Journal of Artificial Intelligence Research 50 (2014) 535-572 Submitted 12/13; published 07/14

Integrating Queueing Theory and Scheduling for Dynamic Scheduling Problems

Daria Terekhov DTEREKHO@MIE.UTORONTO.CA
Tony T. Tran TRAN@MIE.UTORONTO.CA
Department of Mechanical and Industrial Engineering
University of Toronto, Toronto, Ontario, Canada

Douglas G. Down DOWND@MCMASTER.CA
Department of Computing and Software
McMaster University, Hamilton, Ontario, Canada

J. Christopher Beck JCB@MIE.UTORONTO.CA
Department of Mechanical and Industrial Engineering
University of Toronto, Toronto, Ontario, Canada

Abstract

Dynamic scheduling problems consist of both challenging combinatorics, as found in classical scheduling problems, and stochastics due to uncertainty about the arrival times, resource requirements, and processing times of jobs. To address these two challenges, we investigate the integration of queueing theory and scheduling. The former reasons about long-run stochastic system characteristics, whereas the latter typically deals with short-term combinatorics. We investigate two simple problems to isolate the core differences and potential synergies between the two approaches: a two-machine dynamic flowshop and a flexible queueing network. We show for the first time that stability, a fundamental characteristic in queueing theory, can be applied to approaches that periodically solve combinatorial scheduling problems. We empirically demonstrate that for a dynamic flowshop, the use of combinatorial reasoning has little impact on schedule quality beyond queueing approaches. In contrast, for the more complicated flexible queueing network, a novel algorithm that combines long-term guidance from queueing theory with short-term combinatorial decision making outperforms all other tested approaches. To our knowledge, this is the first time that such a hybrid of queueing theory and scheduling techniques has been proposed and evaluated.

1. Introduction

To behave intelligently over an extended period of time, a situated agent must be able to deal with dynamic changes in its tasks and goals. New tasks (e.g., targets for surveillance, see Burns, Benton, Ruml, Yoon, & Do, 2012; deliveries, see Bent & Van Hentenryck, 2007; requests for a drink, see Petrick & Foster, 2013) arrive continuously and must be incorporated into the ongoing problem-solving process.

Similarly, in a real-world scheduling problem, the set of jobs changes as customer orders arrive, are processed, and leave the system. Different jobs have different requirements for processing on different resources, and these characteristics often do not become known until the arrival of the job. Depending on the scheduling horizon, a scheduler may have a number of possibly conflicting objectives. Examples of short-term objectives include minimizing tardiness or makespan, while over the long term a scheduler may want to guarantee that the facility can handle the expected pattern of demand without catastrophic failures (e.g., without the number of jobs waiting for processing becoming extremely large).

Our thesis is that long-term stochastic reasoning as studied in queueing theory can be usefully combined, both theoretically and algorithmically, with shorter-term combinatorial reasoning that has been traditionally studied in scheduling within artificial intelligence (AI) and operations research. Such a combination is challenging, as queueing theory and scheduling have developed independently for many years and, as a consequence, have different performance measures of interest, standard problem settings, and assumptions. In this paper, we take steps toward the integration of queueing and scheduling by studying two dynamic scheduling problems and making the following contributions:

- We show that the fundamental queueing theory notion of *stability* can be used to analyze periodic scheduling algorithms, the standard approach to dynamic scheduling from the scheduling literature. We show, for each of the problems studied, that periodic scheduling algorithms can be proved to be maximally stable: no queueing policy or scheduling approach will allow the system to operate at a higher load or achieve a higher throughput.

- We show, in the context of one dynamic scheduling problem, that the long-term, stochastic reasoning of queueing theory can be combined with short-term combinatorial reasoning to produce a hybrid scheduling algorithm that achieves better performance than either queueing or scheduling approaches alone.

This paper is organized as follows. Section 2 provides the necessary background on dynamic scheduling algorithms, queueing theory, and stability. Next, we discuss our general problem settings and assumptions about job arrivals and the time at which various problem characteristics become known. Section 3 addresses our first, simpler problem, scheduling of a dynamic two-machine flowshop, presenting both our theoretical results on stability and our empirical results comparing algorithm performance. Section 4 turns to the second, more complex environment, a flexible queueing network. For the problem of scheduling in this setting, in addition to the theoretical and numerical examination of queueing and scheduling approaches, we propose and analyze a queueing/scheduling hybrid. In Section 5, we discuss the broader implications of our results for scheduling and AI problem solving. We conclude in Section 6.

2. Preliminaries

This section introduces the background that forms the context for our study. We review the general dynamic scheduling problem and two solution approaches: combinatorial scheduling and queueing theory. We then provide a detailed discussion of stability, the fundamental analytical concept we use in our theoretical contributions. Finally, we discuss our problem settings and assumptions.

2.1 Dynamic Scheduling Problems

The dynamic scheduling problems that we are interested in are characterized by a stream of jobs arriving stochastically over time. Each job requires a combination of resources, sequentially and/or in parallel, for different processing times. The existence of any particular job and its corresponding characteristics is not known until its arrival. However, we make the assumption, as in queueing theory but unlike in typical scheduling settings, that we have stochastic information of the distribution of job arrivals and characteristics. Jobs may require a complex routing through the available resources, which may be heterogeneous but with known and deterministic capacities.

Such a scheduling problem may have both short- and long-term objectives. In typical scheduling contexts, the goal is to construct a schedule that achieves an optimal level of performance for a given optimization criterion (e.g., mean flow time) for the jobs that actually arrive. Often we settle for evaluating algorithm performance over a finite time horizon; we refer to such criteria as *short-term* criteria. In contrast, *long-term* objectives focus on system-level performance measures such as stability, which establishes whether the instantaneous number of jobs will remain finite over an infinite horizon for particular system parameters.

