2.2.2.3 Robust Proactive Scheduling

Robust proactive scheduling focuses on the creation of predictive schedules that perform well in a dynamic environment. Robustness is attained by defining measures of predictability and scheduling with the foresight that disruptions may occur. Proactive scheduling often employs statistical information of uncertainty to create predictive schedules that maintain feasibility and good performance when subject to events.

Mehta and Uzsoy [72] present a robust proactive approach for a single machine system subject to breakdowns. In their paper, they attempt to minimize the maximum lateness in an environment where jobs arrive dynamically and machines fail randomly. Consider a predictive schedule S_p and realized schedule S_r for a planned time horizon. Allow $C_i(S_p)$ ($C_i(S_r)$) and $L_{max}(S_p)$ ($L_{max}(S_r)$) be the completion time of job i and maximum lateness in the predicted (realized) schedule. They account for the following costs through deviation of the planned schedule:

Earliness penalty $E_i(S_r)$

A penalty for completing a job earlier than initially planned in the original schedule.

$$E_i(S_r) = \max\{C_i(S_p) - C_i(S_r), 0\}.$$

Delay Penalty $D_i(S_r)$

A penalty for completing a job later than initially planned in the original schedule.

$$D_i(S_r) = \max\{C_i(S_r) - C_i(S_p), 0\}.$$

Realized Schedule Maximum Lateness $L_{max}(S_r)$

They assume that a job cannot leave the system until after the initially planned completion time, so $C_i(S_p)$ is considered when $C_i(S_r)$ completes earlier.

$$L_{max}S(r) = \max_i\{\max(C_i(S_p), C_i(S_r)) - d_i\}.$$

The proactive approach presented then attempts to minimize $\sum_i(E_i(S_r) + D_i(S_r))$ while ensuring an acceptable $L_{max}(S_r)$. Stochastic breakdowns of machines are considered by focusing on the expectation of these values, $E[\sum_i(E_i(S_r) + D_i(S_r))]$ and $E[L_{max}(S_r)]$. Their scheduler attempts to optimize a schedule by first creating a sequence of jobs and then inserting idle times before each job. Mehta and Uzsoy show that such a proactive approach could, if given the same starting sequence and rescheduling rules, create the same schedule as a presented predictive-reactive approach with the addition of decreased delay costs. Further, they show that proactive scheduling yields higher predictability than predictive-reactive scheduling if jobs are allowed to leave the system earlier than planned. They conclude that heuristically inserting idle times in a controlled manner when creating predictive schedules significantly improves the robustness of a schedule with very little deterioration of the primary performance measure, L_{max}.

Beck and Wilson [11] address a *job shop* scheduling problem when the duration of jobs are uncertain. They combine Monte Carlo simulation with deterministic scheduling algorithms to optimize the makespan of a schedule. Beck and Wilson show that the incorporation of uncertainty leads to stronger correlations between deterministic and probabilistic makespans and the corresponding ability to find better probabilistic makespans.

2.3 Benders Decomposition

In 1960, Benders [13] wrote a dissertation and a paper later in 1962 [14] on what would eventually be called Benders decomposition. In his work, a method is developed to solve problems which are commonly a mixed integer linear program (MILP). A general MILP that Benders

decomposition deals with is

$$\begin{aligned} \min \quad & c^T x + f^T y \\ \text{s.t.} \quad & Ax + By \geq b \\ & x \geq 0 \\ & y \geq 0 \text{ and integer} \end{aligned}$$

where x and y are vectors of variables having dimensions p and q, respectively. A, B are matrices, and b, c, f, are vectors having appropriate dimensions.

Benders decomposition partitions the problem into the integer and continuous variables by first assigning values to the integer variables, y, in a master problem and then solving the dual of the remaining problem in a subproblem. Let $\bar{y}$ be some fixed values assigned to y, the resulting problem is

$$\begin{aligned} \min \quad & c^T x \\ \text{s.t.} \quad & Ax \geq b - B\bar{y} \\ & x \geq 0 \end{aligned}$$

The dual problem of this LP is

$$\begin{aligned} \max \quad & (b - B\bar{y})^T u \\ \text{s.t.} \quad & A^T u \leq c \\ & u \geq 0 \end{aligned}$$

where u is a vector of dual variables.

The Benders approach then solves two different problems, a relaxation of the MILP to solve the pure integer problem as a master problem

$$\begin{aligned} \min \quad & z \\ \text{s.t.} \quad & z \geq f^T y + (b - By)^T \bar{u} \\ & y \geq 0 \text{ and integer} \end{aligned}$$

and a subproblem

$$\begin{aligned} \max \quad & f^T\bar{y} + (b - B^T\bar{y})^T u \\ \text{s.t.} \quad & A^T u \leq c \\ & u \geq 0 \end{aligned}$$

The master problem produces values for y which can then be used to define the subproblem. One then checks the solution by setting $UB = \min(UB, f^T y + (b - B^T\bar{y})^T\bar{u})$. If $\bar{z} \geq UB$, the optimal solution has been found. Otherwise, the cut $z \geq f^T y + (b - B^T y)^T\bar{u}$ is added to the master problem and another iteration is performed. z represents a lower bound on the achievable objective function and converges to optimal.

The partitioning scheme developed by Benders has been successfully applied to aircraft scheduling, power systems, and facility location problems to name a few [27, 92, 111]. In these studies, using Benders decomposition significantly decreased runtime for obtaining optimal solutions.

Hooker [53] developed a similar model to Benders decomposition, called logic-based Benders decomposition. He uses the Benders decomposition framework, but replaces the linear programming dual subproblem with a CP inference dual subproblem. The master problem incorporates a relaxation of this dual subproblem and the method iterates between master problem and subproblem until an optimal solution is found. By replacing the dual subproblem with CP, Hooker is able to generalize the Benders decomposition strategy and allow greater flexibility to model a larger class of problems. Through various studies, the logic-based Benders decomposition is able to outperform many of the previous state-of-the-art approaches in a variety of problems [33, 51, 52].

2.4 Dynamic Queueing Network with Flexible Servers

In this section, we introduce a problem considered by Andradóttir et al. [4]. The problem they examine provides the foundation for our work as a case on which we can investigate the effects of queueing and scheduling in an environment with complex and stochastic characteristics.

The queueing network studied by Andradóttir et al. is concerned with an environment where servers are flexible. Here, flexibility is the ability for a server to process a set of job classes and is analogous to the parallel machine scheduling problem presented earlier. The jobs arrive to the system and belong to a customer class which represents the current processing stage of a customer in a network. As the client progresses through the network, it may be routed to a different customer class. The class defines which servers can provide service to the customer as well as the rate at which the customer is served. Therefore, properly allocating servers to customer classes could increase efficiency because the service rates are not uniform. Andradóttir et al. considers the capacity of the queueing network and identify a tight bound on the achievable arrival rate as well as provide a job-allocation that can get arbitrarily close to the bound.

The focus of this dissertation is applying queueing and scheduling techniques to the queueing network with flexible servers as studied by Andradóttir et al. In following sections, we provide a detailed definition of the queueing network and describe the policy developed.

2.4.1 Problem Definition

Andradóttir et al. define a queueing network where incoming customers are routed to different classes and there are non-homogeneous servers that are able to serve a set of the available customer types. Customers arrive to the system with rate λ and are routed to a class i with probability p_i. Each class has a buffer of infinite size. Once in the system, a customer of class i switches to class k after completion of service with probability p_{ik} and leaves the system with probability p_{i0}.

The servers in the system can be assigned to different classes. At any time, a server is able to leave a class to begin service at another. However, switching times from a class may be nonzero. If a server j wishes to leave class i to start serving class k, that server must be idle for a time of $\zeta_{ik}^{j}(n)$ on the nth occurrence of such a switch. When assigned to a specific class i, a server j will work at a rate of μ_{ji}. This rate is dependent on the server and the class being serviced. If server j cannot be assigned to a class k, then $\mu_{jk} = 0$. If multiple servers are working on a single class at any point in time, the servers work in parallel acting similarly to an M/M/c queue where c is the number of servers assigned to the class.

An operator of this system is interested in the order to execute jobs and on which servers. Thus, the operator must decide which job to assign to an idle server if jobs are present in the system. Since processing times are dependent on the server, different assignments will lead to very different schedules. The goal of an operator of this system is therefore concerned with efficiently allocating servers such that the greatest arrival rate can be handled during execution.

2.4.2 Allocation LP

Andradóttir et al. present an allocation linear programming (LP) model which can find a bound on the arrival rate of jobs that a system can handle. To do so, the model aims to maximize the stability region of the network and define the greatest arrival rate, λ, to which the system can theoretically maintain a finite expected queue length and always ensure a non-negative probability of having empty queues. To define the allocation LP, the effective arrival rate of each customer class must be found. Given, K, a set of customer classes and, M, a set of servers, solving the following system of linear equations leads to the total arrival rate of a stable system,

$$\lambda_k = p_k\lambda + \sum_{i=1}^{K} p_{ik}\lambda_i \qquad k \in K$$

Let $\alpha_1, ..., \alpha_k$ be the solution to the system of equations if $\lambda = 1$; then $\lambda\alpha_k$ is the unique solution to the system of equations and the allocation LP is

$$
\begin{aligned}
\max \quad & \lambda \\
\text{s.t.} \quad & \sum_{j=1}^{M} \delta_{jk}\mu_{jk} \geq \lambda\alpha_k, \quad k \in K && (2.1)\\
& \sum_{k=1}^{K} \delta_{jk} \leq 1, \qquad\quad j \in M && (2.2)\\
& \delta_{jk} \geq 0 \qquad\qquad k \in K;\, j \in M && (2.3)
\end{aligned}
$$

where δ_{jk} represents the fractional amount of time that server j should serve customers of class k to maximize λ. Allow λ^* to be the optimal value of λ in the LP.

The allocation LP is a fluid representation of the queueing network. A fluid approach to such a network examines the system at very heavy loads where a system begins to exhibit the properties of a continuous fluid rather than discrete jobs. As the number of jobs in the system increase to infinity, the behaviour of the system converges to a fluid [21, 45]. The fluid approach also disregards switching times in the LP model. The argument is that the switching times become negligible as the number of jobs to serve between switching classes increases. Hence, constraint (2.1) ensures that the amount of service allocated to each class is sufficient to handle the workload that class will put on the system. Constraint (2.2) guarantees the model will only assign as much resource as the network has available and non-negativity is enforced by constraint (2.3).

2.4.3 Round Robin Policy

Andradóttir et al. develop the round robin policy to assign jobs to servers and achieve allocations that are arbitrarily close to the solution of the allocation LP. To ensure that the policy can achieve a given job arrival rate $\lambda < \lambda^*$, each server j visits classes in V_j in cyclic order, where V_j is an ordered list of all classes k with $\mu_{jk}\delta^*_{jk} > 0$. At each class $k \in V_j$, the server either serves a maximum of l_{jk} customers or until there are no more customers of class k in the

system.

Given $\epsilon = \frac{\lambda^* - \lambda}{\lambda^* + \lambda}$, $m_{jk} = \frac{1}{\mu_{jk}}$, and $1\{\delta_{jk} > 0\}$ a function that is 1 if $\delta_{jk} > 0$ and 0 otherwise, Andradóttir et al. show that stability is guaranteed for any system where $\lambda < \lambda^*$ if l_{jk} is chosen to be

$$l_{jk} = \left\lceil \frac{(1-\epsilon)(s + \sum_{i \in K} m_{ji} 1\{\delta > 0\})\delta_{jk}}{\epsilon m_{jk}} \right\rceil$$

By following the round robin policy, the δ_{jk} values will be sufficiently mimicked so that stability is guaranteed for any system for which stability is possible.

2.4.4 Discussion

The results of the round robin policy provide the queueing network with a method which can maximize the stable region. From the perspective of the operator, such a solution means that the current available resources (i.e., the servers) are being utilized so that obtaining further resources is not required when the system is subject to heavy loads unless the true arrival rate is above λ^*.

From the perspective of a customer who submits a job however, system performance is not very meaningful. Jobs are often concerned with their own waiting time in the system. Such a measure of schedule quality is not addressed in the work by Andradóttir et al. Given two methods, one which is stable and one unstable, we would expect the stable system will also have better waiting times than the unstable. Nevertheless, if two methods are both in a stable region, there is potential for significantly different performance.

Scheduling approaches often express performance measures relevant to jobs as objectives in their methods. Therefore, it is of interest whether scheduling or a combination of scheduling and queueing can be used to both achieve good/optimal system performance and optimize a scheduling criteria such as the jobs' waiting times.

2.5 Summary

In this chapter, we provided an overview of the literature on the two fields of queueing theory and scheduling. We also looked at the Benders decomposition technique and an extension due to Hooker. Finally, we looked at a queueing network with flexible servers and presented the round robin policy developed by Andradóttir et al. which maximizes the arrival rate the system can handle.

While there is research that studies scheduling in dynamic environments and scheduling in queueing environments, the work is not extensive. Specifically, scheduling in queueing environments has mainly been limited to simple dispatch rules such as SPT. Even in dynamic scheduling, focus is on uncertainty in the processing times or on machine failures.

In Chapters 3, we remove setup times and re-routing in the queueing network with flexible servers and use techniques from queueing theory and scheduling to improve upon current solutions. Chapter 4 examines a static parallel machine scheduling problem with setup times where we provide a logic-based Benders decomposition approach to finding schedules. Chapter 5 makes use of the model developed in Chapter 4 to include setup times to the queueing network and considers the system under queueing and scheduling models. The scheduling techniques provide the tools to solve combinatorial optimization problems which can often arise in queueing networks, whereas the queueing approaches deal with the dynamic characteristics. A combination of queueing and scheduling attempts to use the strengths of both research areas to more accurately represent real world problems which are both dynamic and require complex combinatorial reasoning.

Chapter 3

Scheduling a Queueing Network with Flexible Servers

In this chapter we investigate the use of scheduling in a dynamic queueing network. Specifically, we study a variation of the queueing network with flexible servers introduced in Section 2.4. We simplify the network by assuming that it has no setup times or rerouting. Andradóttir et al. [4] showed that efficient allocation of resources can be the difference between a stable and unstable system. We show that a hybrid technique produces schedules with better response time performance than the round robin policy and prompt departure of jobs from the system by using the long term goals of the round robin policy and short term optimization of scheduling.

In the following section, two different scheduling models are proposed for the queueing network problem. The first is a greedy dispatch scheduler whereas the second scheduling model is a periodic scheduler. Experimental results comparing the performance of the scheduling models and an existing policy are shown in Section 3.2. Section 3.3 proposes and evaluates a hybrid model that makes use of queueing analysis to improve one of the scheduling models. Section 3.4 is the conclusion to this chapter.

3.1 Scheduling Models

Andradóttir et al.'s problem is a queueing network with jobs belonging to one of K job classes. These jobs arrive to the system according to a Poisson process with rate λ and belong to a class k with probability a_k. Jobs are to be served by M servers which process in parallel with exponentially distributed rate μ_{jk}, the rate which server j serves a job of class k; Section 2.4 presents a detailed description of this problem. The goal of our adaptation to this problem is to assign jobs to servers to minimize the flow time in a queueing network with heavy loads. Given a_i and d_i, the arrival and departure times of a job i respectively, the flow time of job i is

$$ft_i = d_i - a_i$$

Andradóttir et al.'s objective was to maximize the stability region of a system by finding the greatest arrival rate the system can handle and create a policy which guarantees stability at that capacity. However, no explicit attention was paid to the performance metrics related to scheduling, such as flow time. In our work, we are concerned with both stability and scheduling metrics. By examining flow time performance of the system, we consider the timely processing of jobs as well as stability issues. Specifically, if the mean flow time converges to a finite value, the system is stable.

We study two scheduling models for this problem:

- The *MinFT* model is a dispatch policy that greedily assigns jobs to machines to minimize flow time.

- The *MinMksp* model is a periodic scheduler that assigns jobs to machines to minimize the makespan of each period using Mixed Integer Programming.

Both of these scheduling models make the assumption that the service time of a job i on server j, p_{ij}, is determined upon arrival. Service and arrival times of jobs that have not yet arrived remain unknown.

3.1.1 MinFT

The first scheduling model minimizes the flow time of jobs upon arrival to the system. When a job arrives, a decision is made to assign the job to a server. The choice is made by finding the server with the earliest completion time if the job is to be scheduled directly after all previously scheduled jobs on a server. Minimizing the completion time of a job is equivalent to minimizing the flow time of that job. The flow time of a job i if assigned to server j is denoted as f_{ij} and is

$$f_{ij} = r_j + \sum_{x \in \Gamma_j} p_{xj} + p_{ij}$$

The first term, r_j, represents the total remaining time that server j will be busy processing the job that is currently being serviced. $\sum_{x \in \Gamma_j} p_{xj}$ is the summation of the processing times of all jobs belonging to Γ_j, the set of queued jobs assigned to server j. If a server j is idle with no jobs queued when a decision must be made, the job can immediately enter service and complete in p_{ij} time units.

The assignment will be made such that,

$$ft_i = \min_j(f_{ij})$$

Where there are ties, the job will enter one of the tied queues arbitrarily.

3.1.2 MinMksp

The second scheduling model minimizes the makespan of specified time segments. The minimization is performed using Mixed Integer Programming (MIP) to assign jobs to servers during each segmented period. By minimizing the makespan, the scheduler aims to maximize the throughput of the system. The choice for minimizing the makespan rather than total flow time of a period is made for two reasons. Previous work by Terekhov et al. [103] successfully applied the periodic minimization of the makespan to a dynamic two machine flow shop and polling system to increase throughput and reduce flow time. In our problem, the minimization of makespan will maximize the period's throughput where we hope to obtain similar long

term results as Terekhov et al. The second reason for minimizing the makespan is because the problem is tractable for the problem sizes we are concerned with. Minimizing flow time to optimality during each period is impractical given time constraints.

3.1.2.1 Periodic Scheduling

Periodic scheduling is a method which can handle dynamic scheduling problems where job arrivals to the system are not known in advance. The *MinMksp* scheduler will look at the system at a point in time and schedule only the jobs that are present. The scheduler treats the system as though it were static. By performing this type of scheduling, objectives for each deterministic period can be optimized.

The *MinMksp* model minimizes the makespan of the static system during every period. Thus, the scheduler will complete all the known jobs in the fastest possible time. The schedule found to minimize the makespan will be executed until the next period begins. In the new period, the scheduler will again evaluate the system of all new jobs that are present. These jobs will be scheduled in the same manner to create the next period.

A period in the *MinMksp* model is defined as the time from when the scheduler evaluates the system and implements a schedule until any server is once again available. Any jobs belonging to a previous period but not yet completed will stay assigned to the servers they were scheduled on and await processing. Following such a period means that as soon as a server is free, a new schedule is made with the newly arrived jobs.

3.1.2.2 Mixed Integer Programming Model

At the beginning of every period, a MIP model is solved. This MIP model assigns all unscheduled jobs present in the system in order to minimize the makespan of the schedule. The MIP

model is

$$\begin{aligned}
\min \quad & C_{max} \\
\text{s.t.} \quad & \sum_{i=1}^{I} x_{ij} p_{ij} + r_j \leq C_{max}, \quad j \in M && (3.1) \\
& \sum_{j=1}^{M} x_{ij} = 1, && i \in I \quad (3.2) \\
& x_{ij} \in \{0, 1\}, && i \in I;\ j \in M \quad (3.3)
\end{aligned}$$

where

C_{max}: Makespan of the period

x_{ij}: 1 if job i is assigned to server j

r_j: The remaining time that server j is busy processing jobs belonging to schedules made during previous periods

I: The set of jobs to be scheduled during the period

This model minimizes the makespan, C_{max}, by constraining it to be at least as large as the maximum scheduled busy period of a server. Minimization is guaranteed by constraint (3.1). r_j is the amount of time that server j will be busy until it is able to process new jobs and $\sum_{i=1}^{I} x_{ij} p_{ij}$ is the total processing time allotted to the server by the new jobs. Added together, the two terms equal the total time the server will be busy by following the schedule. Constraint (3.2) enforces that each job be assigned to exactly one server and constraint (3.3) defines variables x_{ij} as binary variables.