To solve a dynamic scheduling problem, the jobs must be assigned to appropriate resources and start times, respecting the resource and temporal constraints. As jobs arrive, there must be an online process to make decisions: it is not possible to solve the entire problem offline.

Solving dynamic scheduling problems is challenging both due to the combinatorics of the interaction of jobs, resources, and time, and due to the stochastics: to make a decision, one can use only the information that is known with certainty at a decision point and the stochastic properties of scenarios that may occur in the future.

2.2 Combinatorial Scheduling

In the classical scheduling literature, it is common to assume *full information*: all jobs and their characteristics are known prior to any decision making. For example, the job shop scheduling problem (JSP) consists of $|J|$ jobs and $|M|$ resources (French, 1982). Each job consists of a set of $|M|$ completely ordered activities with a corresponding processing time and resource requirement: the job must use the specified resource for its full, uninterrupted processing time. While a number of objective functions have been defined, the most common is to minimize the makespan: the time between the start of the first activity and the end of the last one. Many solution approaches have been proposed, ranging from complete techniques such as branch-and-bound (Carlier & Pinson, 1989; Brucker, Jurisch, & Sievers, 1994), mixed integer programming (Bowman, 1959; Manne, 1960), and constraint programming (Fox, 1987; Baptiste, Le Pape, & Nuijten, 2001; Beck, 2007; Beck, Feng, & Watson, 2011), to dispatch rules (Pinedo, 2003) and meta-heuristics (Nowicki & Smutnicki, 1996, 2005).

The scheduling literature has also addressed problems with uncertainty. Pinedo (2003) refers to *stochastic* scheduling problems as those with random variables for the processing times and/or release dates (i.e., arrival times) of jobs. While acknowledging the similarity to the dynamic problems studied in queueing theory, Pinedo points out that a primary difference is that stochastic scheduling is typically concerned with optimizing the schedule for a finite number of jobs rather than the long-term system behaviour over a potentially infinite stream of arriving jobs (Pinedo, 2003, ch. 11). Approaches to solving such stochastic scheduling problems often involve the formulation of a determinstic scheduling problem to optimize the expected value or some other probabilistic measure of the objective function (Daniels & Carrillo, 1997; Pinedo, 2003; Beck & Wilson, 2007; Wu, Brown, & Beck, 2009), insertion of temporal or resource redundancy to cope with realizations of random variables (Leon, Wu, & Storer, 1994; Davenport, Gefflot, & Beck, 2001), or multi-stage stochastic programming (Herroelen & Leus, 2005).

When solving dynamic problems, the scheduling community typically adopts the periodic scheduling approach of solving a collection of linked static sub-problems (Bidot, Vidal, Laborie, & Beck, 2009; Ouelhadj & Petrovic, 2009). At a given time point, the static problem, consisting of the jobs currently present in the system, is solved to optimize some short-term objective function. That schedule is then executed (wholly or partially) until the creation of the next sub-problem is triggered. This viewpoint means that methods developed for static scheduling problems become directly applicable to dynamic ones. Such methods can effectively deal with complex combinatorics and can optimize the quality of schedules for each static sub-problem. However, they tend to overlook long-run performance and stochastic properties of the system.

There are a few examples of work that modifies the short-term objective or problem-solving process to address the fact that the static problem is part of the long-term scheduling problem. Branke and Mattfeld (2002, 2005) address a dynamic job shop problem with the overall objective of minimizing the mean tardiness of jobs. They use a periodic scheduling approach based on a genetic algorithm that solves each sub-problem to minimize a combination of tardiness with a preference to place resource idle time toward the end of the sub-problem schedule. The intuition is that later idle time allows jobs that arrive in the future to be "slotted in" to the current schedule with little disruption. Empirical results demonstrate a lower idle time over the long term compared to simply minimizing tardiness.

Another example is the framework of Online Stochastic Combinatorial Optimization (OSCO) (Van Hentenryck & Bent, 2006; Mercier & Van Hentenryck, 2007), where the set of existing jobs plus a sample of future arrivals is used to create a static scheduling sub-problem. Multiple samples and optimizations are intricately combined to arrive at a set of decisions for the existing jobs. Mercier and Van Hentenryck (2007) show that such algorithms scale better than Markov Decision Processes (MDPs) and result in good performance when there is a small expected difference between the best performance of clairvoyant and non-clairvoyant decision makers. Similar approaches have been developed in AI planning (Yoon, Fern, Givan, & Kambhampati, 2008; Burns et al., 2012).

These examples employ the same underlying approach to dynamic scheduling problems: adapt a static technique by incorporating intuitive (Branke & Mattfeld, 2002) or sampled (Van Hentenryck & Bent, 2006) information about the future. These approaches serve as an inspiration for us to develop a more formal and general understanding of how long-term objectives can be achieved by solving a series of short-term, static scheduling problems.

2.3 Queueing Theory

Queueing theory is the mathematical study of waiting lines (Gross & Harris, 1998). It models systems in which one or more servers (machines) at one or more service stations process arriving customer requests (jobs). Fundamentally, queueing theory focuses on the formal analysis of the evolution of the queues and related metrics over time given a particular system definition or class of systems. The standard notation to describe a queueing process, due mainly to Kendall (1953), is represented by $A/B/X/Y/Z$. A describes the inter-arrival process (distribution of time between two successive arrivals), B the service time distribution, X the number of parallel machines, Y the buffer size (maximum job capacity of the system), and Z the queue discipline (scheduling order). Common distributions found in the literature for A and B are deterministic[1] (D), exponential (M) and general (G). Given a defined arrival and service time distribution, queueing discipline, and number of machines and buffer capacity, the queueing literature is most commonly interested in determining steady-state system parameters such as stability conditions, expected waiting time and expected queue length. Although a significant portion of the literature focuses on these descriptive models for steady-state performance metrics, transient behaviour and prescriptive models are also studied, and are of interest in this paper. Markov processes (Down & Meyn, 1994; Dai & Meyn, 1995; Bolch, Greiner, de Meer, & Trivedi, 2006), fluid models (Dai, 1995; Dai & Weiss, 1996), and linear programming (LP) models (Andradóttir, Ayhan, & Down, 2003; Down & Karakostas, 2008) are typical approaches in analyzing queueing systems.

The area of queueing theory that deals with prescriptive models is frequently called the design and control of queueing systems (Tadj & Choudhury, 2005; Gross & Harris, 1998). In both queueing design and control problems, the goal is to find optimal values for the controllable parameters of the queue. These parameters include the number of machines available for processing arriving jobs, the buffer capacity, the arrival rate of jobs to the queue(s), the service rates of the machine(s), as well as any combination of these. Queueing design problems are static: once the optimal value of a controllable parameter is determined, it becomes a fixed characteristic of the queue. Queueing control problems, on the contrary, are dynamic: the goal in such problems is usually to determine an optimal action to take when the system is in a particular state. For example, consider a retail facility with workers who have to serve stochastically arriving customers and also perform back-room tasks which are independent of the customer arrival process (Terekhov, Beck, & Brown, 2009). In order to optimize the performance of such a facility, one has to solve the queueing design problem of finding the optimal number of cross-trained servers to employ as well as the related queueing control problem of determining when to switch these workers between the two task types. We refer the reader to the papers of Tadj and Choudhury (2005) and Crabill, Gross, and Magazine (1977) for overviews of design and control problems involving queues.

Queueing theory has taken the viewpoint that, since it is impossible to create an optimal schedule for every single sample path in the evolution of the system, one should aim to achieve optimal performance in some *probabilistic* sense (e.g., in expectation) over an infinite time horizon. This goal could be attained by construction of a policy based on the global stochastic properties of the system. For example, a policy could specify how start time

1. Deterministic here refers to all inter-arrival or service times having the same value.

assignments should be made whenever a new job arrives or completes processing. However, the schedule resulting from such a policy, while being of good quality in expectation, may be far from optimal for the particular realization of uncertainty that occurs. Moreover, queueing theory generally studies systems with simple combinatorics, as such systems are more amenable to rigorous analysis of their stochastic properties.

2.4 Stability

Stability[2] is a fundamental concept in queueing theory and forms the main part of our theoretical analysis. Informally, a system is *stable* if its queues remain bounded over time. The stability of a system is dependent on the scheduling policy it employs: for a given set of problem parameters (e.g., arrival and processing distributions), one policy may stabilize the system while another might not. Knowledge of whether a system is stable for a given job arrival rate, processing rate and scheduling policy is considered a precursor to more detailed analysis and is essential for practical applications (Kumar, 1994; Dai & Weiss, 1996).

Formally, a system operating under a particular queueing discipline is stable if the Markov process that describes the dynamics of the system is *positive Harris recurrent* (Dai, 1995). Positive Harris recurrence implies the existence of a unique stationary distribution. Due to the considerable notation required, we do not formally define positive Harris recurrence here, but instead refer the reader to the work of Dai (1995), Dai and Meyn (1995) and Bramson (2008).

In the special case when the state space of the Markov process is countable[3] and all states communicate, positive Harris recurrence is equivalent to the more widely-known concept of positive recurrence (Bramson, 2008). A Markov chain is positive recurrent if every state s is positive recurrent: the probability that the process starting in state s will return to s is 1, and the expected time to return to this state is finite (Ross, 2003). In particular, this property guarantees that the system will empty.

2.5 Problem Settings

Both queueing theory and combinatorial scheduling focus on the efficient use of scarce resources over time. However, their different emphases (i.e., stochastics vs. combinatorics) indicate that it may be possible to combine them in order to provide a better understanding of, and stronger solution approaches to, dynamic scheduling problems.

A first challenge in this study is to determine the problem settings and assumptions. As queueing and scheduling sometimes make differing assumptions, we have been guided by three heuristics in choosing problems. First, we sought the simplest problem settings where queueing and scheduling had not arrived at the same solution. For example, in many single machine dynamic scheduling problems, identical, optimal dispatch rules/queueing policies exist in both literatures. Second, where assumptions in the two areas are consistent, we

2. Unfortunately, *stability* has multiple meanings in closely related literature. In the scheduling literature, a predictive schedule is called *stable* if its execution is close to what was planned (Bidot et al., 2009). Similarly, in work on scheduling under uncertainty, stability analysis concerns the identification of the range of values that the processing times may take while the given schedule remains optimal (Sotskov, Sotskova, Lai, & Werner, 2010). Here we use the meaning of stability from queueing theory.
3. This process is referred to as a *continuous-parameter Markov chain* in the book by Gross and Harris (1998).

adopted them. And third, when assumptions are contradictory, we attempted to choose realistic settings that also result in novel and challenging problems.

As a consequence, we make the following assumptions in the two systems studied here.

- As in queueing, we assume that we know the distribution of the job inter-arrival times.
- We assume that the job durations become known upon arrival and are realizations of the corresponding known distributions. Queueing models typically assume that the distribution of job durations is known, but the actual processing time of a job is not available until its *completion* time.[4] In the classical scheduling literature, exact job durations are known prior to construction of a schedule and jobs processed on the same machine have, in general, different durations. As knowledge of both distributions and actual durations upon arrival can be justified by applications in which historical data is available and in which similar activities are repeatedly executed (e.g., the serving of the same static web page, the cutting of a piece of wood with standardized dimensions), we adopt both assumptions.
- For each queue, the sequences of processing times for all machines and the sequence of inter-arrival times are independent and identically distributed sequences. They are also mutually independent.
- Mean processing times and mean inter-arrival times are finite.
- The inter-arrival times are unbounded and continuous.