3.1.2.3 Creating a Schedule

With the assignment of jobs to servers found by the MIP model, the order of jobs on each server still needs to be decided. Any order will achieve the same makespan since processing times of all jobs are sequence independent. A first come first serve (FCFS) policy is used once jobs have been allocated in order to make more direct comparisons with the Round Robin policy (Section 2.4) and attempt to ensure that jobs are given equal priority. All jobs that are assigned

to a server will be served in increasing order of arrival times. Where there are ties, a choice is made arbitrarily among the tied jobs.

3.2 Experimental Results

The Round Robin policy presented in Section 2.4 and scheduling models were empirically studied through the use of simulation to find stability and flow time performance. Two different cases were tested: two queue classes and two servers, and four queue classes and four servers. Parameters for both test cases are available in Appendix B.

The simulation was coded in C++, while the linear programming (LP) from the Round Robin policy and MIP model from the *MinMksp* model were coded in IBM ILOG CPLEX 12.1. All experiments were tested on a Dual Core AMD 270 CPU with 1 MB cache, 4GB of main memory, running Red Hat Enterprise Linux 4. For each of the test cases, 5 varied loads between 0.8 to 0.99 of the maximum theoretical load the system can handle were simulated. At each load, 20 instances were tested for 10,000 time units, to create a total of 200 simulations per model.

In both test cases, service rates were skewed to create an asymmetric system where servers were unique. Symmetric systems were not examined because maximizing stability can be achieved using any naive greedy algorithm, e.g., randomly assigning a job to a free server will achieve maximum capacity because a job served on any server has identical processing times. Maximum utilization of resources is attained when a server is minimally idled.

3.2.1 Processing Times Generation

With the service rates defined, a process for generating service times is needed. In a queueing theory analysis, defining a service rate and distribution is sufficient. Without dealing with the complex combinatorial problems of the system, the queueing analysis does not consider the actual processing times; as long as they are generated from the assumed distribution, the

queueing analysis is valid. For example, if a job enters a system with two servers and the service rate of the first server is twice that of the second server, there is no guarantee that the processing time from the two servers maintain this relationship. It is only guaranteed that the expected value of such a process is twice as fast on server one. However, the variation from expected times cannot be advantageously used without having the processing time information of a job on each server. We call the type of processing time generation, where processing times are independently generated for each job and server pair, *independent* processing time generation. The standard model in scheduling with unrelated machines assumes this method of generating processing times [39, 61].

To generate *independent* processing times p for an exponential distribution, we examine the cumulative distribution function of the distribution. If G is the cumulative distribution function of the exponential distribution, then the random variable U defined by $U = G(p)$ is uniformly distributed on the interval (0,1). Using the inverse of the cumulative distribution function, an exponentially distributed variable can be obtained from a uniformly distributed one. Thus, for every job i arriving to the system belonging to class k, a uniformly distributed variable U_j is generated for each server and processing times are calculated to be

$$p_{ij} = G^{-1}(U_j) = \frac{-\ln(U_j)}{\mu_{jk}}$$

Using the *independent* processing time generation will lead to an incorrect upper bound on the achievable arrival rate found by the Allocation LP. For example, examine a system with two servers and one class of jobs. The system is symmetric such that the service rate of the class is the same on both servers. If the service rate μ is an exponential distribution and arrival rate λ is a Poisson process, the load of the system is $L = \frac{\lambda}{2\mu}$.

Stability is guaranteed if the load is less than 1. However, if we assume that processing times are known, p_1 and p_2 for server 1 and 2 respectively and that these processing times are independent of the processing times on other servers, it is not true that the load is the same as before. If a job is always routed to the faster server of the two, the expected processing time

would be

$$\min(E[p_1], E[p_2]) = \frac{E[p_1]E[p_2]}{E[p_1] + E[p_2]}$$

Since the service rate are equal, the expected processing times are also the same which gives

$$\min(E[p_1], E[p_2]) = \frac{E[p_1]^2}{2E[p_1]}$$

Thus, the expected processing time is halved and the service rate is doubled if a job is always dispatched to the server which will serve the job fastest.

The increased service rate from using the minimum processing time changes the load of the system to

$$\frac{\lambda}{4\mu}$$

The load here is smaller for the same λ values which leads to increased stability of the system. With job processing times that are independent of times on other machines, following a policy like the *MinMksp* model that takes advantage of the anomalies from expectations, creates schedules that can handle a greater arrival to the system than is possible according to the queueing analysis.

Figure 3.1 illustrates how taking advantage of the processing time information leads to greater performance at higher loads. The mean flow time for each instance was recorded for the varied loads and an average of all the instances are graphed. Not only is the mean flow time of the system much lower at high loads, the performance implies that the system is stable farther past a load of 1 that was used. Within the range of loads tested, it is apparent that an asymptote does not exist leading to greater stability.

The assumption that the processing time for a job is independent of the requirements on other machines is unrealistic in most real world problems. It is more intuitive to assume the processing times for a job across machines are inter-dependent. Such an assumption also allows for more interesting comparisons between the Round Robin policy and scheduling models. Therefore, the simulation model generates an amount of work for each job, w_i, using an exponential distribution with rate 1, which is then augmented linearly by the service rate for

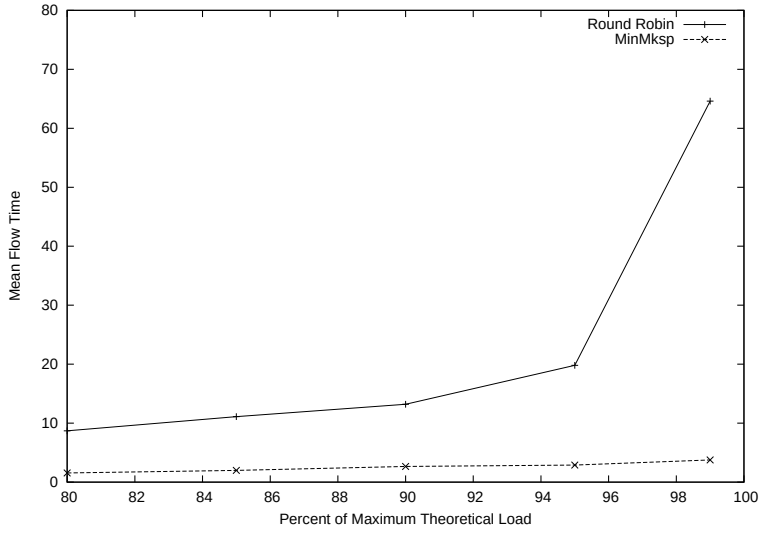

Figure 3.1: Comparison for a system with 2 servers and 2 class types under independent service time generation.

each machine, based on the job's class, to create the processing time on a machine $p_{ij} = \frac{w_i}{\mu_{jk}}$. The rest of our work assumes that the processing times of jobs are created in such a manner and will be known as the *dependent* processing time generation approach.

3.2.2 Performance Comparison

Figures 3.2 and 3.3 illustrate results from simulating the different models on both of the test cases with dependent service time generation. Similar to Figure 3.1, the graphs indicate an average mean flow time across all instances for the varied loads. In Figure 3.2, the two scheduling models can be seen to create better schedules than the Round Robin policy. At the lower loads, the *MinFT* model is able to obtain the lowest mean flow time performance of the three models. At higher loads, the two scheduling models maintain better performance than the Round Robin policy while surprisingly maintaining stability up to a load of 0.99. These results imply that experimentally the scheduling models are stabilizing the system.

Increasing the system size to four servers and four job classes creates substantially different results. Figure 3.3 shows that at lower loads, the Round Robin policy obtains the lowest mean flow time. This is in contrast to performing worst at all loads on the smaller system. As the load increases, the *MinMksp* model performance increases to be better than the Round Robin policy and the experimental results indicate that stability is being obtained at loads arbitrarily close to 1. However, the *MinFT* model shows very poor performance and instability past loads of 0.9. These results indicate that a scheduling algorithm that greedily attempts to minimize flow time cannot handle stability of a larger system at high loads. When stability is not attained, the performance degrades as jobs will be waiting longer and longer to be serviced; the mean flow time of the system explodes along with the queue sizes.

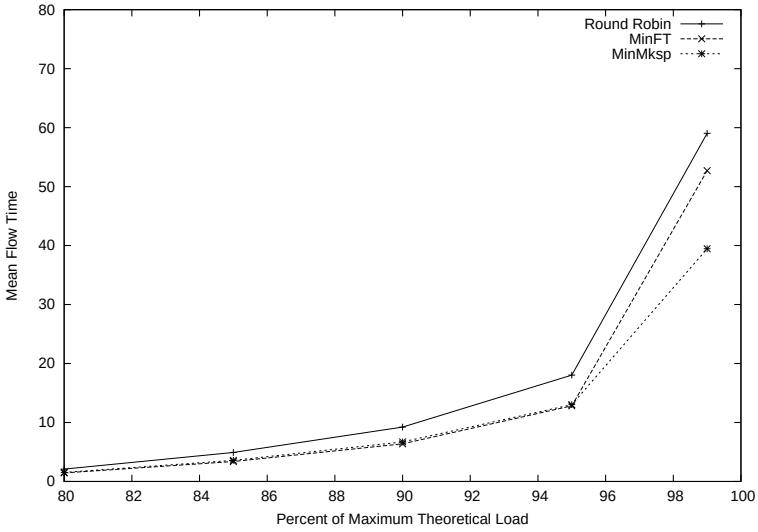

Figure 3.2: Comparison for a system with 2 servers and 2 class types.

The schedules produced from the simulation exhibit more information on the behaviour of the models than just mean flow time performance. In the queueing theory community, job classes with skewed priority can be of interest [95, 101, 104]. This same idea applies to scheduling where jobs may have a certain reward associated with their completion. While not

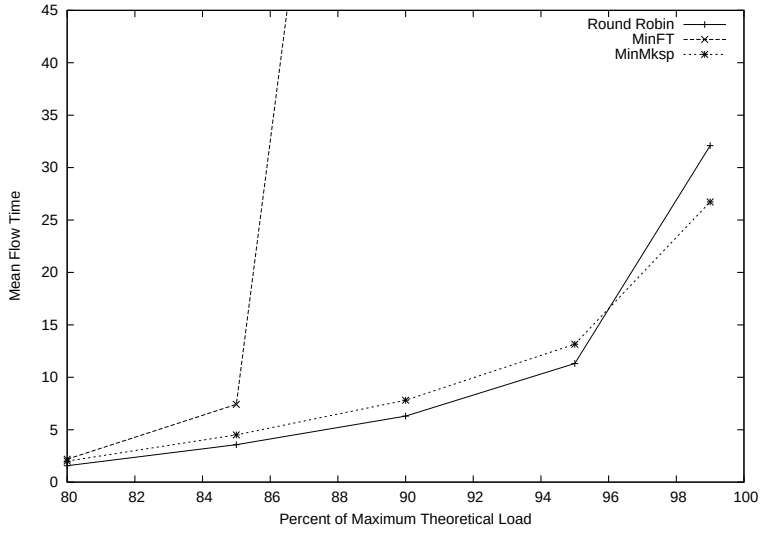

Figure 3.3: Comparison for a system with 4 servers and 4 class types.

directly considered as an objective in the queueing network studied, obtaining data on how fair a policy is to different customer classes can provide interesting insight. If classes have equal priority, a fair policy would attempt to equally minimize the flow time of all classes. Figure 3.4 and 3.5 show the variance of a model for mean flow time of each class. These figures show that as load increases, the variance of the system for the Round Robin policy drastically increases.

The large variance of the Round Robin policy occurs because the policy uses less information online about the state of the system. Such a policy may overcompensate to serve a job class immediately rather than delaying service to better distribute resources for a fairer allocation. In contrast, the scheduling models have very low variance because neither model considers job classes during the decision process. Schedules are made without information that a job belongs to a specific class. By disassociating jobs from classes, all classes are given equal priority. Having the ability to maintain equality between classes may be useful in many environments. If jobs are customers in a service oriented system, customers of a class that realizes longer delays will feel slighted and one can expect loss of good will. Even with a system

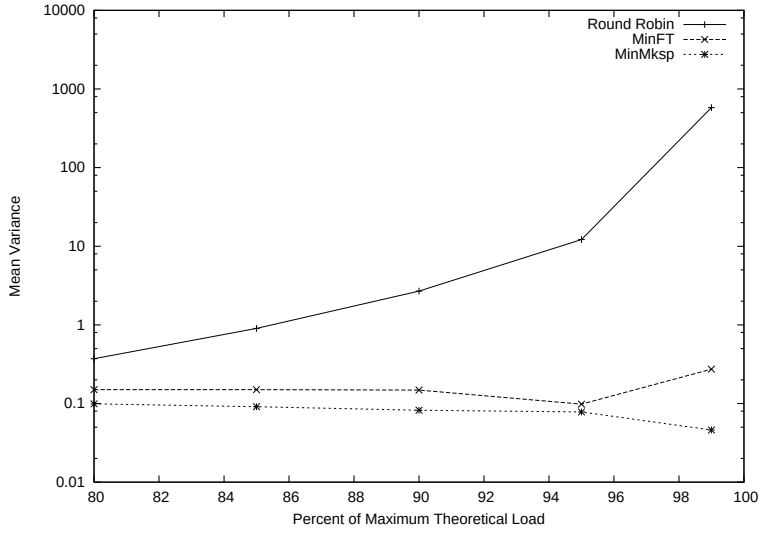

Figure 3.4: Class mean flow time variance for a system with 2 servers and 2 class types.

that has similar overall performance, the customers from a longer waiting class will be much less satisfied. Their loss in satisfaction can often be expected to be less than the gain from the classes that have lower delays.

If a situation were to arise where there are higher priority customers, the *MinMksp* model could partially accommodate the differently weighted classes. One method of doing so that does not rely on changing the model much is to assign customers to servers as done previously, but rather than using FCFS, serve customers of higher priority first. Sequencing based on priority allows for a decrease in flow time of higher priority customers with minimal changes to the model. Greater changes may yield better results if priority is a significant concern. One could sequence high priority customers first and then redefine periods such that a rescheduling occurs once all customers of high priority have been serviced. Rescheduling would delay the processing of low priority customers if new customers arrive.

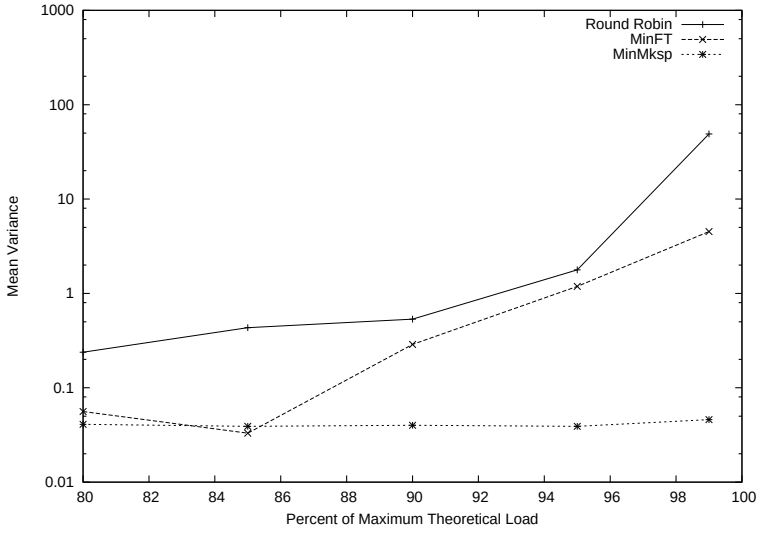

Figure 3.5: Class mean flow time variance for a system with 4 servers and 4 class types.

3.3 Combining Queueing Theory and Scheduling

The results in Section 3.2 clearly show the possible performance gains while still maintaining stability when using the *MinMksp* model. We are able to effectively maximize the throughput of a period by minimizing the makespan. While short term maximization of throughput does not guarantee the same in the long run, the model is still achieving stability in all instances of our simulation. Similar results were found by Terekhov et al. [103] where a periodic scheduler minimizing makespan is able to perform well at high loads by balancing long and short term objectives.

From the results of the larger system with four servers and four job classes, it is evident the Round Robin policy performs best at lower loads while the *MinMksp* model improves at very high loads. To investigate the crossover in performance, we study the effects when minimizing makespan during a period under high and low load conditions. Stability of the system and better flow time performance demonstrates that minimizing makespan is similar to maximizing

throughput as the load approaches 1. In fact, if we assume a static problem where all job arrivals are known a priori along with processing time information, minimizing the makespan maximizes throughput for any set period of time. As the load of a system increases, the queue lengths increase and result in larger period lengths that will closely mimic the omniscient static problem. Therefore, the greater the load of a system, the better the *MinMksp* model performs.

At lower loads, stability becomes less of a concern because maintaining control of the queue size is not difficult. However, efficient management of resources is still important to reduce the mean flow time of jobs. Surprisingly, the greedier *MinFT* and *MinMksp* models at lower loads are not able to perform as well as the Round Robin policy. We would expect the scheduling models to make use of faster servers to reduce the waiting time of jobs when a faster server, but the results show otherwise.

The two scheduling models presented do not account for the dynamic aspect of the queueing network properly and suffer in performance because of that. In the *MinFT* model, no thought is given to future arriving jobs. The *MinMksp* model attempts to reserve resources to future jobs, but does so naively by minimizing the makespan. One can see that there are situations where multiple different schedules will yield the same minimum makespan. If one server is assigned jobs in such a manner that it has the latest completion time among all servers, the rest of the servers can be scheduled in any way as long as their own respective latest completion time is equal to or earlier than the latest server. Consequently, multiple schedules exist which the *MinMksp* model cannot distinguish between as better or worse for efficient resource management when it creates these static schedules.

3.3.1 Hybrid Model

We present a hybrid model in order to make use of the benefits of scheduling and queueing theory. The ability of the *MinMksp* model to deal with the complex combinatorics allows it to take advantage of a system's state information while the Round Robin policy is able to use knowledge of service and arrival rate distributional properties to manage the allocation of

servers. Both tools are effective and have their own respective strengths. Adapting a model to gain the benefits of both may help to improve overall performance.

The hybrid model presented uses the *MinMksp* model as a framework. Like the scheduling model, the hybrid model is a periodic scheduler with a MIP solved during each period. Minimization of the makespan is still an objective for this MIP so that the performance of the *MinMksp* model is expected at heavy loads. We also make use of the allocation LP used in the Round Robin policy to obtain the δ_{jk} values which represent what the long run allocation of servers to classes should be. The δ_{jk} values help indicate which servers are more efficient on specific classes and are used to guide assignment decisions.