While the final three assumptions are standard in the queueing literature dealing with stability analysis (Dai, 1995) and satisfied by commonly used distributions (e.g., the exponential distribution), the combination of the first two assumptions typically has not received much attention in the queueing or scheduling literature. The setting is somewhat similar to the *online-time* setting in online scheduling (Pruhs, Sgall, & Torng, 2004) where jobs arrive to the system dynamically and job processing times become known upon arrival. However, that setting makes no assumption about known inter-arrival and duration distributional information.

3. The Two-Machine Dynamic Flowshop

The first problem we consider is a dynamic two-machine permutation flowshop. Within this setting, we demonstrate the stability of a periodic scheduling approach that uses processing time information, and provide a numerical comparison of queueing and scheduling approaches.

In a dynamic two-machine permutation flowshop, arriving jobs must be processed first on machine 1 and then on machine 2, and the order in which jobs are processed on the two machines must be the same. We assume that the inter-arrival time distribution is general with mean $\frac{1}{\lambda}$, while the processing time distributions for machine 1 and 2 are general with means $\frac{1}{\mu_1}$ and $\frac{1}{\mu_2}$, respectively. Thus, the load for machine 1 is $\rho_1 = \frac{\lambda}{\mu_1}$, and the load for

4. One exception is the *deterministic* distribution, under which the durations of all jobs are assumed to be identical (Gross & Harris, 1998).

machine 2 is $\rho_2 = \frac{\lambda}{\mu_2}$. Both ρ_1 and ρ_2 are assumed to be less than 1, as these are known necessary conditions for stability (Gross & Harris, 1998). Once a job arrives to the system, its processing times on both machines become known with certainty. Both machines are of unary capacity. Preemptions are not allowed. The queues in front of machine 1 and machine 2 are both assumed to be of infinite size. The goal of the problem is to sequence the jobs on machine 1 and machine 2 in order to minimize the flow time: the time between the arrival of a job and its completion on machine 2.

The dynamic flowshop (Park, 1988; Sarper & Henry, 1996; El-Bouri, Balakrishnan, & Popplewell, 2008) is an extension of the classical flowshop environment that has been extensively studied in the scheduling literature (Widmer & Hertz, 1989; Taillard, 1990). In the queueing literature, this system is known as a *tandem queue* or a network of queues in series and has also received significant attention (Burke, 1956; Reich, 1957; Jackson, 1957; Gross & Harris, 1998; Andradóttir & Ayhan, 2005).

3.1 Algorithms

We consider four periodic scheduling approaches to the two-machine dynamic flowshop problem. In each case, a schedule is formed by optimizing a value of interest for a *sub-problem*: the set of jobs present in the system at a particular time. In the dynamic flowshop, the start time of every new sub-problem is equal to the completion time of the last job on machine 1 in the previous sub-problem, as in Figure 1. Since the jobs in a new sub-problem can begin execution after all scheduled jobs have been processed on the *first* machine, at the time of scheduling there may still be jobs from a previous schedule that need processing on machine 2. These previous scheduling decisions are not altered when the next sub-problem is solved.

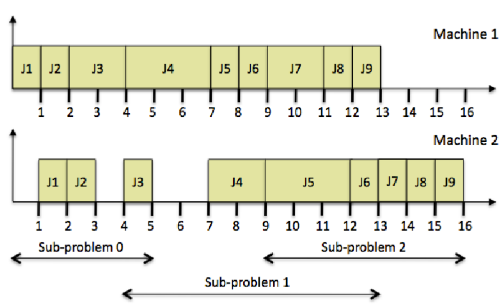

Figure 1: Dynamic flowshop with three sub-problems and three jobs per sub-problem. The start of a sub-problem is the start of a set of jobs on machine 1, the end of a sub-problem is the end of a set of jobs on machine 2.

To the best of our knowledge, no queueing policy has been proved to be optimal for the flow time objective, even in the expected sense, for a dynamic two-machine flowshop under our assumptions. Our first two approaches, however, are chosen because of theoretical results for systems related to ours, as discussed below. For the dynamic flowshop, none of our periodic scheduling methods make use of distributional information.

3.1.1 FCFS

Jobs in a sub-problem are sequenced in non-decreasing order of their arrival times to the system. *FCFS* achieves the smallest expected flow time in a two-machine dynamic flowshop in the class of work-conserving, non-preemptive policies that *do not use* processing time information (Towsley & Baccelli, 1991). Note that the periodic *FCFS* policy creates identical schedules to the standard (non-periodic) first-come, first-served policy in queueing theory, where jobs are processed in the order they arrive.

3.1.2 SPT$_{sum}$

Jobs in a sub-problem are processed in non-decreasing order of the *sum* of their durations on machine 1 and machine 2. This policy choice is motivated by the fact that, in the case when the server is a single unary resource, shortest processing time first minimizes the expected flow time (Wierman, Winands, & Boxma, 2007).

3.1.3 CompletionTime

Since minimizing sub-problem flow time under our assumptions is equivalent to minimizing the sum of job completion times in a sub-problem, a natural sub-problem objective is the sum of completion times of activities on the second machine. Optimizing the total completion time will lead to the best *short-run* performance but it is unclear how this method will perform with respect to *long-run* objectives. Minimizing the sum of completion times in a two-machine flowshop is NP-hard (Pinedo, 2003).

3.1.4 Makespan

The fourth method we employ is motivated by reasoning about a proxy measure that we speculate may result in strong *long-run* flow time performance. The minimum makespan schedule for a sub-problem, by definition, allows the subsequent sub-problem to start as early as possible on machine 2, implying potentially lower flow times for all future sub-problems until the system empties. Therefore, while we are not likely to achieve optimal flow time performance on each sub-problem, we conjecture that makespan minimization may lead to better long-run flow time performance.

The optimal makespan schedule for a static two-machine flowshop can be found in polynomial-time using Johnson's rule (Conway, Maxwell, & Miller, 1967). Johnson's rule divides jobs into two sets: set I consists of all jobs whose processing time on machine 1 is less than or equal to its processing time on machine 2, and set II consists of all the remaining jobs. Set I is processed before set II. Note that, as required, Johnson's rule creates permutation schedules. Within set I, jobs are sequenced in non-decreasing order of the processing time on machine 1, while within set II, jobs are sequenced in non-increasing order of the processing times on machine 2.

3.2 Stability

We study and compare the stability of the only one of our policies that does not use processing time information, i.e., *FCFS*, and one of our policies that does, i.e., *makespan.* In particular, we show that the condition for the stability of *FCFS* follows trivially from known results in the queueing literature. Using this result, more significantly, we show that the *makespan* approach is stable under the same condition. To the best of our knowledge, this is the first example of stability analysis of scheduling policies based on observed processing times.

We leave the study of stability of our two remaining methods that utilize processing time information for future work. Showing stability of the *completionTime* approach may prove to be especially challenging since it possesses neither a specific structure that can be utilized in the proof nor a property that makes it easily comparable to *FCFS*.

Theorem 1. *If* $\frac{\lambda}{\min\{\mu_1,\mu_2\}} < 1$*, then the periodic* FCFS *policy is stable for the two-machine dynamic flowshop.*

Proof. Under our assumptions, the dynamic two-machine flowshop is a generalized Jackson network, and the periodic *FCFS* policy is equivalent to the standard, non-periodic implementation of *FCFS*. Stability of generalized Jackson networks under the condition that the load of each machine is strictly less than 1 (in our case, $\lambda/\mu_1 < 1$ and $\lambda/\mu_2 < 1$) is known (Dai, 1995). □

The above result extends to $|M| > 2$ machines: fluid model methodology (Dai, 1995) can be used to show the stability of *FCFS* in an $|M|$-machine flowshop under the condition that $\frac{\lambda}{\min_{m\in\{1,\ldots,|M|\}}\{\mu_m\}} < 1$.

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

Lemma 1. *The amount of work completed by* $\hat{t}_n$ *is maximized by the* makespan *policy. That is,* $v_n \geq v_n^\pi$ *for all* n *and all non-idling* π*.*

Proof. For any arbitrary non-idling policy π, v_n^π can be written as $v_n^{1,\pi} + v_n^{2,\pi}$, where $v_n^{1,\pi}$ is the amount of work completed by $\hat{t}_n$ on machine 1 and $v_n^{2,\pi}$ is the amount of work completed by $\hat{t}_n$ on machine 2. (In Figure 2, $\hat{t}_0$ is 5, $v_0^{1,\pi} = 5$ and $v_0^{2,\pi} = 3$.) For the *makespan* approach, we use the notation without the superscript π, i.e., $v_n = v_n^1 + v_n^2$. In the dynamic flowshop, the amount of work completed by $\hat{t}_n$ on machine 1 is the same for any non-idling policy. Thus, it remains to prove that $v_n^2 \geq v_n^{2,\pi}$. We prove this by induction.
Base Case: $v_0^2 \geq v_0^{2,\pi}$ since any policy other than *makespan* can complete the set of initial jobs only at the same time as *makespan* ($\hat{t}_0$) or later.
Inductive Hypothesis: Assume the property is true for $\hat{t}_n$, that is, $v_n^2 \geq v_n^{2,\pi}$.
Inductive Step: We need to show the same property for $\hat{t}_{n+1}$, i.e., that $v_{n+1}^2 \geq v_{n+1}^{2,\pi}$. For the *makespan* approach, $v_{n+1}^2 = v_n^2 + \gamma((\hat{s}_n, \hat{s}_{n+1}])$, where $\gamma((\hat{s}_n, \hat{s}_{n+1}])$ is the total machine 2 workload that arrives in the time period $(\hat{s}_n, \hat{s}_{n+1}]$ and, therefore, is the workload that is processed in the sub-problem starting at $\hat{s}_{n+1}$ and ending at $\hat{t}_{n+1}$.

By the induction hypothesis, we know that at $\hat{t}_n$, $v_n^2 \geq v_n^{2,\pi}$. The amount of work processed on machine 2 by time $\hat{t}_{n+1}$ by policy π, $v_{n+1}^{2,\pi}$, equals the amount of work processed by π by time $\hat{t}_n$ plus some fraction of the difference in the amount of work completed by

π and *makespan* by $\hat{t}_n$ plus some fraction of the amount of machine 2 work that arrives in $(\hat{s}_n, \hat{s}_{n+1}]$. Thus, $v_{n+1}^{2,\pi} \leq v_n^{2,\pi} + (v_n^2 - v_n^{2,\pi}) + \gamma((\hat{s}_n, \hat{s}_{n+1}]) = v_n^2 + \gamma((\hat{s}_n, \hat{s}_{n+1}]) = v_{n+1}^2$. $\square$

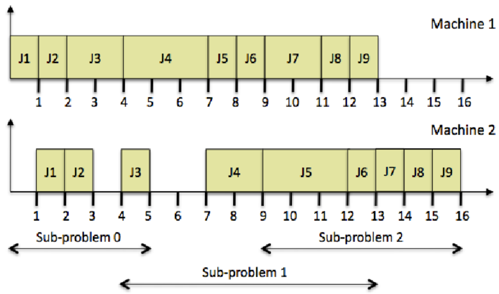

Figure 2: Schedule for policy π for a two-machine flowshop.

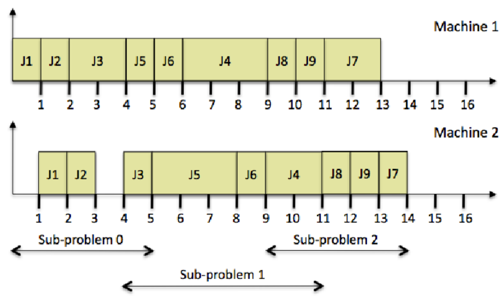

Figure 3: Schedule for *makespan* for the same problem instance as in Figure 2.