The MIP model used during each period is

$$\begin{aligned}
\min \quad & \alpha C_{max} + (1-\alpha)\sum_{j=1}^{M}\sum_{k=1}^{K} c_{jk} & & \\
\text{s.t.} \quad & \sum_{i=1}^{I} x_{ij}p_{ij} + r_j \leq C_{max} & j \in M & \quad (3.4)\\
& \sum_{j=1}^{M} x_{ij} = 1 & i \in I & \quad (3.5)\\
& \sum_{i \in k} x_{ij}p_{ij} - \delta_{jk}\sum_{i=1}^{I} x_{ij}p_{ij} \leq c_{jk} & k \in K;\ j \in M & \quad (3.6)\\
& c_{jk} \geq 0 & \forall j \in M;\ j \in M & \quad (3.7)\\
& x_{ij} \in (0,1) & \forall i \in I;\ j \in M & \quad (3.8)
\end{aligned}$$

where

c_{jk}: The amount of time that a realized assignment of server j to class k for a schedule is greater than what δ_{jk} suggests

α: A parameter used to scale the importance of deviation versus makespan

M: The set of machines

I: The set of jobs to be scheduled

K: The set of job classes

This new MIP model has a bi-criteria objective of minimizing the makespan and deviation from δ_{jk}. α is introduced in the objective function to determine the trade-off made between makespan and deviation. The value for α is an input parameter which can range between 0 and 1. When $\alpha = 0$, this scheduler becomes the *MinMksp* model. Constraints (3.4), (3.5), and (3.8) are the same as (3.1), (3.2), and (3.3). Constraint (3.6) assigns a penalty, c_{jk}, as the amount of time that a server j spends on a class k more than if the δ_{jk} values were followed exactly. Finally, Constraint (3.7) bounds c_{jk} to be non-negative.

Using this MIP model allows for the scheduler to maintain the priority of minimizing makespan, but where slack exists and multiple schedules obtain the same makespan, the model can distinguish the schedules based on their efficient allocation of resources. An allocation that accounts for future arrivals devotes δ_{jk} fraction of time during a schedule. Therefore, the amount of time that should be allocated for a j and k pair is $\delta_{jk} \sum_{i=1}^{I} x_{ij} p_{ij}$. That is, the total amount of work that is available during the creation of a schedule for a server and class pair adjusted by the fraction of work that should be assigned to that server.

Depending on the α value chosen, a schedule may not be optimal for makespan. If a sufficiently low value for α is used, the schedule found may sacrifice makespan in order to better mimic the δ_{jk} values. Therefore deciding a value for α is important in the performance of the model. If the value is too high, emphasis on queueing analysis will be too small. However, if the α value is too low, the makespan of a schedule will suffer.

Of interest to the queueing community is the stability of such a hybrid model. We know that the round robin policy maximizes the capacity the system can handle, but the same guarantees for the scheduling algorithms presented are not apparent. Terekhov et al. [102] prove that minimizing the makespan periodically leads to stability in a two-machine *flow shop* if the system is stabilizable. In their proof, the periodic scheduler is compared to a FCFS dispatch rule, which is known to maximize the capacity, and they show that the scheduler will reach a stable steady state if the FCFS policy can as well. We believe that a similar proof can be made for the *MinMksp* model in comparisons to the round robin policy. In this dissertation, a formal

proof is not presented, but we look at the intuition behind the stability of the scheduler.

The results in Section 3.2.2 imply that the *MinMksp* model does achieve maximum stability. As the load of the system increases, the size of the queue increases. The increase in the number of jobs leads to the *MinMksp* model acting like a fluid model. In the extreme case with an infinite number of jobs, solving the minimum makespan for a period of infinite length is equivalent to solving the fluid allocation LP. Therefore, as the size of the system grows, the *MinMksp* model tracks the allocation LP assignments. However, the same may not be true with the hybrid model. By trying to track the δ_{jk} values of the allocation LP, we allow for the makespan to be greater than the optimal value. Intuitively, one would expect that at very heavy loads, minimizing makespan and tracking the δ_{jk} values to be equivalent, but we can not provide a guarantee that it is so.

3.3.2 Experimental Results

We tested the hybrid model on the same instances as in Section 3.2. Different α values were tried in order to understand the effect it would have on performance for the larger system. Results of varying α are illustrated in Figure 3.6. The graph shows that using α values between 0.5 and 0.9 all led to good performance; the differences were minimal up to a load of 0.9. However, performance quickly deteriorated as α was lowered to 0.4 and below. In our experiments for this particular system, it was found that $\alpha = 0.6$ had the best performance across all loads. Therefore, the results comparing the hybrid model with the Round Robin policy and *MinMksp* model make use of an α value of 0.6.

Figures 3.7 and 3.8 compare the hybrid model to the Round Robin policy and *MinMksp* model. The results demonstrate that the hybrid model performs better than the *MinMksp* model at all tested loads. For a system with two servers and two classes, the hybrid model is the best performing scheduler. Increasing the size of the problem, the Round Robin policy and hybrid model show equivalent performance at lower loads. Using the hybrid model increases the *MinMksp* model performance to that of the Round Robin policy. The hybrid performance is

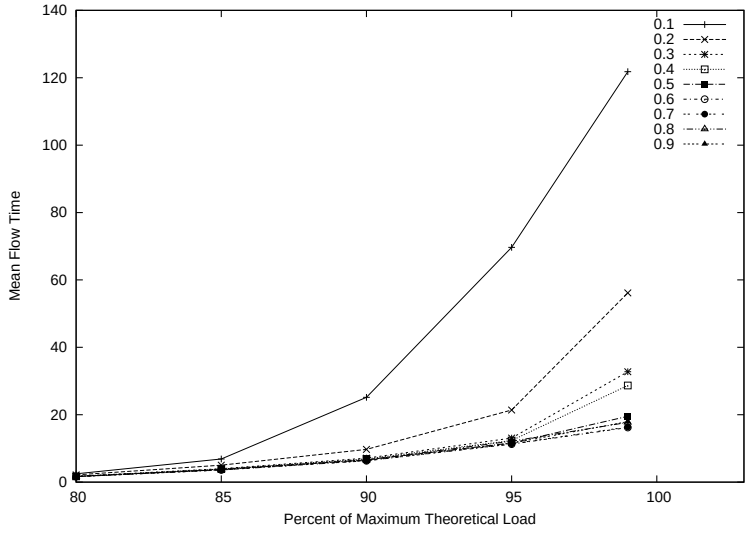

Figure 3.6: Performance of different α values for a system with 4 servers and 4 class types.

much better than the other two models when increasing the loads closer to 1.

The results show that guiding the *MinMksp* model with the queueing analysis greatly affects performance. Incorporating the δ_{jk} values in the scheduler helps reduce the mean flow time and allows the hybrid model to closely follow or outperform the previous best. One can conclude that the hybrid model does not only obtain the best performance, but also is a robust approach to scheduling. At all loads, the hybrid model maintains a competitive mean flow time and is able to either track or beat the other best performing models.

Finally, Figure 3.9 presents the variance of the mean flow time across all job classes for the larger system. Previous results indicated that using the Round Robin policy led to different priority for each job class and increased variance. The hybrid model is able to maintain a low variance even with the inclusion of class specific information.

To ensure that guiding the scheduling model with the allocation LP values is why the model is performing better, we alter the δ_{jk} values and simulate again. If following the queueing analysis is what we should do, deviating from them should guide us to worst schedules. Therefore,

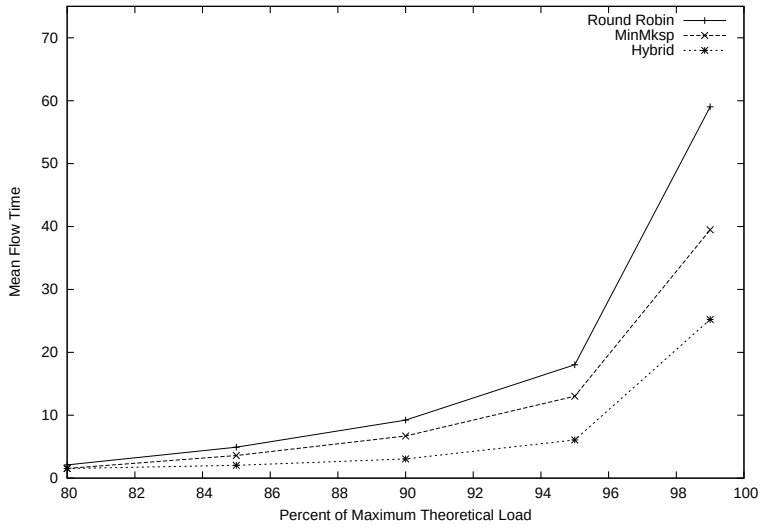

Figure 3.7: Comparison for a system with 2 servers and 2 class types.

we replace all δ_{jk} values with $1 - \delta_{jk}$. The more resources which the allocation LP assigns to a class, the less will be assigned in the hybrid. Additionally, we use $\alpha = 0.9$ to ensure that the guiding effect does not interfere greatly with the makespan. Figure 3.10 shows how a misguided hybrid has poor performance. Especially as the load of the system increases, the misguided hybrid is not able to maintain stability like the properly guided hybrid.

3.4 Conclusion

In this chapter, two scheduling approaches are proposed for a queueing network problem with flexible servers. These approaches are compared to a policy developed in the queueing theory literature. Simulation showed that the scheduling and queueing theory approaches have their own respective strengths for maintaining stability and reducing flow time; the scheduling model is able to take advantage of information at the operational level and perform optimization while the queueing theory approach uses high level knowledge of the system to create

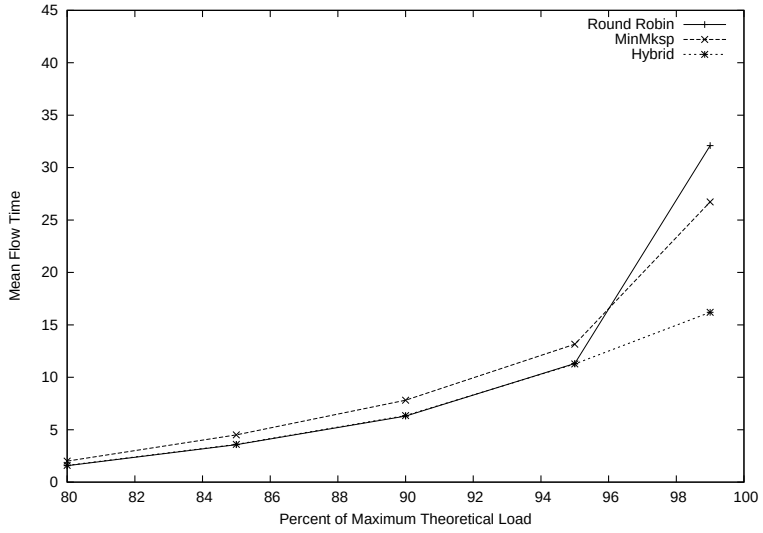

Figure 3.8: Comparison for a system with 4 servers and 4 class types.

smart decisions at a strategic level. Based on the experimental results, a hybrid model is formulated which attempts to combine the strengths and ideas of scheduling and queueing theory. Through use of the hybridized model, performance gains are observed which improve the flow time of a scheduling approach at all tested loads. Further, the hybrid model is able to achieve a performance equal to that of the queueing theory policy at all loads where such a policy has the lowest mean flow time.

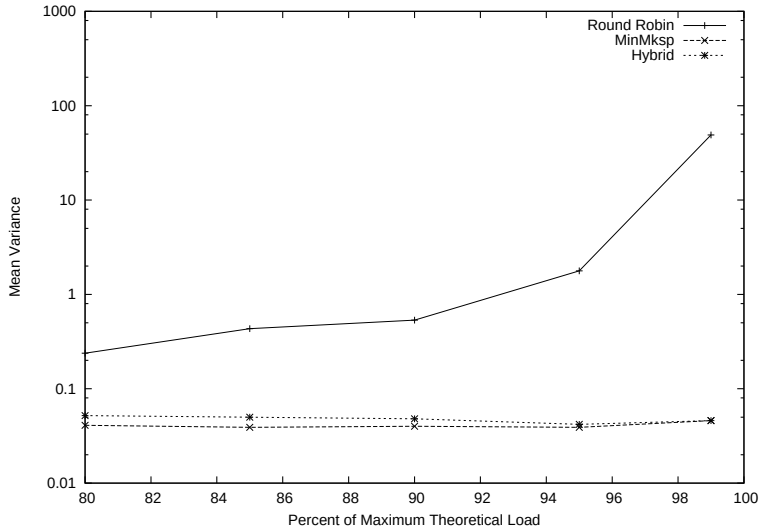

Figure 3.9: Class mean flow time variance for a system with 4 servers and 4 class types.

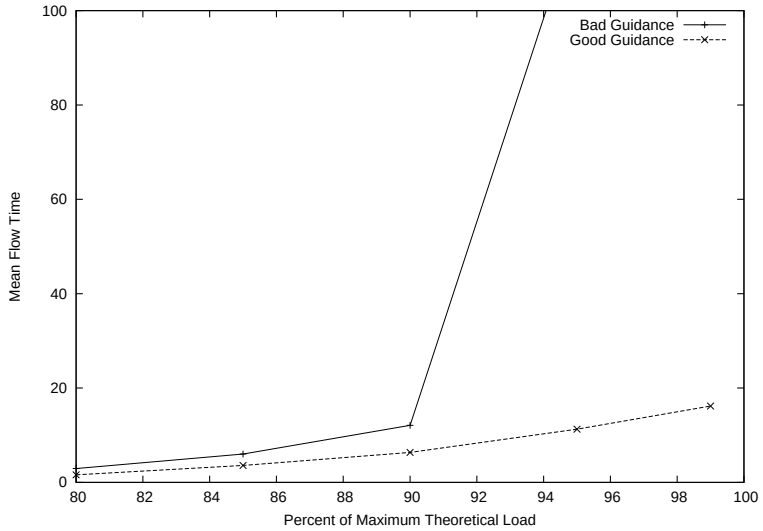

Figure 3.10: Mean flow time for a system with 4 servers and 4 class types.

Chapter 4

Scheduling Alternative Resources with Setup Times

In the previous chapter, we investigated a queueing network with flexible servers. The problem studied ignored setup times that may exist when a server switches between jobs of different classes. Setup times create a complex problem where minimizing makespan is not as simple as the *MinMksp* model. This chapter focuses on a static scheduling problem where setup time exists between jobs with the goal of minimizing the makespan of a schedule. We will revisit the dynamic scheduling problem in Chapter 5 making use of the techniques developed in this chapter.

In many practical scheduling problems, the scheduler is faced with alternative machines and sequence dependent setup times. That is, a job may be assigned to one of a set of machines and a minimum time must elapse between consecutive jobs on the same machine. For example, reactors in a chemical plant must be cleaned when changing from processing one mixture to another. The cleaning times may depend on which job comes before the cleaning and which comes after. If the preceding chemical affects the succeeding one, cleaning may take longer to ensure that the reactor is properly prepared. The products processed in the reverse order may not have the same ill effects because the offending product in the previous example may not

suffer the same contamination and so cleaning may take less time. Further examples can be found in the plastic, glass, paper and textile industries where setup times of significant length exist [36]. Allahverdi et al. [3] review the importance of setup times in real world problems.

The problem described is the unrelated parallel machines scheduling problem (PMSP) with machine and sequence dependent setup times. In the PMSP, jobs must be assigned to one of a set of alternative resources where jobs assigned to the same resource have a setup time defined as the time that must elapse between the end of one job and the start of the next. The setup time is sequence and machine dependent in that the elapsed time between jobs will differ depending on what order and to which machine the pair of jobs are assigned. We concern ourselves with minimizing the makespan or the maximum completion time of a schedule as the objective function. Using the three-field notation given by Graham et al. [43], this problem can be denoted as $(R|sds|C_{max})$.

In this chapter, we develop an exact method to solve the PMSP using a logic-based Benders decomposition approach. The Benders decomposition makes use of a Mixed Integer Program (MIP) master problem and either a Constraint Program (CP) solver or a specialized Traveling Salesman Problem (TSP) solver for the subproblems. The MIP model is used to find tight lower bounds on the makespan and machine assignments, while the CP or TSP solvers sequence the jobs on machines. Finally, a relaxation on optimality is imposed on the Benders decomposition to allow the model to find feasible schedules for larger problems. Although the model is no longer an exact method, the performance gains make the incomplete algorithm attractive.

4.1 Background

In this section, we will define the PMSP with sequence and machine dependent setup times. A review of related work is presented and finally, an existing MIP model to solve the PMSP is shown.

4.1.1 Problem Definition

In the PMSP, a set of jobs, N, are to be scheduled on a set of machines, M, with the objective of minimizing the makespan. Each job has a processing time p_{ij}, the time to process job j on machine i. The machines in this system are unrelated, i.e., a job j can have a processing time greater than job k on one machine, but the reverse could be true on another machine. There is a sequence and machine dependent setup time, s_{ijk}, which is the time that must elapse before a machine can begin processing job k if job j precedes it on machine i. The setup times are assumed to follow the triangle inequality $s_{ijk} \leq s_{ijl} + s_{ilk}$ and are only incurred when switching from one job to another. The goal of the problem is to determine how to assign jobs to machines and then sequence these jobs, on each machine.

4.1.2 Related Work

Most research has been focused on the PMSP with identical machines [22, 32, 44, 63]. Research on the PMSP with unrelated machines has concentrated on the problem without setup times. An exact algorithm was developed by Lancia [65] to minimize makespan for a problem with 2 parallel machines. Lancia devised a branch and bound algorithm that was an extension of an algorithm by Carlier [19] for solving a single machine problem. A genetic algorithm, simulated annealing, and tabu search were compared by Glass et al. [39]. They conclude that tabu search is capable of finding the best solutions in short periods of time (20 seconds), but the performance comparison for a larger time limit (100 seconds) shows no clear distinction between the three local search methods.

The PMSP with sequence and machine dependent setup times is strongly NP-Hard because the single machine scheduling problem with sequence dependent setup times ($1|\text{sds}|C_{max}$) is equivalent to a traveling salesman problem (TSP) [8]. Thus, the PMSP with setup times can be thought of as an allocation and routing problem where cities are allocated to salesmen who then must find their own tour. Even in the case where all machines are identical ($P|\text{sds}|C_{max}$),

the problem is strongly NP-hard [36, 73]. The combinatorial complexity of the PMSP has resulted in little research in exact optimization methods. A MIP model does exist [47] which obtains the optimal schedule for the PMSP with setup times with total completion time or total tardiness as objectives. However, this MIP model is only able to solve for small instances: 9 jobs, 2 machines or 8 jobs, 4 machines. The sizes of these problems are not very practical for real life use where the number of jobs can be much larger.

Al-Salem [2] developed the *partitioning* heuristic to solve large instances of the PMSP with setup times. The *partitioning* heuristic made use of a constructive and improvement heuristic to assign jobs to machines and a TSP-like heuristic to sequence them. Rabadi et al. [87] presented a metaheuristic titled Meta-RaPS. The heuristic is a modified COMSOAL (Computer Method of Sequencing Operations for Assembly Lines) [5] approach with the goal of minimizing makespan. In their paper, direct comparisons were made between Meta-RaPS and the *partitioning* heuristic on problems up to 120 jobs and 12 machines and Meta-RaPS was found to be 10% better in some cases. Helal et al. [49] developed a tabu search to solve the same problem and also compared their model to the *partitioning* heuristic. The tabu search was found to be up to 8% better than the *partitioning heuristic* on the same sized instances as Rabadi et al. An ant colony optimization (ACO) method was implemented and shown to perform better then the partitioning heuristic, tabu search, and Meta-RaPS for the unrelated PMSP with setup times [6]. In this study, all four heuristics were tested on instances up to size of 120 jobs and 8 machines where the performance was based on the deviation of the makespan found to that of a lower bound. ACO was found to be within 1% to 7% deviation of the lower bound with Meta-RaPS close behind.