For example, consider the schedules in Figures 2 and 3: $\hat{s}_0 = 0$, $\hat{s}_1 = 4$, $\hat{s}_2 = 9$, $\hat{t}_0 = 5$, $\hat{t}_1 = 11$, $\hat{t}_2 = 14$, $v_2^2 = v_1^2 + \gamma((\hat{s}_1, \hat{s}_2]) = 9 + 3 = 12$, $v_2^{2,\pi} = 7 + (9 - 7) + 1 \leq 12$.[5]

The lemma does not hold if π is idling since an idling policy may create a better schedule by waiting and taking more jobs into account.

Theorem 2. *If* $\frac{\lambda}{\min\{\mu_1,\mu_2\}} < 1$ *then the periodic* makespan *policy is stable for the two-machine dynamic flowshop.*

Proof. We know that *FCFS* is stable under the given condition. By Lemma 1, at every sub-problem completion time, the *makespan* approach has finished at least as much work as *FCFS*. Therefore, the two-machine dynamic flowshop with the periodic *makespan* policy is stable under the same condition as *FCFS*. $\square$

The theorem provides a sufficient condition for stability. As noted above, from the literature (see e.g., Gross & Harris, 1998), we know that $\frac{\lambda}{\mu_1} < 1$ and $\frac{\lambda}{\mu_2} < 1$ are necessary conditions for stability of this system. Since ensuring that $\frac{\lambda}{\min\{\mu_1,\mu_2\}} < 1$ is the same as ensuring $\frac{\lambda}{\mu_1} < 1$ and $\frac{\lambda}{\mu_2} < 1$, we see that $\frac{\lambda}{\min\{\mu_1,\mu_2\}} < 1$ is a necessary and sufficient condition for stability of the dynamic two-machine flowshop under the *makespan* policy.

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

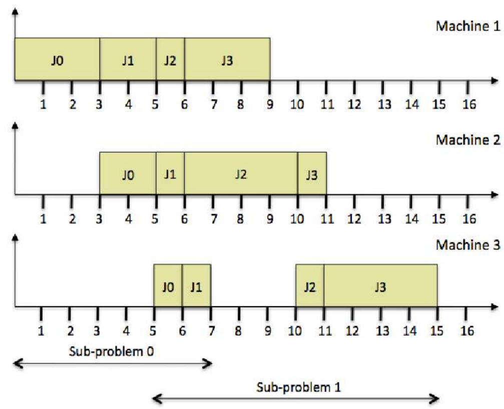

Figure 4: Schedule for *FCFS* for the dynamic flowshop with three machines.

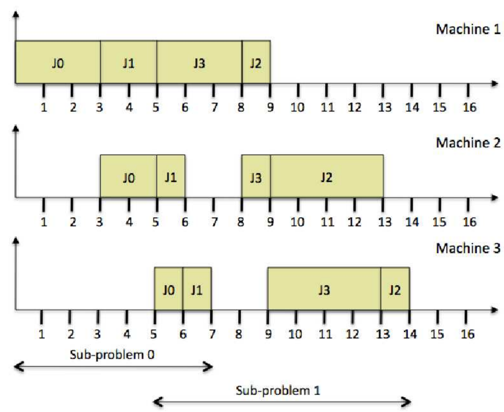

Figure 5: Schedule for *makespan* for the dynamic flowshop with three machines.

Lemma 1 does not hold for an $|M|$-machine flowshop when $|M| > 2$. To illustrate this fact, consider the problem instance in Figures 4 and 5. In this example, sub-problem 0 consists of jobs J0 and J1, and both the *makespan* policy and *FCFS* construct the same schedule, with $\hat{t}_0 = 7$. The second sub-problem consists of J2 and J3, and the two policies result in different schedules. At $\hat{t}_0 = 7$, the amount of work completed by *makespan* is $v_0 = 12$, whereas the amount of work completed by *FCFS* is $v_0^\pi = 13$. Therefore, in this case, $v_0 < v_0^\pi$, which shows that it is possible that the amount of work completed by $\hat{t}_n$ is *not* maximized by the *makespan* policy. We nonetheless conjecture that Theorem 2 extends to the case with more than two machines and can be proved using a fluid model approach.

3.3 Numerical Results

We present experiments comparing the performance of *FCFS*, SPT_{sum}, *makespan* and *completionTime* models for minimizing the mean flow time over a long time horizon. The *completionTime* model was implemented via constraint programming in ILOG Scheduler 6.5

and uses the *completion* global constraint (Kovács & Beck, 2011). The remaining methods were implemented using C++.

To evaluate the performance of our four methods in a dynamic flowshop, we considered a system with exponentially distributed inter-arrival times and exponentially distributed processing times with the same means on both machines. Experiments with uniformly distributed processing times showed identical performance. All parameters were chosen to satisfy the stability conditions of Theorems 1 and 2. We fixed the mean inter-arrival time, $1/\lambda$, to 10, and varied the load on the system from 0.1 to 0.95 by changing the means of the processing time distributions from 1 to 9.496. Note that by Theorems 1 and 2, these parameters guarantee stability of *FCFS* and *makespan*. The results of our experiments are shown in Figure 6. Each point in the figure represents the mean flow time over 100 problem instances of 55,000 jobs each.

In these experiments, the *completionTime* model was run with a time limit of 1 second per sub-problem in order to ensure reasonable run-times. Moreover, since constraint programming is more efficient with integer, rather than real-valued, durations, when using the *completionTime* approach we set durations to be the ceiling of the actual durations multiplied by 100; the resulting processing sequence is then used to construct the actual sub-problem schedule. Note that with a time limit, *completionTime* does not always find the optimal sub-problem schedule. When run on a Dual Core AMD 270 CPU with 1 MB cache, 4GB of main memory, and Red Hat Enterprise Linux 4, the *completionTime* model is able to solve 100% of sub-problems to optimality at loads 0.1 and 0.3, but this percentage decreases to an average of 81.3% for instances at the 0.95 load. This change in performance is due to the increase in sub-problem size as load increases: the average sub-problem size increases from 1.009 at the 0.1 load to 5.452 at the 0.95 load, while the maximum sub-problem size encountered over all instances increases from 5 at the 0.1 load to 172 at the 0.95 load. Experiments with a time limit of 5 seconds showed very similar performance, with the mean flow time at the 0.95 load decreasing to 368.985 (from 371.434 for the *completionTime* model with the 1 second time limit) and the percentage of instances solved to optimality at the 0.95 load increasing to 83.8% on a machine with 4 Intel Core i7-2600 CPUs @ 3.40GHz with 8 MB cache and 9871 MB main memory, running Red Hat Enterprise Linux 6.3.

Figure 6 shows that there is very little difference among the methods. *completionTime* has a slight advantage over the others for loads of 0.7 and less. The *makespan* model is better than *FCFS* and *completionTime* at loads above 0.8. SPT_{sum} is the best-performing model for loads above 0.8, when the static sub-problems become large. This last observation is supported by the results of Xia, Shanthikumar, and Glynn (2000), who have shown that SPT_{sum} is asymptotically optimal for the static average completion time objective as the number of jobs in a two-machine flowshop increases. However, this asymptotic result does not imply that applying SPT_{sum} to each sub-problem results in the best long-run behaviour for high loads. *FCFS*, on the other hand, is the worst performer over all loads, but only marginally so.

These empirical results are not contradictory to Theorem 1. In particular, the fact that *makespan* results in the largest amount of workload being completed by a particular time point does *not* imply that it will minimize the mean flow time. For example, consider sub-problem 1 in Figure 3: both the sequence displayed in the figure, J5 $\rightarrow$ J6 $\rightarrow$ J4, and

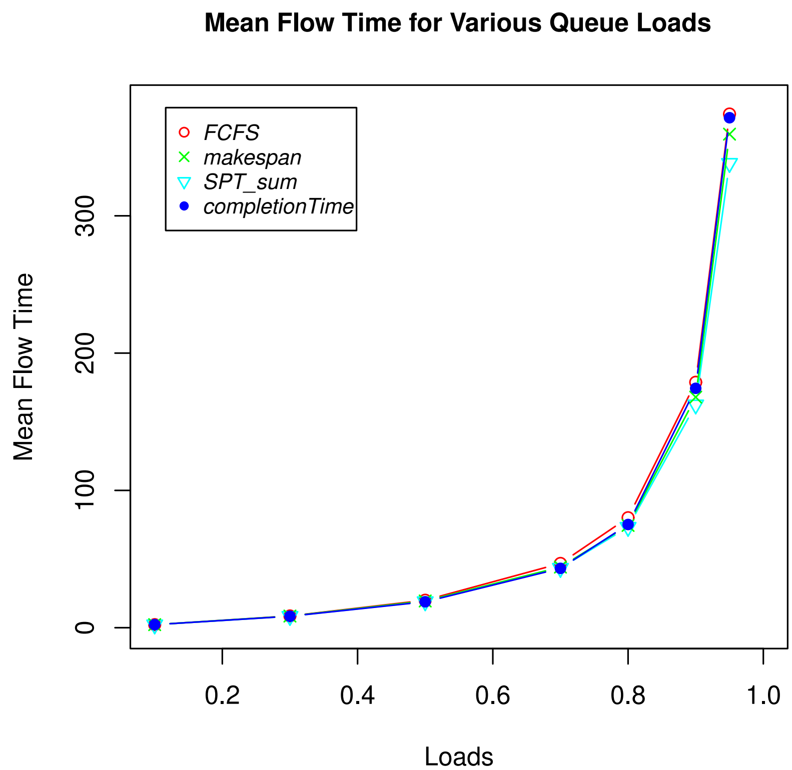

Figure 6: Mean flow times in a dynamic two-machine flowshop for *FCFS*, SPT_{sum}, *completionTime* and *makespan* models as the system load varies.

the sequence J6 → J5 → J4 will minimize makespan, but the latter sequence will achieve a lower total completion time (26 as opposed to 28 for the first sequence) and hence a lower mean flow time for this sub-problem.

3.4 Discussion

Our study of the dynamic two-machine flowshop provides partial support for our thesis of the utility of combining queueing and scheduling. We have shown that stability analysis can be applied to periodic scheduling algorithms and that algorithms that reason about combinatorics can achieve the same stability guarantees as traditional queueing theory approaches. However, in this system we were not able to empirically demonstrate that the scheduling algorithms (*completionTime* and *makespan*) result in better mean flow time performance than queueing approaches.

At low loads, all methods perform equivalently due to small sub-problems; at high loads, the differences in performance lead to the following observations. Firstly, the performance of *completionTime* degrades due to its inability to find good solutions for large sub-problems given the time limit; if it is not able to find a better solution, *completionTime* defaults to the *FCFS* schedule. Secondly, our motivation for using the *makespan* approach did not materialize: the differences between the makespans of the *makespan* approach and SPT_{sum} are not large enough to offset the differences in total completion time (which are in favour of SPT_{sum}). Finally, we conjecture that an implementation of *completionTime* which defaults to the SPT_{sum} schedule would outperform the current approaches at the highest load. A further improvement should result from finding, for each sub-problem, the optimal total completion time schedule with the lowest possible makespan.