Focacci et al. [35] proposed a two-phase algorithm based on CP to optimize both the makespan and the sum of setup times for a similar problem to the PMSP with sequence dependent setup times. In this paper, jobs consist of multiple activities and precedence constraints exist between these activities. In the first phase of the algorithm, a time limited, incomplete branch-and-bound method is used to find solutions with small makespan. The second phase

minimizes the sum of setup times with the constraint that any schedule found must have a makespan equal to or less than the makespan found in the first phase. They run their model on problems of up to 16 jobs, each consisting of 12 activities, and 16 machines.

4.1.3 Mixed Integer Programming Model

A MIP model used to find optimal solutions for the unrelated PMSP with setup times is presented by various researchers [49, 87]. This formulation is based on a similar problem by Guinet [47] with different objectives of total completion time or total tardiness.

$$
\begin{aligned}
\min \quad & C_{max} \\
\text{s.t.} \quad & \sum_{j=0,j\neq k}^{N} \sum_{i=1}^{M} x_{ijk} = 1, && k \in N && (4.1)\\
& \sum_{j=0,j\neq h}^{N} x_{ijh} = \sum_{k=0,k\neq h}^{N} x_{ihk}, && h \in N;\, i \in M && (4.2)\\
& C_k \geq C_j + \sum_{i=1}^{M} x_{ijk}(s_{ijk} + p_{ik}) + V(\sum_{i=1}^{M} x_{ijk} - 1) && && \\
& && j \in N;\, k \in N && (4.3)\\
& \sum_{j=0}^{N} x_{i0j} = 1, && i \in M && (4.4)\\
& C_j \leq C_{max}, && j \in N && (4.5)\\
& C_0 = 0, && && (4.6)\\
& C_j \geq 0, && j \in N && (4.7)\\
& x_{ijk} \in \{0;1\}, && j \in N, && \\
& && k \in N, i \in M && (4.8)
\end{aligned}
$$

where

C_{max}: Maximum completion time (makespan)

C_j: Completion time of job j

x_{ijk}: 1 if job k is processed directly after job j on machine i

x_{i0k}: 1 if job k is the first job to be processed on machine i

x_{ij0}: 1 if job j is the last job to be processed on machine i

s_{i0k}: setup time before job k if it is the first job on machine i

V: A large positive number

Constraint (4.1) ensures that each job is scheduled on a single machine only and after exactly one other job. Constraint (4.2) ensures that each job cannot be preceded or succeeded by more than one job. Constraint (4.3) sets the completion times of each job such that if job j precedes job k, job k cannot also precede job j to create an infeasible cycle. If job k is processed directly after job j, $\sum_{i=1}^{M} x_{ijk} - 1 = 0$ and the constraint makes it so that $C_k \geq C_j + s_{ijk} + p_{ik}$ with i being the machine on which the two jobs are assigned. If job k is not scheduled directly after job j on a machine, $\sum_{i=1}^{M} x_{ijk} - 1 = -1$ and the large V term makes the constraint redundant. Constraint (4.4) guarantees that only one job can be scheduled first on each machine. Constraint (4.5) sets the makespan to be at least as large as the largest completion time of all jobs. Constraint (4.6) sets the completion time of job 0, an auxiliary job used to enforce the start of a schedule, to zero and constraint (4.7) ensures positive completion times. Constraint (4.8) defines the decision variables as binary.

The MIP model was used to benchmark the quality of heuristics for small problem instances. Previous researchers solved problems of 8 jobs and 4 machines and 9 jobs and 2 machines [49, 87]. They found that larger problems were intractable and the MIP model could not be used to find optimal schedules anymore. For example, Helal et al. found the problem with 9 jobs and 2 machines required 5 hours to find solutions on average.

4.2 Logic-Based Benders Decomposition

The PMSP with setup times can be decomposed into an assignment master problem and sequencing subproblem. In the assignment master problem, the jobs are assigned to machines. This assignment results in multiple subproblems where each machine is a scheduling problem to sequence the assigned jobs. A MIP model is presented for the master problem. The use of MIP takes advantage of operations research tools to obtain tight lower bounds on the PMSP with setup times. Sequencing is accomplished in the subproblem where CP and TSP solvers are more adept at obtaining answers quickly.

4.2.1 Assignment Master Problem

The MIP formulation of the master problem is a relaxation of the PMSP with setup times. In this relaxation, jobs are assigned to machines, but instead of solving for a single sequence of jobs on each machine, many smaller subsequences are allowed. The setup times are calculated for each subsequence; their sum is a lower bound on the actual total setup time on a machine. This assignment and subsequencing leads to an infeasible schedule if multiple subsequences are created. However, the relaxation gives a tighter lower bound than if setup times are completely ignored, while being significantly less difficult than the full problem. The master

problem is

$$
\begin{aligned}
\min \quad & C_{max} \\
\text{s.t.} \quad & \sum_{j \in N} x_{ij} p_{ij} + \xi_i \leq C_{max}, && i \in M && (4.9) \\
& \sum_{i \in M} x_{ij} = 1, && j \in N && (4.10) \\
& \xi_i = \sum_{j \in N} \sum_{k \in N, k \neq j} y_{ijk} s_{ijk}, && i \in M && (4.11) \\
& x_{ik} = \sum_{j \in N} y_{ijk}, && k \in N;\ i \in M && (4.12) \\
& x_{ij} = \sum_{k \in n} y_{ijk}, && j \in N;\ i \in M && (4.13) \\
& cuts && && (4.14) \\
& x_{ij} \in \{0; 1\}, && j \in N;\ i \in M && (4.15) \\
& 0 \leq y_{ijk} \leq 1, && j, k \in N;\ i \in M && (4.16)
\end{aligned}
$$

where

C_{max}: Makespan of the master problem

ξ_i: Total setup time incurred from all sequences on machine i

x_{ij}: 1 if job j is processed on machine i

y_{ijk}: 1 if job k is processed directly after job j on machine i

y_{i0k}: 1 if job k is processed first on machine i

y_{ij0}: 1 if job j is processed last on machine i

The makespan on each machine with the relaxed setup times is defined in constraint (4.9) as the summation of processing times for all jobs that are assigned to that machine and the relaxed total setup times. Setting the makespan to be greater than or equal to the relaxed makespan of each machine enforces that the MIP model optimizes the maximum makespan across all machines. Constraint (4.10) ensures that each job is assigned to exactly one machine. Constraint (4.11) assigns the relaxed setup time of a machine i, ξ_i, to be a lower bound on the additional time required from the sequencing of jobs, y_{ijk}, and their respective setup times, s_{ijk}. The relaxation of setup times allows, instead of a sequence of jobs from the first to last

job processed on a machine, many smaller sequences independent of each other. For example, given jobs j, k, j_3, j_4, and j_5, a feasible sequence is [*start* - j - k - j_3 - j_4 - j_5 - *end*]. However, (4.12) and (4.13) set each job to have exactly one other job scheduled directly before and after it without the restriction of cycles as was seen in constraint (4.3). This will make it possible to assign two sequences, [*start* - j - k - j_3 - *end*] and the cycle [j_4 - j_5 - j_4 - j_5...]. Constraint (4.14) are *cuts* added to the master problem from the subproblem each time an infeasible solution is found. In the first iteration of the master problem, the set of *cuts* is empty. The last constraints, (4.15) and (4.16), force the decision variables x_{ij} to be binary, i.e., either a job j is assigned machine i or not and y_{ijk} to be between 0 and 1.

The master problem formulation is equivalent to solving the (R|sds|C_{max}), but instead of solving for the exact single sequence of jobs to process on a machine, many subsequences are allowed which will include all jobs. The relaxation creates a tight lower bound for the actual makespan of a machine and is similar to solving the assignment problem. Therefore, the makespan found from solving the master problem may be infeasible given a proper sequencing of jobs.

4.2.2 Sequencing Subproblem

Once a solution of the master problem is found, the set of jobs to schedule on each machine is known. These sets of jobs create $|M|$ subproblems, one for each machine. In this section, two different subproblem formulations are presented: a CP and a TSP model. Both models will create a sequence of jobs on a machine such that the makespan is minimized. This objective can also be thought of as minimizing the sum of setup times since the set of jobs to be processed and their processing times are already determined.

4.2.2.1 Constraint Program

Let t_j and e_j be the start and end times for job j respectively. C_{max}^{hi} represents the makespan for machine i in iteration h. The CP formulation of the subproblem is shown below.

$$
\begin{aligned}
\min \quad & C_{max}^{hi} & & \\
\text{s.t.} \quad & t_j + p_{ij} = e_j & j \in N' & \quad (4.17)\\
& t_j \geq e_k + s_{ikj} \vee t_k \geq e_j + s_{ijk} & k, j \in N';\, k \neq j & \quad (4.18)\\
& t_0 = 0 & & \quad (4.19)\\
& \text{disjunctive}(t_j, p_{ij}) & & \quad (4.20)\\
& C_{max}^{hi} \geq e_j & j \in N' & \quad (4.21)
\end{aligned}
$$

The objective of the subproblem is to sequence the jobs in such a way as to minimize the makespan. The set of jobs to schedule is N' which are the jobs chosen in the master problem with an additional auxiliary job (job 0) which has setup times and processing times equal to 0. This extra job acts as the first job in the sequence which incurs no setup times for the job that is scheduled after it. Constraint (4.17) sets the end time of each job to be the start time plus the processing time. Constraint (4.18) ensures that setup times are adhered to. If a job k is processed directly after a job j, then constraint (4.18) becomes an equality constraint. In the case where job k is not directly processed after a job j, but still processed later, the constraint holds given that setup times adhere to the triangle inequality. Constraint (4.19) sets the start time of job 0 to zero. Constraint (4.20) is a global constraint that ensures the unary resource constraint is satisfied such that only one job is allowed to be processed on a server at any point in time. Finally, contraint (4.21) sets the makespan of the schedule to be equal to the latest completion time of all jobs.

We implement this model in IBM ILOG Scheduler 6.7. Each job is an activity in IBM ILOG Scheduler, represented by the variables t_j and e_j. The machine that these jobs are assigned to is represented by a unary resource and the setup times, s_{jk}, are assigned as the elements of an IloTransitionParam matrix. The IloTransitionParam matrix is an asymmetric square matrix that defines the sequence dependent setup times in IBM ILOG Scheduler for each activity pair.

4.2.2.2 Traveling Salesman Problem

We know that the sequencing of jobs on a single machine is equivalent to a TSP with directed edges, also known as an asymmetric TSP. In the TSP representation, jobs are the nodes and distances between nodes are the setup time between the two connected jobs and the processing time of the job which is the start node. Therefore, traveling along any edge from node a to node b will contribute the cost of processing job a and the setup time from job a to job b. By representing the problem in such a way, it is possible to see that the setup time problem on a single machine is equivalent to a TSP and a cycle of a TSP from a start node to all other nodes and back to the initial start node is the sequence of jobs to process and the distance traveled being the makespan. The TSP representation is shown in Figure 1.

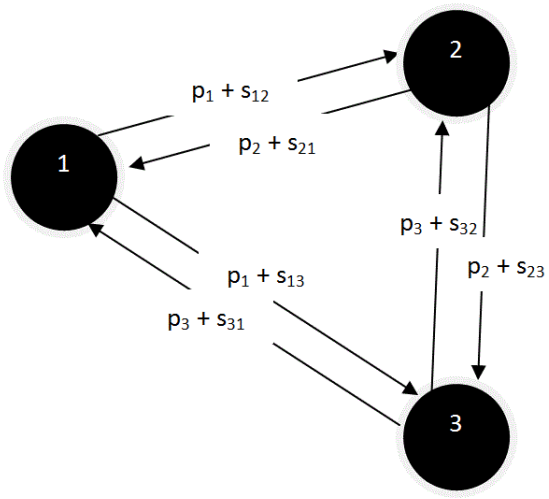

Figure 4.1: TSP representation

It can be seen that if the order of jobs to be processed is 1, 3 then 2, the distance traveled would be, $p_1 + s_{13} + p_3 + s_{32} + p_2$. This tour distance is equal to the makespan of processing jobs in that order. Therefore, it may be better to use a TSP solver which has been optimized to solve such problems in place of a generic CP solver which is more expressive, but not as good at solving this problem.

4.2.2.3 Feasible Schedules from the Subproblem

Solving the sequencing subproblem leads to a feasible schedule. Often with Benders decomposition, a model searches in the infeasible region of solutions and the first feasible solution found is the optimal one. In our logic-based Benders decomposition approach, while the search is performed in the infeasible region, the sequencing subproblem solves the local schedule once the jobs are allocated to machines. The solution from the subproblem creates a feasible schedule with a makespan equal to the largest makespan found across all subproblems. Therefore, it is possible to stop the solving at any time after the first complete iteration and obtain a feasible schedule. This schedule may not be optimal if the problem solving is stopped prematurely. However, it is possible to compare this value against the makespan found from the most recent master problem solution to calculate an upper bound on how far the current schedule is from optimality. Therefore, the Benders decomposition will store the best solution found so far. At the completion of all subproblems during an iteration, the schedule created will be compared to the best solution found so far and it will be updated if necessary.

4.2.3 Cuts

If the makespan found in the subproblem is less than or equal to the master problem's makespan C_{max}, then this subproblem is feasible and no cuts are added to the master problem. In the case where the makespan found is greater than C_{max}, a cut is created and sent to the master problem. The master problem is then re-solved with the added cut. The cut from such a subproblem in an iteration h is

$$C_{max} \geq C_{max}^{hi*}[1 - \sum_{j \in N'}(1 - x_{ij})] \quad i \in M \quad (C1)$$

Here, C_{max} is the makespan variable in the master problem and C_{max}^{hi*} is the makespan found in iteration h when solving the subproblem for machine i. The cut states that the future solutions of the master problem can only decrease the makespan if another assignment of jobs is given. That is, if the same assignment is given to the subproblem, the x_{ij} variables that are

part of this cut will all equal to 1. If this is the case, then $(1 - x_{ij}) = 0$ for all j and the makespan of the subproblem becomes a lower bound on C_{max}. When a different assignment is made and at least one of the x_{ij} variables that previously had a value of 1 is 0, the cut becomes redundant as the right hand side will be at most zero. This cut follows the two conditions defined by Chu and Xia [23] to be a valid cut: the cut removes the current solution from the master problem and does not eliminate any global optimal solutions.

The cut presented is a type of *nogood* cut [51], stating that the current solution is infeasible and so is removed from the search space. We can tighten the cut by introducing the values minPre_{hj}, minSuc_{hj}, maxPre_{hj}, maxSuc_{hj} and $\mathit{minTran}_{hj}$. minPre_{hj} and maxPre_{hj} are the minimum and maximum setup times if job j directly succeeds another job that is assigned to the same machine in iteration h respectively. Similarly, minSuc_{hj} and maxSuc_{hj} are the minimum and maximum setup times if job j directly precedes another job that is assigned to the same machine in iteration h. Finally, $\mathit{minTran}_{hj}$ is the minimum setup time between any two jobs in N' excluding job j. These values are defined as,

$$\begin{aligned}
\mathit{minPre}_{hj} &= \min_{k \in N'; k \neq j} (s_{ikj}) \\
\mathit{minSuc}_{hj} &= \min_{k \in N'; k \neq j} (s_{ijk}) \\
\mathit{maxPre}_{hj} &= \max_{k \in N'; k \neq j} (s_{ikj}) \\
\mathit{maxSuc}_{hj} &= \max_{k \in N'; k \neq j} (s_{ikj}) \\
\mathit{minTran}_{hj} &= \max_{k \in N'; l \in N'; k \neq l} (s_{ikl})
\end{aligned}$$

A proposed better cut could use more information to find a tight upper bound on the total reduction of the makespan when a job is removed from a schedule. To find such an upper bound, we work backwards and calculate the increase in the makespan if job j is added to an optimal sequence consisting of the jobs in N' except j. Assume that the optimal makespan of the schedule, C', is known. If job j is inserted into the schedule greedily, the insertion may occur at either a position that minimizes the setup time to job j or from job j. In the worst case scenario, if the insertion that occurred to minimize the setup to (from) job j, the job that

succeeds (precedes) j may result in the largest setup time from (to) job j. This insertion will further remove a single setup time since job j may be scheduled between any two other jobs. In the worst case, the removed setup time is the minimum setup time between any two jobs in N'. We know that the insertion results in a feasible schedule and in the best case will give the optimal makespan for scheduling N'. The increase in makespan of adding job j is denoted as θ_{hij}. We know that

$$C_{max}^{hi*} \leq C' + \theta_{hij}$$

which rearranged is

$$C' \geq C_{max}^{hi*} - \theta_{hij}$$

Therefore, if the optimal makespan of a schedule consisting of all jobs in N' is known, removing θ_{hij} from the optimal makespan is guaranteed to result in a makespan less than or equal optimal if job j is removed. The cut can then be

$$C_{max} \geq C_{max}^{hi*} - \sum_{j \in N'} (1 - x_{ij})\theta_{hij} \quad i \in M \quad (C2)$$

where,

$$\theta_{hij} = p_{ij} - \textit{minTran}_{hj} + \min[(\textit{minPre}_{hj} + \textit{maxSuc}_{hj}), \textit{maxPre}_{hj} + \textit{minSuc}_{hj}]$$

A third cut, (C3), is developed to further improve upon (C2). (C3) attempts to not only include extra information about the jobs being assigned, but also account for the jobs that are not assigned to the subproblem machine. This is achievable by incrementing the makespan by the processing time of a job if that job is to be assigned to a machine. The cut is then

$$C_{max} \geq C_{max}^{hi*} + \textstyle\sum_{k \notin N'} x_{ik} p_{ik} - \sum_{j \in N'} (1 - x_{ij})\theta_{hij} \qquad i \in M \quad (C3)$$

4.2.4 Stopping Condition

The Benders approach will iterate between master problem and subproblems until an optimal solution is found and proved. Optimality is proven if either of two conditions is met. The first condition that can prove optimality is if all subproblems solved during an iteration find makespans less than or equal to the C_{max} from the master problem. The solution is optimal because the master problem provides a lower bound on the achievable makespan of the problem. If all subproblems prove that a schedule can be created with the makespan less than or equal to C_{max} of the master problem, it is proven that the schedule is optimal. The second condition that can prove optimality and provide a stopping condition is that the C_{max} found from the master problem be equal to the best feasible makespan found so far as defined in Section 4.2.2.

4.3 Computational Results

The Benders decomposition model and MIP model were tested on an Intel Pentium 4 CPU 3.00GHz Hyperthread Tech with 2 MB cache per core, 1 GB of main memory, running Red Hat 3.4.6-3. The MIP master problem and MIP model were implemented with IBM ILOG CPLEX 12.1 and the CP subproblem was implemented with IBM ILOG Solver 6.7 and IBM ILOG Scheduler 6.7. The TSP solver used was tsp_solve.[1] Experiments were run for problem instances of 10, 20, 30, and 40 jobs. For each job size, between 2 and 5 machines were tested. Each of these combinations had a total of 10 instances for a total of 160 instances. A time limit of 3 hours was used. Processing times for each machine job pair were generated from a uniform distribution between 1 and 100. To obtain setup times that were sequence dependent and follow the triangular inequality assumption, each job was given two different sets of coordinates on a Cartesian plane for every machine. The setup times are the Manhattan distances from one job's coordinates to the other's. Distances between the second set of coordinates is used to

[1] A free TSP solver available online at (http://www.or. deis.unibo.it/research_pages/tspsoft.html) which is written in C++.

provide asymmetric setup times. For example, job 0 and job 1 would be given coordinates χ_{0a}, χ_{0b}, ψ_{0a}, ψ_{0b}, χ_{1a}, χ_{1b}, ψ_{1a}, and ψ_{1b}. Setup time from job 0 (1) to job 1 (0) would then be $|\chi_{0a} - \chi_{1a}| + |\psi_{0a} - \psi_{1a}|$ ($|\chi_{0b} - \chi_{1b}| + |\psi_{0b} - \psi_{1b}|$).

Table 4.1 shows results comparing the MIP model, CP Benders, and TSP Benders. In both Benders decomposition models, cut (C2) was used. For these results, the time until an optimal solution was found and proved were recorded. Where the solving timed out, 3 hours was used.

The Benders decomposition model's results, both CP and TSP versions, are a significant improvement over the MIP performance. It is clear that the Benders decomposition approach is capable of solving much larger problems in significantly shorter run times. The Benders decomposition approach, with a CP solver, solves up to 20 jobs in the time limit and the majority of instances of up to 30 jobs. Replacing the CP solver with a TSP solver, the Benders model is able to increase the number of solvable jobs up to 30 consistently. In contrast, the MIP model is able to solve only 10 jobs in the 3 hour time limit.

We see that increasing the number of machines has a greater effect on the performance of the models than increasing the number of jobs. In both Benders models, the master problem had difficulty solving for increased machines. In fact, the TSP subproblem is able to solve each subproblem in milliseconds while the MIP master problem can spend hours searching for an assignment. The opposite is seen for the MIP model. The MIP model has difficulties sequencing large number of jobs on machines and so, in the case where there are only 2 machines, the MIP model has a very high runtime for instances of 10 jobs and 2 machines. When the number of machines is increased to 5, the sequencing problem is simpler and results in fast runtimes.

Using the Benders decomposition model with the TSP subproblem, the three different cuts presented in Section 4.2.3 are tested on the same set of instances for problem sizes of 10, 20, and 30 jobs. The results are presented in Table 4.2.

Table 4.2 shows that the performance of cut (C2) is the best overall. In most cases, the difference is not significant, but there are cases where clear differences are found. Specifically,

		MIP		**CP Benders**		**TSP Benders**	
N	M	Avg Runtime	# uns.	Avg Runtime	# uns.	Avg Runtime	# uns.
10	2	9885.14	9	0.24	0	**0.12**	0
	3	7924.16	7	0.29	0	**0.22**	0
	4	1169.69	1	0.55	0	**0.42**	0
	5	13.59	0	0.56	0	**0.45**	0
20	2	10800.00	10	9.79	0	**1.08**	0
	3	10800.00	10	6.41	0	**2.02**	0
	4	10800.00	10	239.15	0	**53.12**	0
	5	10800.00	10	509.85	0	**122.46**	0
30	2	10800.00	10	7997.13	5	**2.15**	0
	3	10800.00	10	3244.75	1	**27.85**	0
	4	10800.00	10	10041.10	2	**203.13**	0
	5	10800.00	10	8804.01	5	**750.89**	0
40	2	10800.00	10	10800.00	10	**4.78**	0
	3	10800.00	10	10800.00	10	**651.30**	0
	4	10800.00	10	10800.00	10	**1538.69**	0
	5	10800.00	10	10800.00	10	**7404.07**	5

Table 4.1: Comparison of MIP, CP Benders, and TSP Benders: Average CPU runtime in seconds and the number of unsolved instances.

		(C1)		**(C2)**		**(C3)**	
N	M	Avg Runtime	Avg # Iter	Avg Runtime	Avg # Iter	Avg Runtime	Avg # Iter
10	2	0.24	2.1	**0.12**	2.1	**0.12**	2.1
	3	0.25	1.9	**0.22**	1.9	0.26	1.9
	4	0.42	1.8	0.42	1.8	**0.40**	1.7
	5	0.56	1.5	**0.45**	1.4	0.47	1.4
20	2	1.15	4.5	**1.08**	4.5	1.09	4.5
	3	2.06	2.7	**2.02**	2.7	2.05	2.7
	4	56.87	3.9	**53.12**	3.9	70.06	3.9
	5	153.33	4.4	**122.46**	4.0	124.30	3.8
30	2	2.17	4.4	**2.15**	4.4	2.28	4.4
	3	29.31	5.4	**27.85**	5.2	30.52	5.2
	4	224.67	4.9	**203.13**	4.5	222.83	4.7
	5	784.39	5.6	**750.89**	5.3	817.22	5.3

Table 4.2: Comparison of different cuts on the TSP-Benders: Average CPU runtime in seconds and the average number of iterations until optimality is found and proven.

(C2) is able to, in a small number of instances, create a cut that removes an assignment that (C1) would not, which reduces the total number of iterations required and the time needed to solve the problem. In the 20 jobs and 5 machines test case, the average number of iterations for (C1) is 4.4 while cut (C2) was able to decrease the iterations to 4, resulting in a 30 second runtime difference on average to reduce the runtime by about 25%.

One would then assume that cut (C3), using more information, would create better cuts. However, the results showed that cut (C3) performs worst than cut (C2). Cut (C3) created a more difficult master problem to solve which increased the total solve time per iteration, while not becoming much tighter than (C2) to reduce the overall number of iterations. The test case with 20 jobs and 5 machines shows how (C3) only reduces the number of iterations by a very small amount, a further 0.2 iterations from cut (C2), but increases on average by 2 seconds of runtime. Including into the cut all jobs that were not assigned to a machine accounts for more information, but proves detrimental to the performance of the Benders model.

In the 30 jobs and 4 machines case, we even see cut (C3) increasing the number of iterations over cut (C2). This seems to contradict the fact that (C3) is a tighter cut. The increase in iteration occurs because some degeneracy exists in the problem. Varying the cut may lead to different assignments in the Benders master problem with equal objective functions. In rare instances, it is possible that the other cuts will lead to the optimal assignment while (C3) leads to an equivalent master problem that does not extend to a global solution.

4.4 Relaxed Benders Decomposition

Though the Benders decomposition approach obtains significant speed-ups, the problem sizes that can be solved are limited compared to what heuristic models are currently solving. For larger problems, Benders decomposition is not able to complete a single iteration of the master problem. Problem instances of 120 jobs are tested in previous papers using heuristics and local search [49, 87]. Specifically, in work done by Helal et al. [49], schedules for problem

instances of 100 jobs and 10 machines are found within minutes. The quality of these schedules is difficult to assess given that the optimal solutions are not known. However, for smaller instances (8 jobs and 4 machines) the optimal solution was known from the MIP model and the heuristic used was experimentally shown to have up to 5% deviation from optimal.

If optimal solutions are not possible due to large instances, it is clear that heuristic solutions are necessary. Stopping the Benders program early and using the best found schedule as shown in Section 4.2.2.3 is one approach to increase the size of problems for which the Benders model can obtain schedules. However, the approach is only useful if the problems are small enough such that the Benders model can solve one complete iteration in a reasonable time. From our experiments, we found some instances of 40 jobs and 5 machines where solving for one complete iteration took more than one hour. Solve times of the subproblem are almost instantaneous when the TSP solver is used, but the MIP master problem may be intractable once the size of the problem reaches 50 or more jobs and 5 machines. For problems as large as 100 jobs, the Benders decomposition model is not likely to solve the master problem within the time limit.

Therefore, we propose not solving the master problem to optimality. During the search for a solution to the MIP, we will allow the solver to stop once a solution is found within some predetermined gap from the best lower bound obtained. Doing so reduces the effort required in the master problem at producing assignments and enables the model to generate feasible schedules faster. The change to the Benders decomposition master problem results in it no longer acting as a true lower bound. However, it is guaranteed that the solution found is within the chosen optimality gap. Experimental results from large instances show that the master problem often spends most of the time proving optimality or obtaining optimality with a gap of less than 5%. The Benders approach would be able to obtain solutions much faster if the master problem was allowed to solve until it has proven it is within 5% of optimal.

Table 4.3 presents results of using the relaxed Benders decomposition on the same set of instances tested in Section 4.3. The optimality gap was varied to examine the speed-up

		Benders 0%		**Benders 2%**		**Benders 5%**		**Benders 10%**	
N	M	Runtime	MRE	Runtime	MRE	Runtime	MRE	Runtime	MRE
10	2	0.12	0	0.12	0.07	0.09	0.38	0.08	1.87
	3	0.22	0	0.23	0	0.23	0.33	0.14	0.96
	4	0,42	0	0.40	0	0.39	0	0.31	0.23
	5	0.45	0	0.43	0.13	0.42	0.03	0.37	0.38
20	2	1.08	0	0.80	0	0.29	0.61	0.27	1.49
	3	2.02	0	1.57	0	1.01	0.27	0.38	2.56
	4	53.12	0	45.78	0	23.41	0.11	4.43	1.46
	5	122.25	0	112.17	0	67.33	0	11.98	0.53
30	2	2.15	0	0.69	0.25	0.39	1.11	0.39	1.79
	3	27.85	0	12.34	0.06	1.33	0.95	0.46	4.58
	4	293.13	0	137.25	0	10.08	0.21	1.36	2.79
	5	750.89	0	321.78	0	181.14	0.06	4.98	1.23
40	2	4.78	0	1.17	0.06	0.85	1.97	0.83	2.48
	3	651.30	0	154.68	0.04	1.88	0.96	0.64	3.76
	4	1538.69	0	1193.00	0.03	21.6	0.73	2.09	4.01
	5	7404.07	0.53	4543.55	0.23	317.93	0.39	3.92	2.82

Table 4.3: Relaxed Benders Performance: Average CPU runtime in seconds and MRE.

obtained when relaxing the problem at different intensities. However, by increasing the gap, the quality of the solution degrades. For the Benders results in Table 4.3, mean relative error (MRE) was calculated using the optimal solution from the Benders problem with a 0% gap. Where optimality was not found or proven, the MRE was calculated based on the tightest lower bound available. This calculation gives the upper bound on the error from the solution to the optimal makespan.

The experimental results indicate that increasing the tolerance of the master problem leads to a significant decrease in runtime. By having a gap of 2%, the runtime of the Benders decomposition reduces to less than half of the exact model in many instances. Further increasing the gap to 5% and 10% leads to reducing the runtime by 3 to 4 orders of magnitude in some instances. The instances that took hours to solve previously had solutions within minutes.

The quality of the solutions found when using an optimality gap are still close to optimal and competitive. Though the gap represents an upper bound on the distance from an optimal solution, the experimental results show that the actual error is on average much less when compared to the optimal solutions found by the complete Benders model. A 2% gap often yielded optimal solutions at lower loads and less than 0.3% realized gaps. Even at 5% gaps, the results were within less than 1% of optimality.

In an attempt to compare the updated Benders decomposition with current state of the art heuristics, similar problems need to be generated. *Scheduling Research* [86] is a resource where data can be found on the performance of the tabu search, Meta-RaPS, and ACO for various problem instances. However, the instances that the heuristics used to gather these results are not available online so we have created instances of the same size to make comparisons. In our previous experiments, processing time and setup times were varied between 1 and 100. The data collected on the state of the art heuristics generate instances with processing time and setup times ranging only from 50 to 100. Therefore, we create new instances with these properties for problems of size 60 to 120 jobs and 2 to 10 machines.

We directly compare the solution found from the updated Benders to a lower bound (LB)

calculated for each problem instance. The lower bound used is the same as that used in the analysis of all heuristics in the work on ACO and is calculated using the equations

$$LB1 = \frac{1}{|M|}\sum_{j=1}^{N} \min_{i=1,\ldots,|M|;k=1,\ldots,|N|}[p_{jk} + s_{ijk}]$$

$$LB2 = \max_{j=1,..,|N|}\left\{\min_{i=1,\ldots,|M|;k=1,\ldots,|N|}[p_{jk} + s_{ijk}]\right\}$$

$$LB = \max(LB1, LB2)$$

We compare the Benders approach with a 5% gap since it resulted in the best compromise between quality and speed. The deviation, Δ, of the schedule found from the lower bound is

$$\Delta = \frac{C_{max}(Benders) - LB}{LB}$$

Table 4.4 demonstrates that the Benders approach is competitive with the state of the art heuristics. On all tests, the Benders model with a 5% gap outperforms Meta-RaPS and tabu search. However, ACO is able to obtain better solutions than Benders for some instances. Problems with fewer machines favour ACO while the problems with more machines favour the Benders method. Not only is the incomplete Benders approach producing good schedules, it is able to do so in a notably shorter time. However, not all instances had runtime information for comparison so the data presented is the only values available for similarly sized instances.

Note that the Δ calculations are an overestimation of how far the schedules really are from the optimal solution. In some of the instances of Table 4.4, the Benders approach with a 5% gap realizes Δ greater than 5%. This is because the lower bound calculations, LB, is not as tight as the MIP lower bound.

The results reveal that the Incomplete Benders can produce schedules just as good or even better than state of the art heuristics in less time. One could further see how parameter tuning of the gap sizes would lead to even better performance to balance the quality and speed required for a specific problem. The quality of ACO schedules for problems with 2 machines outperformed the Benders with a 5% gap, but Table 4.3 exhibits how the Benders is able to quickly

		Meta-RaPS		**Tabu Search**		**ACO**		**Benders 5%**	
N	M	Δ	Runtime	Δ	Runtime	Δ	Runtime	Δ	Runtime
60	2	3.07	-	6.45	-	**1.57**	255.00	2.60	2.01
	4	6.16	-	8.17	-	4.54	192.60	**4.20**	4.84
	6	7.52	-	10.07	87.71	7.12	-	**3.86**	88.67
80	2	2.97	-	5.95	-	**1.25**	416.40	2.26	5.50
	4	6.02	-	7.66	223.81	3.97	311.40	**3.60**	22.83
	6	7.29	-	9.76	-	7.09	-	**5.46**	36.02
	8	7.97	-	10.59	-	7.41	-	**3.47**	57.91
100	2	2.91	-	6.21	133.22	**1.08**	626.40	2.15	18.83
	4	4.62	-	7.06	-	**3.54**	557.40	3.69	52.93
	6	6.15	-	8.84	222.05	5.58	544.20	**4.92**	99.35
	8	9.62	-	10.18	-	8.11	-	**5.15**	329.34
	10	8.89	-	10.73	379.36	8.15	-	**3.07**	539.34
120	2	2.69	-	6.27	-	**0.92**	-	2.61	27.92
	4	4.31	-	6.80	-	**3.00**	-	3.04	159.21
	6	5.38	-	8.20	-	**4.52**	-	4.72	222.91
	8	8.01	-	10.09	-	5.70	825.00	**4.53**	482.68
	10	10.58	-	10.19	-	6.99	-	**3.65**	897.98

Table 4.4: Incomplete Benders Performance: Average CPU runtime in seconds and Δ calculations. The Incomplete Benders approach is not tested on the same instances as the other heuristics since the exact instances were not available. Similarly sized instances were generated for this comparison.

find schedules for 2 machine problems. To that end, if a smaller gap was used for problems with fewer machines, we would expect to see a shorter makespans without sacrificing too much speed.

4.5 Conclusion

In this chapter, we presented a logic-based Benders decomposition approach to minimize the makespan of an unrelated parallel machine scheduling problem with sequence and machine dependent setup times. A MIP model was defined to solve for the assignment of jobs to machines and produce a lower bound on the achievable makespan of the problem. Two subproblems based on a CP and TSP solver were implemented to find optimal schedules for the assignment of jobs on each individual machine. The computational results indicate that the cooperation of MIP and TSP can effectively find optimal solutions. We are able to solve instances four times larger than what was previously possible using a MIP formulation found in the literature and obtain optimal solutions on problems of the same size up to five orders of magnitude faster. Additionally, a relaxation of the Benders decomposition is discussed and results of the changes shows that good solutions can be found in much shorter time and for larger instances if optimality is sacrificed. The incomplete Benders is then compared to the state of the art and shown to be best on many instances with much lower time and a guarantee of being within 5% of optimality.

Chapter 5

Scheduling with Flexible Servers & Setup Times

In Chapter 3, we examined a queueing network with flexible servers where no setup times are incurred when servers switch between processing jobs of different classes. We extend the problem back to the original problem introduced in Section 2.4 to include non-zero setup times in this chapter. Queueing theory and scheduling approaches are studied for the problem and we show that incorporating properties of the queueing model into the scheduling method creates a scheduler with better mean flow time performance than either approach can achieve alone. Further, we prove that the hybrid model is able to guarantee stability for the maximal capacity of the system.

5.1 Back to the Dynamic Problem

The problem studied by Andradóttir et al. , which we introduced in Section 2.4, was concerned with a dynamic system where jobs belong to one of K classes and are processed on one of M servers. Any jobs arriving to the system and unable to be serviced immediately enter a queue of infinite buffer length.

The servers in the system can be assigned to different classes. At any time, a server is able to leave a class to service another. However, switching times from a class may be non-zero. If a server j leaves class i to start serving class k, that server must be idle for a time of $\zeta_{ik}^{j}(n)$ on the nth occurrence of such a switch.

When assigned to a specific class k, a server j will work at a rate of μ_{jk}. This rate is dependent on the server and the class being serviced. If server j cannot be assigned to a class k, then $\mu_{jk} = 0$. We define the realized processing time of a job l on server j to be p_{jl} and has an expected value $\frac{1}{\mu_{jk}}$, where job l belongs to class k.

We studied Andradóttir et al.'s queueing network in Chapter 3, but removed setup times to simplify the problem. Our results showed that a hybrid queueing and scheduling approach was able to experimentally improve performance. Chapter 4 was a study to help us include setup times. We simplified the problem in a different way by making it static and created a logic-based Benders decomposition approach to minimize the makespan of a schedule. In this chapter, we use the Benders decomposition model to schedule the dynamic queueing network with the inclusion of setup times.

5.2 MinMksp Model

The introduction of setup times changes the problem such that the *MinMksp* model presented in Chapter 3 does not accurately represent the network. We update the *MinMksp* model and compare its performance to that of the round robin policy.

5.2.1 Minimizing the Makespan of a Period

We alter the *MinMksp* model from Chapter 3 to account for setup times. Similar to the previous method, the *MinMksp* model used for a queueing network with setup times is a periodic scheduler which schedules all jobs that are present in the network with the goal of minimizing the makespan. Once a server is free again, a new schedule is created with all the jobs that

arrived during the previous period. Instead of using the MIP model to minimize the makespan, we implement the Benders decomposition method developed in Chapter 4.

5.2.2 Performance Comparison

We test the *MinMksp* model and round robin policy on two systems: a system with two servers and four queue classes and another with four servers and four queue classes. These two cases respectively represent a system in which the servers must switch between different classes more often since there are fewer servers than queue classes and a system in which less switching is required since the number of servers and queue classes are equal. Further, each system is tested under two setup time configurations where the setup times are either long or short. The parameters for these systems can be found in Appendix C.

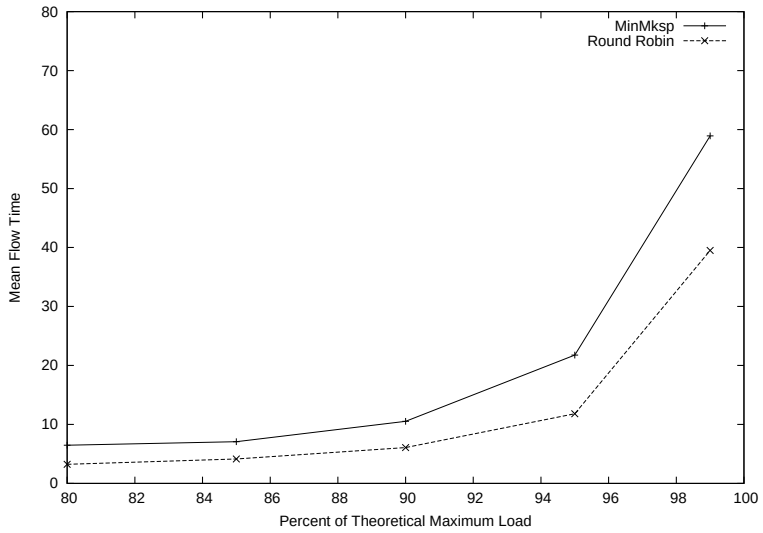

Figure 5.1: Comparison for a system with 2 servers and 4 class types with short setup times.

All experiments were tested on a Dual Core AMD 270 CPU with 1 MB cache, 4GB of main memory, running Red Hat Enterprise Linux 4. For each of the test cases, 5 varied loads between 0.8 to 0.99 of the maximum theoretical load the system can handle were simulated.

At each load, 20 instances were tested for 10,000 time units to create a total of 400 simulations per model.

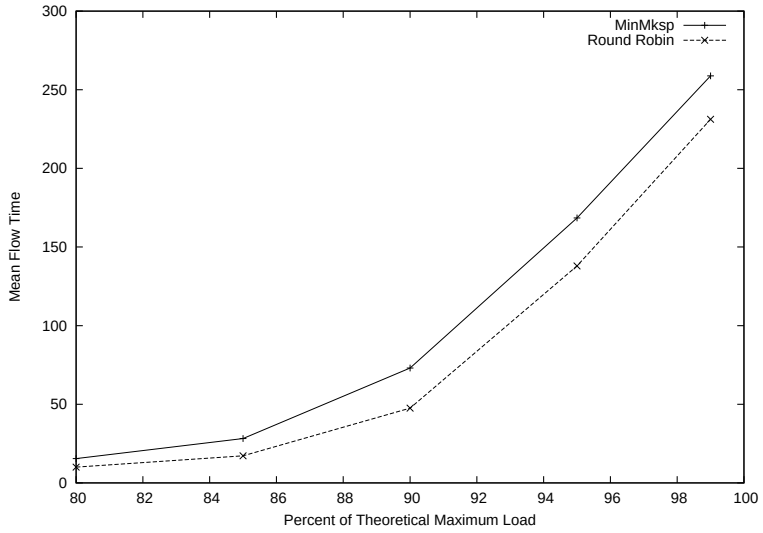

Figure 5.2: Comparison for a system with 4 servers and 4 class types with short setup times.

Figures 5.1 and 5.2 compare results of the two models when the system has setup times with lengths on the same order of magnitude as the service times. Figures 5.3 and 5.4 are results of the same systems respectively, but with setup times that are an order of magnitude larger than service times. These graphs demonstrate that the round robin policy dominates the *MinMksp* model in flow time performance for all tested loads when there are setup times.

5.2.3 Advantages of the Round Robin Policy

We identify two factors which contribute to the favourable performance of the round robin policy. The first factor is that the *MinMksp* model is making unnecessary and bad assignments when considering only the short term goals. In contrast, the round robin policy does not. Second, the scope of the queueing model is longer than the scheduling policy leading to larger cycles.

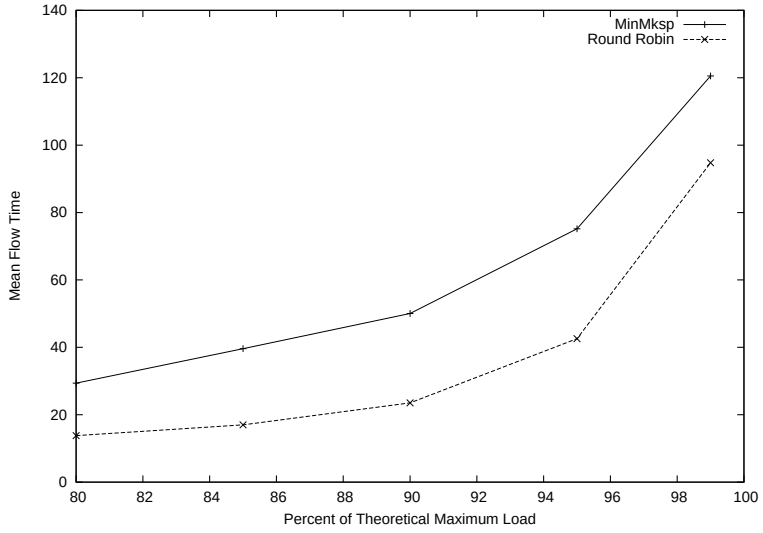

Figure 5.3: Comparison for a system with 2 servers and 4 class types with long setup times.

By attempting to minimize a makespan of a period, the *MinMksp* model will greedily make decisions without accounting for long run performance. Such an approach leads the scheduler to often assign most of a server's time to a class which it can efficiently serve, but to also assign a small number of jobs from another inefficient class when there is sufficient available time in a period. For example, assume a schedule with a makespan that is 40 time units in length. If a server in this schedule is assigned efficient jobs which take up 35 time units and 1 extra job from another class which is 1 time unit in length and incurs 4 time units of setup, the optimal schedule to minimize makespan may be to include the job. However, in the following period the server is likely to proceed with the more efficient class again, further incurring a setup time before service can resume at the original job class. If the setup time to switch back to the more efficient class is another 4 time units, all jobs in that class are delayed by that extra 4 time units before service can occur. In the example, a total of 9 time units is devoted to serve a single job of duration 1. If this job is served elsewhere instead, say to a server which is already serving the class and would result in a total makespan of 41 time units, the scheduler could be in a

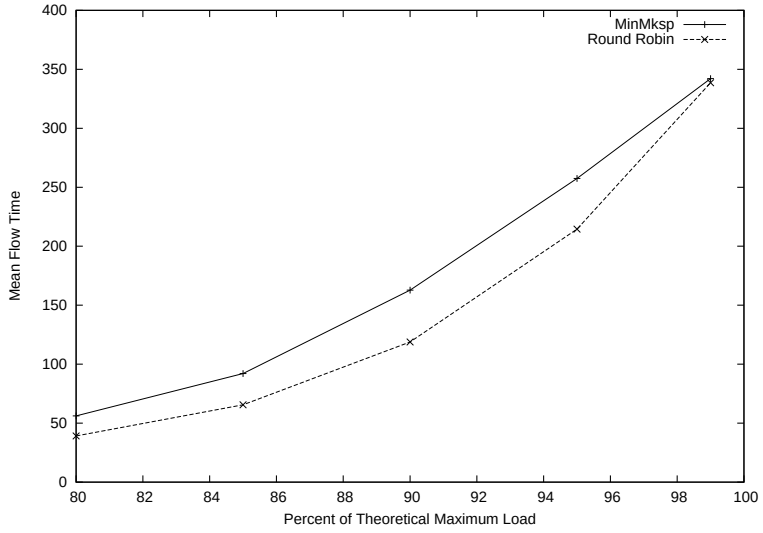

Figure 5.4: Comparison for a system with 4 servers and 4 class types with long setup times.

better situation beginning in the next period. Therefore, there are "hand off" issues from one period to the next such that a scheduler should aim not to make decisions which impair future periods.

The nature of scheduling in periods is that a small number of jobs from inefficient classes are often scheduled to some of the servers. This leads to spending a large portion of time in setting up to and from a class in order to serve only 1 or 2 jobs. Additionally, the scheduler may only consider one setup time when creating a schedule since the potential setup after a job is not modeled if that class is last in a period. It is apparent that there are situations where delaying the processing of these small number of jobs for another server is better than trying to fit it into the schedule immediately.

The second factor which gives the round robin policy an advantage is that servers will spend more time on a class before switching to another. For each server j, specify a list V_j using all classes k with $\delta_{jk} > 0$. A complete cycle for a server is to service all classes in V_j. The round robin policy uses the l_{jk} values to decide on the upper limit of jobs from class k server j will

serve before switching. If there are less than l_{jk} jobs available, the server moves onto the next class when the current queue is empty. During a cycle, the server will have served, at most, $\sum_k l_{jk}$ jobs.

The *MinMksp* model considers only the present jobs in the system during the beginning of a period. The size of a period is experimentally found to be one-tenth of the maximum cycle size for the round robin policy and up to one-fifth of the realized cycle sizes. The round robin policy has a scope that is much greater than that of the *MinMksp* model and will serve more jobs before switching classes since the policy will consider jobs present when the server starts processing a class as well as the jobs that arrive during the time a server is busy in that class. Service of a larger number of jobs between setup times shows that the policy is not limited to dealing with an outdated system and allows for a cycle to include jobs not present when the cycle starts. In comparison, any time the *MinMksp* model switches to a new class when jobs of the previous class still require processing, we can likely expect immediate performance loss since service could have occurred prior to any setup times.

Both advantages presented demonstrate how the inclusion of setup times changes the behaviour of the system substantially enough that the round robin policy is able to obtain better performance across all simulated loads. The round robin policy is more efficiently situating and reducing setup times. A small change to where and when a setup time occurs can lead to dramatic differences in flow time performance even if the resulting total setup time is equivalent.

5.3 Hybrid Model

The previous section illustrated the dominance of the round robin policy over the *MinMksp* model. In this section, we create a hybrid model that takes advantage of the dominating properties of the round robin policy in the framework of the scheduling model.

5.3.1 Guiding the Scheduling Model

We make use of the advantages of the round robin policy to improve the *MinMksp* model. We modify the *MinMksp* model to use the round robin policy to guide the scheduling decisions.

The first problem is to reduce the number of inefficient assignments made by the scheduler. We know an efficient assignment to be one that resembles the δ_{jk} values from the allocation LP. In Chapter 3, guidance from the δ_{jk} values through a penalty cost on deviation was able to significantly improve the performance of the periodic scheduler. Guidance as before is not sufficient when setup times are non-zero. By allowing classes access to servers which are inefficient, we continue to witness the problem of single jobs being placed on a server to greedily reduce makespan. Testing the model under such a penalty led to little or no improvement on the *MinMksp* model.

A more extreme approach is taken. We propose that restricting the scheduler to only allow assignments when $\delta_{jk} > 0$ will reduce the number of inefficient assignments and unnecessary setups. Therefore, we set $x_{jl} = 0$ whenever $\delta_{jk} = 0$ and $l \in k$.

Such a restriction will stop the scheduler from greedily inserting a small number of jobs from class k onto a server j when $\delta_{jk} = 0$. Such assignments can be considered inefficient since, in the long run, the server should be allocating δ_{jk} fraction of time to a class. The restriction also reduces the number of classes that each server will serve thereby decreasing the possible setup times.

One may wonder whether the restriction to positive δ_{jk} is in fact better than the hybrid made in Chapter 3 because it does not take into account the magnitude of the δ_{jk} values. At lower loads where slack exists, the proposed restriction is able to remove extra assignments and therefore reduce premature setup times as we wanted. With slack from lower arrival rates, closely following the δ_{jk} values is less important. We expect greater performance loss from superfluous setups because deviation from δ_{jk} can still lead to stable systems. When increasing the load of the system, minimizing makespan will naturally allocate resources to be similar to the δ_{jk} values because such an assignment maximizes throughput.

The second issue presented earlier is that the round robin policy is considering a larger horizon. Unlike the round robin policy, the scheduling approach is not able to include jobs that have yet to arrive. In order to increase the planning horizon and account for jobs that arrive during a period, we need to change the structure of the periodic scheduler. We wish to allow the scheduler to delay committing to a setup time based on a schedule made at the beginning of a period. Servers should be able to continue processing jobs from a class that it is already processing if new jobs arrive rather than switching classes and delaying the new jobs. Without the ability to reevaluate the system, setup times could be incurred earlier than needed which leads to worst performance. We see such a case if a server is currently serving a job of class 1 and is committed to another job of class 2. If during the service of class 1, a new job belonging to class 1 arrives, a server committed to following through with a set schedule would serve class 1, switch to class 2, then switch back to class 1 to serve the new job. If we reevaluate the system, we would see that serving the new job directly after serving the first job would only incur one setup time.

The proposed change to the periodic scheduler is to commit each server to only serving one class in a period. The model would reschedule once a server has completed serving jobs of one class even if there are still jobs from a different class assigned to that server. By rescheduling, servers do not commit to a setup time until it reevaluates the jobs in the system. During the time which a server is busy serving a class, new jobs may arrive and the server has a chance to process these jobs after a rescheduling occurs and before it switches to a different class.

Because the proposed model only serves one class of jobs during a period, we update how the model chooses which class to serve as well. When a schedule is created in which a server has been assigned jobs of the same class that it served at the end of the previous period, it will continue serving this class to delay any unnecessary setup times. The nature of minimizing the makespan when setup times do not adhere to the triangle inequality means that switching classes immediately may lead to a better makespan even though we could continue serving jobs without incurring a setup. However, the purpose of minimizing makespan is to increase

throughput with the goal of maintaining stability. We are more interested in the mean flow time performance than in minimizing makespan. Since the schedule is expected to change after servicing on a single class, sacrificing optimality of the static schedule to help reduce mean flow time immediately will lead to better performance.

If no jobs of the previous class are assigned to a server, it will serve the class which has been assigned to it with the greatest number of jobs. If we were to schedule the largest number of jobs first, all jobs in the class are delayed only by a single setup time. If ordering is reversed and another class is serviced first, the larger number of jobs are further delayed by two setup times. A delayed setup time indicates that fewer jobs are affected by the setup. For example, if two classes are scheduled on a server and one class has 10 jobs and the other has 5, scheduling the larger class first means 10 jobs are affected by one setup while 5 are affected by two. In the reverse order, 5 jobs are affected by one setup and 10 jobs are affected by two. If both setup times are 1 time unit in length, the first scenario increases the total flow time by 20 time units whereas the second scenario increases the total flow time by 25.

We further impose that a server j will serve at most l_{jk} jobs of class k consecutively. l_{jk}, is chosen using the same values as those used in the round robin policy. We enforce the limit by switching the server regardless of whether there are still more jobs assigned by the scheduler. By limiting the maximum number of jobs a server processes of one class, we ensure that a server will not over commit to any class. It is likely that at high loads there is a greater probability that upon completion of all jobs in a class by a server for the previous period, a new job of that same class would arrive. The server may then be assigned to continue serving this class though other classes have been awaiting service for a long time. The limit imposed allows the system to distribute service so that classes are not waiting for too long.

The changes proposed drastically alter the *MinMksp* model. The makespan is still an objective to be minimized, but the restricting constraint of assigning only when δ_{jk} is positive will result in different schedules with greater than or equivalent makespans. Further, a server only commits to a single class of jobs before reevaluating the system and potentially changing the

schedule.

The hybrid model is an attempt to integrate the *MinMksp* model and round robin policy. We see the new model behaves in a similar fashion to the round robin policy, but maintains the framework of the *MinMksp* model. The modifications made to the scheduling model enforce the scheduler to only serve what the long run steady state analysis defines as efficient classes, while also having the ability to react to changes in the system before major decisions are being made (i.e., switching to a different class).

The contrast between our hybrid model and the round robin policy is the ability to reason about the state of the system. Minimization of makespan allows load balancing to still be effective since any realization could differ from the expected. In the case where more jobs than expected of one class are currently present in the system, our model will be able to recognize it and react accordingly. Also, our model enables the scheduler to return to a previous class before completing a full cycle. In a fully loaded system, completion of cycles is important. However, when loads are low and a realization of the system has some classes which have more arrivals than expected and others with fewer, delaying service of the smaller classes will help to reduce the queues from the larger classes to get better performance. As the classes with fewer jobs get neglected, their queue size will increase and the scheduler will give the class greater priority. By creating schedules in this way, the cost of setup times are dispersed between a greater number of jobs to improve flow time.

5.3.2 Experimental Results

We test the hybrid model on the same instances as in Section 5.2. We further implement the *MinMksp* model with each change independently. When the model is constrained to only assign jobs where $\delta_{jk} > 0$, we label it as the *Restricted* model. We call the model *Reschedule* when we only commit to serving one job class before rescheduling. The hybrid model incorporates both these additions.

Figures 5.5 to 5.8 present the performance of the new models under the same instances as

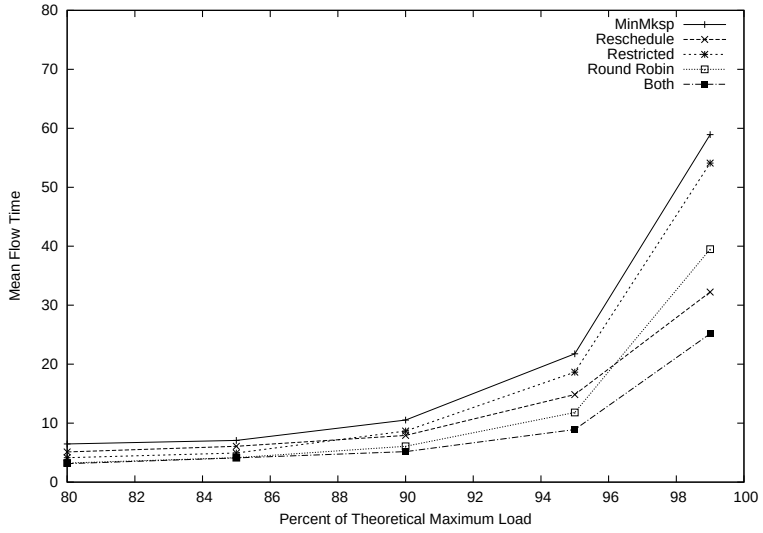

Figure 5.5: Comparison for a system with 2 servers and 4 class types with short setup times.

those previously shown. These results demonstrate that the changes to the *MinMksp* model are able to improve performance. However, the changes individually are still not competitive with the round robin policy. The hybrid model is able to outperform the other models on all tested loads when both changes are added together.

We see that the *Restricted* model is able to produce good performance at lower loads, while the *Rescheduler* model is better at higher loads. We can expect this since at lower loads, the schedule is expected to assign more inefficient jobs. As the load increases, the scheduler naturally mimics the allocation LP because there is less slack to allow otherwise and the *Restricted* model has less impact. The *Reschedule* model in contrast makes more changes to the *MinMksp* model at higher loads when there is a higher probability that jobs will arrive while a server is busy during a period.

The improvement in performance of the hybrid model over the round robin policy comes from the ability to account for the current system state. The round robin policy will follow a predefined cycle regardless of the state of the system. The only time the policy reacts to the

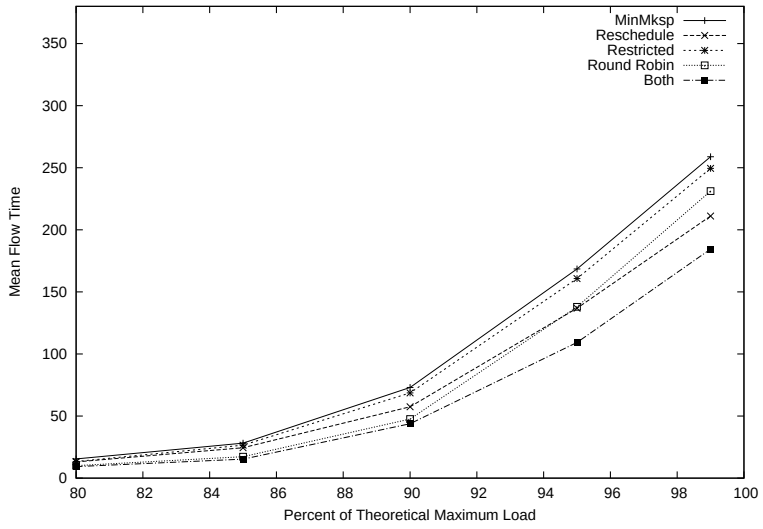

Figure 5.6: Comparison for a system with 4 servers and 4 class types with short setup times.

system is when a queue for a class that it is serving empties. In this situation, the server switches to the next class and continues service. Our hybrid model is able to realize that some classes have a greater number of jobs than others and can be more effectively served immediately. If two classes are awaiting service, one with 10 jobs and one with 1 job, the queueing policy does not discern the two. Service is offered to the class next in the list. However, our hybrid will place greater emphasis on the larger class and service is more likely to begin there.

Another advantage of the hybrid model is the use of processing time information. The inclusion of processing time knowledge allows the scheduler to balance the loads more evenly and better choose which specific jobs should be allocated to servers. In the long run, the round robin policy will realize expectation, but in the short run, the scheduling model can mitigate the effects of outliers in processing time by scheduling accordingly.

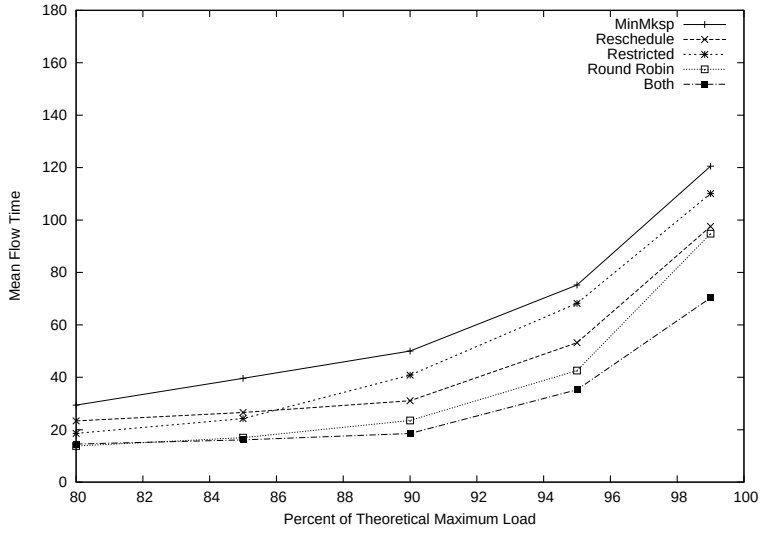

Figure 5.7: Comparison for a system with 2 servers and 4 class types with long setup times.

5.4 Stability of the Hybrid Model

Andradóttir et al.'s round robin policy was proven to maximize the capacity of the system and achieve stability for an arrival rate arbitrarily close to the maximum found in the allocation LP. An important characteristic of a scheduler is to provide good flow time performance to satisfy jobs while also maximizing the stability of the system. Evidently, an important question is whether the hybrid model is able to guarantee stability at the same loads as the round robin policy. The following theorem provides the stability guarantees of the hybrid model.

Theorem 5.4.1 The hybrid model is guaranteed to achieve the desired capacity $\lambda < \lambda^*$.

Proof: To illustrate the stability of our hybrid model, we compare it to the round robin policy. We are interested in the scenario where the round robin policy is stable, but hybrid model is not. If this situation can not happen, then we show that the hybrid model maximizes stability for any capacity $\lambda < \lambda^*$.

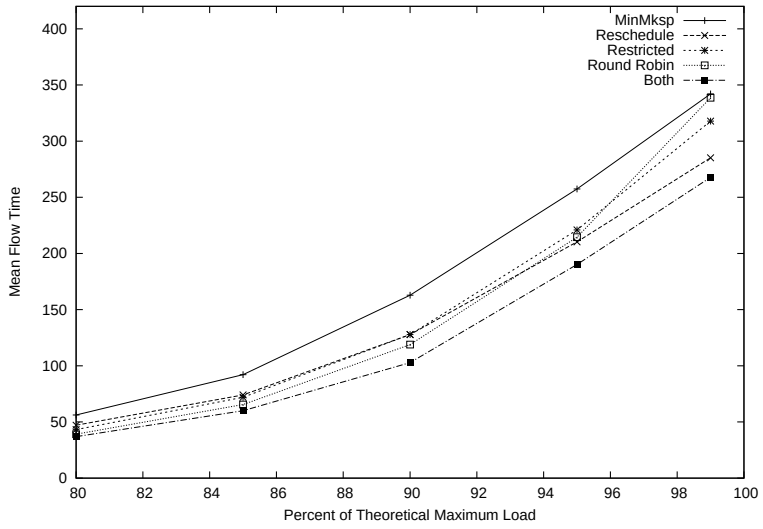

Figure 5.8: Comparison for a system with 4 servers and 4 class types with long setup times.

We assume that the round robin policy is stable and the hybrid model is not. By definition, if the round robin policy is stable, the long run steady state expected number of jobs in the system converges to a finite value $E[N_{RR}]$. Since the hybrid model is unstable, the number of jobs, N_H, increases to infinity with time. Yet, if we consider a situation where N_H has increased to a sufficiently large size, we know that the hybrid model will schedule so that servers process a maximum number of jobs equal to l_{jk}. In this situation, a server will be serving only the classes with non-zero δ_{jk} values and the servers are restricted to serving at most l_{jk} jobs of class k. Here, the behaviour of the hybrid model coincides precisely with that of the round robin policy. The round robin policy guarantees stability by cyclically processing up to l_{jk} jobs for all classes with non-zero δ_{jk}.

Finally, we know that the round robin policy is stable regardless of the state of the system. The hybrid model will satisfy conditions for stability once the system increases past a threshold such that l_{jk} is the number of jobs served between switches. The

scheduler will change to a stable policy as soon as the queue sizes are sufficiently large even if the hybrid model is not able to efficiently reduce the jobs in the system when there are fewer jobs and a server cannot process l_{jk} jobs before switching. At this point, the scheduler will maintain stability since the state of the system is unimportant, allowing us to ignore the performance of the hybrid model when N_h is small. As a result, in any system in which the round robin policy is stable, the hybrid model will also be able to achieve stability. From Theorem 2 in Andradóttir et al.'s [4] work, we know that the round robin policy can achieve any desired capacity $\lambda < \lambda^*$. Therefore, the hybrid model will also be able to achieve these capacities.

■

5.5 Conclusion

In this chapter, we study a queueing network with flexible servers and setup times. Flexibility here means servers can process jobs from many classes at different rates. A scheduling and queueing model are compared experimentally. We found that the scheduling model has much worse mean flow time performance. Through analyzing the advantages of the queueing policy, a hybrid model is created which incorporates properties of the queueing approach into the scheduling framework. The hybrid model is found to have the lowest mean flow time at all loads while also maximizing the arrival rate that can be stabilized.

Chapter 6

Conclusions and Future Work

6.1 Summary and Contributions

6.1.1 Investigation of a Parallel Machine Scheduling Problem

In Chapter 4, we studied a system with unrelated parallel machines and sequence dependent setup times. The machines are capable of servicing any job, but the processing times vary depending on the machine that services the job. There is also a setup time that must elapse between servicing any pair of jobs which is machine and sequence dependent. The goal of the problem is to find a schedule that minimizes the latest completion time of all jobs.

6.1.1.1 A Complete Approach

This dissertation presents a complete method for the parallel machine scheduling problem using logic-based Benders decomposition. By partitioning the problem into an assignment master problem modeled as a mixed integer program (MIP) and sequencing subproblem modeled as a traveling salesman problem (TSP), optimal solutions were obtainable for problems four times larger than with a MIP formulation which is, to the knowledge of the author, the only other complete approach that guarantees minimum makespan in the literature. Furthermore, where an optimal solution was found by both our method and the MIP model, our approach was able

to find the solution up to five orders of magnitude faster.

6.1.1.2 A Heuristic Approach

Given the difficulty of solving the parallel machine problem in consideration, research in this area has been focused on meta-heuristic methods. The current best state-of-the-art algorithms are meta-heuristics techniques which can typically find "good" solutions on problems up to 120 jobs and 10 machines. The complete approach we used can only solve problems half in size. We relax the logic-based Benders decomposition model to accept schedules which are less than optimal for the makespan. Instead of solving the master problem until an optimal solution is found, a percentage gap from optimality is possible. If a solution is found to be within the chosen percent of a lower bound, the solution is accepted. The relaxation enables the model to compete with the performance and speed of other state of the art algorithms on larger problems. We found as the number of machines were increased to 10, the Benders decomposition was able to perform better than the current best by about a factor of 2 when comparing the deviation from a lower bound.

6.1.2 Investigation of a Queueing Network with Flexible Servers

Chapters 3 and 5 examined a queueing network with flexible servers when setup times are zero and non-zero respectively. Jobs arrive to this network dynamically and are to be scheduled on a server. Each server requires different times to process a job, making the allocation of servers to jobs important for the performance of the system. In previous work, a round robin policy is devised which will guarantee stability of a system if it is possible [4]. We extend the objectives of this network to consider the flow time performance of jobs.

This dissertation presented two scheduling algorithms for the queueing network: *MinFT* and *MinMksp*. We demonstrated that the performance of the *MinFT* model which dispatches jobs greedily to minimize the flow time of each job works well in low load situations, but quickly degrades as the load increases. The greedy approach led to instability for a system

where the round robin policy was stable. Though *MinMksp* also considers only short term objectives, it is able to maintain stability similar to the round robin policy by maximizing the throughput periodically.

6.1.2.1 Guiding Scheduling with Queueing

Comparisons between the queueing and scheduling approaches illustrated that emphasis on long and short run goals are important to the stability and flow time performance of a dynamic system. By using the analysis from the queueing model to give the *MinMksp* model insight into long run system performance objectives, we are able to create a hybrid model which realizes the best performance overall for the simulated cases.

In the case where setup times did not exist, the insight gained from the queueing analysis helped the scheduling model by penalizing a deviation from the guidance of the queueing analysis. The penalty helped to align the short term schedule with long term goals improving overall performance.

When setup times are introduced to the problem, guidance through a penalty function becomes ineffective. The bad assignments a scheduler make can propagate to a significant portion of future jobs to reduce performance. In this case, guidance from the queueing analysis comes from a restriction of which classes of jobs can be assigned to each server. We also improve the *MinMksp* model by decreasing the reaction time of the model to newly arrived jobs. Making these two changes helped to drastically increase the performance allowing the new algorithm to achieve the best performance while also guaranteeing stability whenever possible.

6.1.3 Integration of Queueing Theory and Scheduling

Our case study of a queueing network displays how a systems can be dynamic and require combinatorial optimization. We show the potential of integrating the two methodologies by demonstrating that the queueing analysis can guide the scheduling model to better decisions in a queueing network. Our work helps to combine techniques from the two fields of research to

more accurately model a complex and dynamic system.

6.2 Future Work

In this section, we discuss avenues for future research which expand from the work in this dissertation.

6.2.1 Queueing Analysis Focused on Integration with Scheduling

The hybrid models presented in this dissertation make use of queueing analysis available from the literature and directly apply them to our scheduling model. Though there is improved performance, one would question whether better results can be achieved if the queueing analysis was designed with the scheduling model in mind. The guidance from the queueing analysis only tells the scheduler what is efficient for heavily loaded systems. However, the system is often not heavily loaded and slack will exist which the scheduler could theoretically use to improve performance.

We expect that a stronger hybrid approach could be obtainable if a queueing model can help decide when a scheduler should act greedily and when it should be conservative. An important issue is whether an idle server should process an available job immediately if the long run steady state analysis under heavy load chooses otherwise. If the server decides to serve the job and no new jobs arrive during the duration of processing the job, then immediate service was a good choice. However, if a job belonging to an efficient class arrives before the end of service of the inefficient job, the choice could be lead to increased flow time. Specifically, when setup times are nonzero, the immediate switch to a new class of jobs could further delay the wait of a new job since the server will need to spend time switching to another class, serving the jobs in that class, and then switching back to the previous class. Idling the server to await for new jobs can increase performance. Gaining insight into the arrival of a new jobs and servicing of a current job is vital to making good choices more consistently.

Designing queueing analysis to consider the load of the system is also of interest. The objectives of the queueing model is to maximize the capacity of a system. The analysis was only interested in systems with very high loads where achieving stability is not trivial. With a reduced load, the analysis becomes inaccurate and job performance measures are ignored. Re-evaluating the queueing model can help align a new analytical model to our goal of increasing performance when stability is not an issue.

6.2.2 Extensions to the Queueing Network studied in Chapter 3 and 5

Extensions of the dynamic environment studied is another potential direction.

6.2.2.1 Increased Complexity

The network studied by Andradóttir et al. [4] allows rerouting of jobs to different classes upon completion of service. This dissertation does not examine rerouting in the queueing network. With rerouting, the system can be specified to represent a dynamic version of many complex systems that are commonly studied in scheduling. Similarities to a flow shop environment can be observed if all arriving jobs are routed to a single class and after completing service, the job is routed to the next class repeatedly until the job reaches the last class and exits the system. To complete the representation for a dynamic flow shop, the system must limit each server to serving exactly one class and correspond the number of servers to the number of classes. Likewise, a job shop environment allows jobs to enter into multiple classes and routing may occur in many different directions.

The results of combining queueing and scheduling for the simpler problems in Chapters 3 and 5 show the potential of similar techniques on the dynamic job and flow shop. The increased complexity creates a more interesting problem that may further showcase the impact of scheduling in a complex dynamic environment as well as the benefits of long run considerations.

6.2.2.2 Limited Knowledge

Job processing times were assumed to be known as soon as jobs arrived to the system. While the assumption is valid for many systems, there are practical problems when such information is unavailable. The round robin policy does not require detailed information of processing times and does not make use of the processing time of jobs other than the expected rate at which the job classes are processed. In contrast, the scheduling model uses the exact processing time information to better distribute jobs and balance work across multiple servers.

6.3 Conclusion

The central thesis of this dissertation is that guidance from queueing analysis can be effective when scheduling in a dynamic environment. We illustrate the advantages of combining queueing and scheduling by providing a hybrid approach to a dynamic queueing network from the literature. The queueing and scheduling approaches are able to exhibit different characteristics which lead to better performance when situations favour one over the other. The integration of the methods achieves increased performance by maintaining the respective strengths of each model. Cooperation of queueing and scheduling may provide a framework that more accurately models real world problems requiring both the dynamic aspects of queueing theory and the combinatorial optimization techniques of scheduling.

Appendix A

Notation of Queueing Models

A queueing process is described by $A/B/X/Y/Z$, where A indicates the interarrival-time distribution, B the service pattern as described by a the probability distribution for service time, X the number of parallel service channels, Y the restriction on system capacity, and Z the queue discipline.

Characteristic	Symbol	Explanation
A/B	M	Exponential
	D	Deterministic
	E_k	Erlan type k ($k = 1, 2, ...$)
	H_k	Misture of k exponentials
	PH	Phase type
	G	General
X	$1, 2, ..., \infty$	
Y	$1, 2, ..., \infty$	
Z	FCFS	First come, first served
	LCFS	Last come, first served
	RSS	Random selection for service
	PR	Priority
	GD	General discipline

Appendix B

Details for the Experiments in Chapter 3

Job Class (k)	a_k	μ_{1k}	μ_{2k}
1	0.5	9	5
2	0.5	2	1

Table B.1: Problem Parameters for 2 machines and 2 customer classes.

Job Class (k)	δ_{1k}	δ_{2k}
1	0	0.5
2	1	0.5

Table B.2: Results of the Allocation LP for 2 machines and 2 customer classes.

Job Class (k)	a_k	μ_{1k}	μ_{2k}	μ_{3k}	μ_{4k}
1	0.25	3	2	1	1
2	0.25	1	3	1	2
3	0.25	1	3	3	1
4	0.25	1	1	1	1

Table B.3: Problem Parameters for 4 machines and 4 customer classes.

Job Class (k)	δ_{1k}	δ_{2k}	δ_{3k}	δ_{4k}
1	0.67	0	0	0
2	0	0.67	0	0
3	0	0.33	0.33	0
4	0..33	0	0.67	1

Table B.4: Results of the Allocation LP for 4 machines and 4 customer classes.

Appendix C

Details for the Experiments in Chapter 5

Job Class (k)	a_k	μ_{1k}	μ_{2k}	μ_{3k}	μ_{4k}
1	0.25	3	2	1	1
2	0.25	1	3	1	2
3	0.25	1	3	3	1
4	0.25	1	1	1	1

Table C.1: Problem Parameters for 4 machines and 4 customer classes.

Job Class (k)	δ_{1k}	δ_{2k}	δ_{3k}	δ_{4k}
1	0.67	0	0	0
2	0	0.67	0	0
3	0	0.33	0.33	0
4	0.33	0	0.67	1

Table C.2: Results of the Allocation LP for 4 machines and 4 customer classes.

		Server 1				**Server 2**				**Server 3**				**Server 4**			
		From Class															
		1	2	3	4	1	2	3	4	1	2	3	4	1	2	3	4
To Class	1	0	0.1	0.1	0.3	0.4	0	0.2	0.3	0.1	0.1	0	0.3	0.3	0.3	0.3	0
	2	0	0.2	0.3	0.1	0.3	0	0.1	0.1	0.2	0.1	0	0.1	0.1	0.1	0.2	0
	3	0	0.3	0.3	0.5	0.1	0	0.4	0.2	0.1	0.1	0	0.1	0.2	0.5	0.5	0
	4	0	0.5	0.5	0.1	0.2	0	0.2	0.1	0.2	0.3	0	0.2	0.2	0.2	0.4	0

Table C.3: Rate of setup times when they are long for 4 machines and 4 customer classes.

		Server 1				**Server 2**				**Server 3**				**Server 4**			
		From Class															
		1	2	3	4	1	2	3	4	1	2	3	4	1	2	3	4
To Class	1	0	1	1	3	4	0	2	3	1	1	0	3	3	3	3	0
	2	0	2	3	1	3	0	1	1	2	1	0	1	1	1	2	0
	3	0	3	3	5	1	0	4	2	1	1	0	1	2	5	5	0
	4	0	5	5	1	2	0	2	1	2	3	0	2	2	2	4	0

Table C.4: Rate of setup times when they are short for 4 machines and 4 customer classes.

Job Class (k)	a_k	μ_{1k}	μ_{2k}
1	0.25	2	2
2	0.25	1	1
3	0.25	3	1
4	0.25	1	2

Table C.5: Problem Parameters for 2 machines and 4 customer classes.

Job Class (k)	δ_{1k}	δ_{2k}
1	0.428	0
2	0.286	0.571
3	0.286	0
4	0	0.429

Table C.6: Results of the Allocation LP for 2 machines and 4 customer classes.

		Server 1		**Server 2**	
		From Class			
		1	2	1	2
To Class	1	0	0.1	0.4	0
	2	0	0.2	0.3	0
	3	0	0.3	0.1	0
	4	0	0.5	0.2	0

Table C.7: Rate of setup times when they are long for 2 machines and 4 customer classes.

		Server 1		**Server 2**	
		From Class			
		1	2	1	2
To Class	1	0	1	4	0
	2	0	2	3	0
	3	0	3	1	0
	4	0	5	2	0

Table C.8: Rate of setup times when they are short for 2 machines and 4 customer classes.

Bibliography

[1] J. Adams, E. Balas, and D. Zawack. The shifting bottleneck procedure for job shop scheduling. *Management science*, 34(3):391–401, 1988.

[2] A. Al-Salem. Scheduling to minimize makespan on unrelated parallel machines with sequence dependent setup times. *Engineering Journal of the University of Qatar*, 17:177–187, 2004.

[3] A. Allahverdi, J.N.D. Gupta, and T. Aldowaisan. A review of scheduling research involving setup considerations. *Omega*, 27(2):219–239, 1999.

[4] S. Andradóttir, H. Ayhan, and D.G. Down. Dynamic server allocation for queueing networks with flexible servers. *Operations Research*, pages 952–968, 2003.

[5] A.L. Arcus. A computer method of sequencing operations for assembly lines. *International Journal of Production Research*, 4(4):259–277, 1965.

[6] J.P. Arnaout, G. Rabadi, and R. Musa. A two-stage ant colony optimization algorithm to minimize the makespan on unrelated parallel machines with sequence-dependent setup times. *Journal of Intelligent Manufacturing*, pages 1–9, 2008.

[7] H. Aytug, M.A. Lawley, K. McKay, S. Mohan, and R. Uzsoy. Executing production schedules in the face of uncertainties: A review and some future directions. *European Journal of Operational Research*, 161(1):86–110, 2005.

[8] K.R. Baker. *Introduction to sequencing and scheduling*. John Wiley & Sons, 1974.

[9] J.C. Beck. Solution-guided multi-point constructive search for job shop scheduling. *Journal of Artificial Intelligence Research*, 29(1):49–77, 2007.

[10] J.C. Beck, TK Feng, and J.P. Watson. Combining constraint programming and local search for job-shop scheduling. *INFORMS Journal on Computing*, 23(1):1–14, 2011.

[11] J.C. Beck and N. Wilson. Proactive algorithms for job shop scheduling with probabilistic durations. *Journal of Artificial Intelligence Research*, 28(1):183–232, 2007.

[12] C.E. Bell. Optimal operation of an m/m/2 queue with removable servers. *Operations Research*, 28(5):1189–1204, 1980.

[13] J.F. Benders. *Partitioning in mathematical programming*. PhD thesis, Utrecht, 1960.

[14] J.F. Benders. Partitioning procedures for solving mixed-variables programming problems. *Numerische Mathematik*, 4(1):238–252, 1962.

[15] J.H. Blackstone, D.T. Phillips, and G.L. Hogg. A state-of-the-art survey of dispatching rules for manufacturing job shop operations. *International journal of production research*, 20(1):27–45, 1982.

[16] G. Brigham. On a congestion problem in an aircraft factory. *Journal of the Operations Research Society of America*, 3(4):412–428, 1955.

[17] J. Bruno, E.G. Coffman Jr, and R. Sethi. Scheduling independent tasks to reduce mean finishing time. *Communications of the ACM*, 17(7):382–387, 1974.

[18] C. Buyukkoc, P. Varaiya, and J. Walrand. The cμ-rule revisited. *Advances in Applied Probability*, 17(1):237–238, 1985.

[19] J. Carlier. The one-machine sequencing problem. *European Journal of Operational Research*, 11(1):42–47, 1982.

[20] J. Carlier and E. Pinson. An algorithm for solving the job-shop problem. *Management science*, 35(2):164–176, 1989.

[21] H. Chen and D.D. Yao. A fluid model for systems with random disruptions. *Operations Research*, pages 239–247, 1992.

[22] T.C.E. Cheng and C.C.S Sin. A state-of-the-art review of parallel-machine scheduling research. *European Journal of Operational Research*, 47(3):271–292, 1990.

[23] Y. Chu and Q. Xia. A hybrid algorithm for a class of resource constrained scheduling problems. In *Proceedings of the 2nd Conference on Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems*, pages 110–124. Springer, 2005.

[24] L.K. Church and R. Uzsoy. Analysis of periodic and event-driven rescheduling policies in dynamic shops. *International Journal of Computer Integrated Manufacturing*, 5(3):153–163, 1992.

[25] EG Coffman Jr, MR Garey, and DS Johnson. An application of bin-packing to multiprocessor scheduling. *SIAM Journal on Computing*, 7:1, 1978.

[26] R.W. Conway, W.L. Maxwell, and L.W. Miller. *Theory of scheduling*. Dover Pubns, 2003.

[27] J.F. Cordeau, G. Stojkovic, F. Soumis, and J. Desrosiers. Benders decomposition for simultaneous aircraft routing and crew scheduling. *Transportation science*, 35(4):375, 2001.

[28] P. Cowling and M. Johansson. Using real time information for effective dynamic scheduling. *European Journal of Operational Research*, 139(2):230–244, 2002.

[29] T.B. Crabill, D. Gross, and M.J. Magazine. A classified bibliography of research on optimal design and control of queues. *Operations Research*, pages 219–232, 1977.

[30] R.K. Deb. Optimal control of batch service queues with switching costs. *Advances in Applied Probability*, 8(1):177–194, 1976.

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

[32] S. Dunstall and A. Wirth. Heuristic methods for the identical parallel machine flowtime problem with set-up times. *Computers & Operations Research*, 32(9):2479–2491, 2005.

[33] M.M. Fazel-Zarandi and J.C. Beck. Solving a location-allocation problem with logic-based benders decomposition. In *Principles and Practice of Constraint Programming-CP 2009: 15th International Conference, CP 2009 Lisbon, Portugal, September20-24, 2009 Proceedings*, page 344. Springer, 2009.

[34] A. Federgruen and K.C. So. Optimality of threshold policies in single-server queueing systems with server vacations. *Advances in applied probability*, 23(2):388–405, 1991.

[35] F. Focacci, P. Laborie, and W. Nuijten. Solving scheduling problems with setup times and alternative resources. In *Proceedings of the 5th International Conference on Artificial Intelligence Planning and Scheduling*, 2000.

[36] P.M. França, M. Gendreau, G. Laporte, and F.M. Muller. A tabu search heuristic for the multiprocessor scheduling problem with sequence dependent setup times. *International Journal of Production Economics*, 43(2-3):79–89, 1996.

[37] D.K. Friesen. Tighter bounds for the multifit processor scheduling algorithm. *SIAM Journal on Computing*, 13:170, 1984.

[38] M.R. Garey, D.S. Johnson, and R. Sethi. The complexity of flowshop and jobshop scheduling. *Mathematics of Operations Research*, pages 117–129, 1976.

[39] CA Glass, CN Potts, and P. Shade. Unrelated parallel machine scheduling using local search. *Mathematical and Computer Modelling*, 20(2):41–52, 1994.

[40] T. Gonzalez and S. Sahni. Open shop scheduling to minimize finish time. *Journal of the ACM (JACM)*, 23(4):665–679, 1976.

[41] J. Grabowski and M. Wodecki. A very fast tabu search algorithm for the permutation flow shop problem with makespan criterion. *Computers & Operations Research*, 31(11):1891–1909, 2004.

[42] R.L. Graham. Bounds on multiprocessing timing anomalies. *SIAM Journal on Applied Mathematics*, 17(2):416–429, 1969.

[43] R.L. Graham, E.L. Lawler, J.K. Lenstra, and A.H.G.R. Kan. Optimization and approximation in deterministic sequencing and scheduling: A survey. *Annals of Discrete Mathematics*, 5(2):287–326, 1979.

[44] S.C. Graves. A review of production scheduling. *Operations Research*, 29(4):646–675, 1981.

[45] H.C. Gromoll. Diffusion approximation for a processor sharing queue in heavy traffic. *The Annals of Applied Probability*, 14(2):555–611, 2004.

[46] D. Gross and C. Harris. *Fundamentals of queueing theory*. Wiley Interscience, 1998.

[47] A. Guinet. Textile production systems: a succession of non-identical parallel processor shops. *The Journal of the Operational Research Society*, 42(8):655–671, 1991.

[48] R. Haupt. A survey of priority rule-based scheduling. *OR spectrum*, 11(1):3–16, 1989.

[49] M. Helal, G. Rabadi, and A. Al-Salem. A tabu search algorithm to minimize the makespan for the unrelated parallel machines scheduling problem with setup times. *International Journal of Operations Research*, 3(3):182–192, 2006.

[50] W. Herroelen and R. Leus. Project scheduling under uncertainty: Survey and research potentials. *European journal of operational research*, 165(2):289–306, 2005.

[51] J.N. Hooker. A hybrid method for the planning and scheduling. *Constraints*, 10(4):385–401, 2005.

[52] J.N. Hooker. Planning and scheduling by logic-based benders decomposition. *OPERATIONS RESEARCH-BALTIMORE THEN LINTHICUM-*, 55(3):588, 2007.

[53] JN Hooker and G. Ottosson. Logic-based Benders decomposition. *Mathematical Programming*, 96(1):33–60, 2003.

[54] WA Horn. Minimizing average flow time with parallel machines. *Operations Research*, 21(3):846–847, 1973.

[55] E. Horowitz and S. Sahni. Exact and approximate algorithms for scheduling nonidentical processors. *Journal of the ACM (JACM)*, 23(2):317–327, 1976.

[56] J.R. Jackson. Scheduling a production line to minimize maximum tardiness. Technical report, DTIC Document, 1955.

[57] J.R. Jackson. An extension of johnson's results on job idt scheduling. *Naval Research Logistics Quarterly*, 3(3):201–203, 1956.

[58] S.M. Johnson. Optimal two-and three-stage production schedules with setup times included. *Naval research logistics quarterly*, 1(1):61–68, 1954.

[59] JC Ke and KH Wang. Cost analysis of the m/m/r machine repair problem with balking, reneging, and server breakdowns. *Journal of the Operational Research Society*, 50(3):275–282, 1999.

[60] D.G. Kendall. Stochastic processes occurring in the theory of queues and their analysis by the method of the imbedded markov chain. *The Annals of Mathematical Statistics*, 24(3):338–354, 1953.

[61] D.W. Kim, K.H. Kim, W. Jang, and F. Frank Chen. Unrelated parallel machine scheduling with setup times using simulated annealing. *Robotics and Computer-Integrated Manufacturing*, 18(3-4):223–231, 2002.

[62] E. Kim and M.P. Van Oyen. Beyond the cμ rule: Dynamic scheduling of a two-class loss queue. *Mathematical Methods of Operations Research*, 48(1):17–36, 1998.

[63] ME Kurz and RG Askin. Heuristic scheduling of parallel machines with sequence-dependent set-up times. *International Journal of Production Research*, 39(16):3747–3769, 2001.

[64] J. Labetoulle, EL Lawler, JK Lenstra, and A.H.G.R. Kan. Preemptive scheduling of uniform machines subject to release dates. *Progress in combinatorial optimization*, page 245, 1984.

[65] G. Lancia. Scheduling jobs with release dates and tails on two unrelated parallel machines to minimize the makespan. *European Journal of Operational Research*, 120(2):277–288, 2000.

[66] E.L. Lawler. Sequencing jobs to minimize total weighted completion time subject to precedence constraints. *Algorithmic aspects of combinatorics*, 2:75–90, 1978.

[67] E.L. Lawler and J. Labetoulle. On preemptive scheduling of unrelated parallel processors by linear programming. *Journal of the ACM (JACM)*, 25(4):612–619, 1978.

[68] C.Y. Lee, L. Lei, and M. Pinedo. Current trends in deterministic scheduling. *Annals of Operations Research*, 70:1–41, 1997.

[69] J.K. Lenstra, A.H.G.R. Kan, and P. Brucker. Complexity of machine scheduling problems. *Studies in integer programming*, 1:343–362, 1977.

[70] H. Matsuura, H. Tsubone, and M. Kanezashi. Sequencing, dispatching and switching in a dynamic manufacturing environment. *International Journal of Production Research*, 31(7):1671–1688, 1993.

[71] G. McMahon and M. Florian. On scheduling with ready times and due dates to minimize maximum lateness. *Operations research*, 23(3):475, 1975.

[72] S.V. Mehta. Predictable scheduling of a single machine subject to breakdowns. *International Journal of Computer Integrated Manufacturing*, 12(1):15–38, 1999.

[73] A.S. Mendes, F.M. Muller, P.M. França, and P. Moscato. Comparing meta-heuristic approaches for parallel machine scheduling problems. *Production Planning & Control*, 13(2):143–154, 2002.

[74] L. Min and W. Cheng. A genetic algorithm for minimizing the makespan in the case of scheduling identical parallel machines. *Artificial Intelligence in Engineering*, 13(4):399–403, 1999.

[75] J.J. Moder and C.R. Phillips Jr. Queuing with fixed and variable channels. *Operations Research*, pages 218–231, 1962.

[76] C.L. Monma and J.B. Sidney. Sequencing with series-parallel precedence constraints. *Mathematics of Operations Research*, pages 215–224, 1979.

[77] C.L. Monma and J.B. Sidney. Optimal sequencing via modular decomposition: characterization of sequencing functions. *Mathematics of Operations Research*, pages 22–31, 1987.

[78] P.M. Morse and T. Teichmann. Queues, inventories, and maintenance. *Physics Today*, 11:33, 1958.

[79] E. Nowicki and C. Smutnicki. A fast taboo search algorithm for the job shop problem. *Management Science*, pages 797–813, 1996.

[80] E. Nowicki and C. Smutnicki. A fast tabu search algorithm for the permutation flow-shop problem. *European Journal of Operational Research*, 91(1):160–175, 1996.

[81] E. Nowicki and C. Smutnicki. An advanced tabu search algorithm for the job shop problem. *Journal of Scheduling*, 8(2):145–159, 2005.

[82] E. Nowicki and S. Zdrzalka. A note on minimizing maximum lateness in a one-machine sequencing problem with release dates. *European journal of operational research*, 23(2):266–267, 1986.

[83] D. Ouelhadj and S. Petrovic. A survey of dynamic scheduling in manufacturing systems. *Journal of Scheduling*, 12(4):417–431, 2009.

[84] M. Pinedo. *Scheduling: theory, algorithms, and systems*. Springer Verlag, 2008.

[85] AB Piunovskiy. Bicriteria optimization of a queue with a controlled input stream. *Queueing Systems*, 48(1):159–184, 2004.

[86] G. Rabadi. Scheduling research virtual center: http://schedulingresearch.com/, 2008.

[87] G. Rabadi, R.J. Moraga, and A. Al-Salem. Heuristics for the unrelated parallel machine scheduling problem with setup times. *Journal of Intelligent Manufacturing*, 17(1):85–97, 2006.

[88] J. Romaní. Un modelo de la teoria de colas con numero variable de canales. *Trabajos de Estadística y de Investigación Operativa*, 8(3):175–189, 1957.

[89] I. Sabuncuoglu and M. Bayiz. Analysis of reactive scheduling problems in a job shop environment. *European Journal of operational research*, 126(3):567–586, 2000.

[90] L. Schrage. A proof of the optimality of the shortest remaining processing time discipline. *Operations Research*, 16(3):687–690, 1968.

[91] S.V. Sevastianov and G.J. Woeginger. Makespan minimization in open shops: A polynomial time approximation scheme. *Mathematical Programming*, 82(1):191–198, 1998.

[92] M. Shahidehopour and Y. Fu. Benders decomposition: applying benders decomposition to power systems. *Power and Energy Magazine, IEEE*, 3(2):20–21, 2005.

[93] C.S. Shukla and F. Frank Chen. The state of the art in intelligent real-time fms control: a comprehensive survey. *Journal of Intelligent Manufacturing*, 7(6):441–455, 1996.

[94] J.B. Sidney and G. Steiner. Optimal sequencing by modular decomposition: Polynomial algorithms. *Operations research*, 34(4):606–612, 1986.

[95] C.E. Skinner. A priority queuing system with server-walking time. *Operations Research*, 15(2):278–285, 1967.

[96] W.E. Smith. Various optimizers for single-stage production. *Naval Research Logistics Quarterly*, 3(1-2):59–66, 1956.

[97] V. Suresh and D. Chaudhuri. Dynamic scheduling–a survey of research. *International Journal of Production Economics*, 32(1):53–63, 1993.

[98] L. Tadj and G. Choudhury. Optimal design and control of queues. *Top*, 13(2):359–412, 2005.

[99] E. Taillard. Some efficient heuristic methods for the flow shop sequencing problem. *European Journal of Operational Research*, 47(1):65–74, 1990.

[100] E. Taillard. Parallel taboo search techniques for the job shop scheduling problem. *ORSA journal on Computing*, 6:108–108, 1994.

[101] H. Takagi. Queuing analysis of polling models. *ACM Computing Surveys (CSUR)*, 20(1):5–28, 1988.

[102] D. Terekhov, D.G. Down, and J.C Beck. Stability of periodic scheduling approaches in systems with known processing times. Unpublished Manuscript, 2011.

[103] D. Terekhov, T.T. Tran, and J.C. Beck. Investigating Two-Machine Dynamic Flow Shops Based on Queueing and Scheduling. In *Proceedings of ICAPS2010 Workshop on Planning and Scheduling Under Uncertainty*, 2010.

[104] D. Towsley and S.K. Tripathi. A single server priority queue with server failures and queue flushing. *Operations research letters*, 10(6):353–362, 1991.

[105] P. Van Hentenryck and R. Bent. *Online Stochastic Combinatorial Optimization*. MIT Press, 2006.

[106] P.J.M. Van Laarhoven, E.H.L. Aarts, and J.K. Lenstra. Job shop scheduling by simulated annealing. *Operations research*, pages 113–125, 1992.

[107] G.E. Vieira, J.W. Herrmann, and E. Lin. Analytical models to predict the performance of a single-machine system under periodic and event-driven rescheduling strategies. *International Journal of Production Research*, 38(8):1899–1915, 2000.

[108] G.E. Vieira, J.W. Herrmann, and E. Lin. Rescheduling manufacturing systems: a framework of strategies, policies, and methods. *Journal of Scheduling*, 6(1):39–62, 2003.

[109] H.M. Wagner. An integer linear-programming model for machine scheduling. *Naval Research Logistics Quarterly*, 6(2):131–140, 1959.

[110] J. Walrand. *An introduction to queueing networks*, volume 21. Prentice Hall Englewood Cliffs, NJ, 1988.

[111] P. Wentges. Accelerating benders' decomposition for the capacitated facility location problem. *Mathematical Methods of Operations Research*, 44(2):267–290, 1996.

[112] A. Wierman and M. Harchol-Balter. Classifying scheduling policies with respect to unfairness in an m/gi/1. *ACM SIGMETRICS Performance Evaluation Review*, 31(1):238–249, 2003.

[113] A. Wierman, E.M.M. Winands, and O.J. Boxma. Scheduling in polling systems. *Performance Evaluation*, 64(9-12):1009–1028, 2007.

[114] W. Winston. Optimality of monotonic policies for multiple-server exponential queuing systems with state-dependent arrival rates. *Operations Research*, 26(6):1089–1094, 1978.

[115] S.D. Wu, R.H. Storer, and P.C. Chang. One-machine rescheduling heuristics with efficiency and stability as criteria. *Computers & Operations Research*, 20(1):1–14, 1993.

[116] Q. Xiao-long, T. Li-xin, and LIU Wen-xin. Dynamic scheduling: A survey of research methods [j]. *Control and Decision*, 2, 2001.

[117] T. Yamada and R. Nakano. Job-shop scheduling. *Genetic algorithms in engineering systems*, 1996.

[118] T. Yamada and R. Nakano. Job-shop scheduling by simulated annealing combined with deterministic local search. *Meta-heuristics: Theory and applications*, pages 237–248, 1996.