Subsequent analytical work on this problem setting shows that the *completionTime* approach results in a smaller long-run mean flow time over T time periods than *makespan* if, for the majority of sub-problems, the difference between *completionTime* and *makespan* in individual sub-problem total completion times is larger than the difference in the makespans prior to the start of the sub-problem. Furthermore, in a more complex problem setting (i.e., a polling system with a two-machine flowshop server) it is shown, both analytically and empirically, that for the objective of minimizing long-run flow time, an algorithm that minimizes the makespan of the sub-problems outperforms one that minimizes the sum of completion times (Terekhov, 2013, ch. 7). While these results do not directly address the combination of queueing and scheduling approaches, they demonstrate that the impact of sub-problem optimization criteria on long-term performance measures is delicate and not yet well understood.

4. A Queueing Network with Flexible Servers

To more deeply investigate the combination of queueing theory and scheduling, we examine a second system with a different combinatorial structure. As in the first problem, we provide an analysis of stability and a numerical comparison of queueing and scheduling approaches. However, unlike in the first problem, in this setting we also propose and evaluate a hybrid queueing/scheduling algorithm.

The system of interest is a queueing network where heterogeneous servers are required to serve jobs belonging to specific classes. This problem is an example of a classical queueing system (Bramson & Williams, 2000; Andradóttir et al., 2003). The system differs from the dynamic flowshop in that jobs may be processed on any machine, and the assignment of the job to a machine must be made. Jobs arrive over time via an arrival process with rate λ and generally distributed inter-arrival times. An arriving job belongs to a class k with probability p_k. We assume that there are no preemptions: once a job begins execution it must be processed for its full processing time. When a server m is assigned to a job of class k, the job's processing time is generally distributed with rate μ_{mk}; once a job j arrives to the system, the processing times on each server become known and are denoted by d_{mj}. We assume that all servers are able to serve jobs of all classes. If multiple servers are working on a single class at any point in time, they work in parallel so that each job is served by exactly one server.

The equivalent static problem from the scheduling literature is the parallel machine scheduling problem (Pinedo, 2003). However, the scheduling literature either assumes machines are related or unrelated. In a related parallel machine scheduling problem, each machine has an inherent speed that determines job processing times. If machine a is twice as fast as machine b, it will always require half the time to process any job than machine b does. When machines are unrelated, there is no correlation in processing times of a job on the different machines. In the queueing network problem, processing times for a given job class are stochastically related because they are drawn from the same distribution on a given machine. However, there are no class relationships across different machines so it may be that $\mu_{ok} < \mu_{mk}$ but $\mu_{ol} > \mu_{ml}$ for $m \neq o, k \neq l$. We adopt the queueing theory assumption.

4.1 Algorithms

We present five different approaches for scheduling jobs in the queueing network with flexible servers: two queueing policies, two scheduling policies, and a hybrid queueing theory and scheduling method.

To illustrate the five approaches, we present a small example of a possible realization for a system with two servers and two job classes. There are a total of five jobs: three belonging to class 1 and two belonging to class 2. Each job is represented as J$\{k,j\}$, where k represents the class of a job and j is the job number. Table 1 provides the arrival and processing times on the two machines (servers) for each job in our example. Notice that both servers will process class 2 jobs for the same amount of time, but server 2 is 1.5 times slower than server 1 on jobs of class 1.

{Class, Job ID}	J{1,1}	J{1,2}	J{1,3}	{2,1}	{2,2}
Arrival Time	1	3	5	0	12
Server 1 Processing Time	8	6	4	6	6
Server 2 Processing Time	12	9	6	6	6

Table 1: Example job arrival and processing times.

The two queueing policies do not employ a periodic scheduler. Instead, they are dispatch rules used to decide which job should be processed when a server becomes available. In both policies, a linear program (LP), called the *allocation linear program*, is used to determine the portion of total capacity that each server should allocate to each job class (Andradóttir et al., 2003). This LP is solved only once, before any scheduling decisions are made. The allocation LP is derived from a fluid representation of the queueing network that assumes that the number of jobs present in the system is so large as to exhibit properties of a continuous fluid (Chen & Yao, 1992; Dai, 1995). The allocation LP is as follows:

$$\begin{aligned}
\max \quad & \lambda \\
\text{s.t.} \quad & \sum_{m=1}^{|M|} \delta_{mk}\mu_{mk} \geq \lambda p_k, && k \in K && (1)\\
& \sum_{k=1}^{|K|} \delta_{mk} \leq 1, && m \in M && (2)\\
& \delta_{mk} \geq 0 && k \in K;\, m \in M && (3)
\end{aligned}$$

where

λ: the arrival rate of jobs,
δ_{mk}: the fractional amount of time that server m should spend on class k jobs,
K: the set of job classes,
M: the set of servers.

The decision variables for this problem are λ and δ_{mk}. The objective is to maximize the arrival rate of jobs while maintaining stability of the system. Stability is achieved through constraint (1) which ensures that the total capacity allocated to a job class will be greater than or equal to the total amount of work generated from that job class.

The solution of the allocation LP provides a tight upper bound on the maximum arrival rate for which the system can be stabilized, λ^*, and values for the resource allocation proportions, δ^*_{mk} (Andradóttir et al., 2003). However, it does not provide a method to assign jobs to servers and does not decide how to sequence the jobs. To make these decisions, the *Round Robin* policy (Andradóttir et al., 2003) or the linear programming-based affinity scheduling (*LPAS*) heuristic (Al-Azzoni & Down, 2008) can be used.

4.1.1 Round Robin

In the *Round Robin* policy, each server m cyclically visits classes in V_m, where V_m is an ordered list of all classes k with $\mu_{mk}\delta^*_{mk} > 0$. While serving a class $k \in V_m$, the server processes at most l_{mk} jobs before moving on to the next class. If the server is idle at any point before l_{mk} jobs are served, it switches to the next class in V_m. Given a desired arrival rate λ, let $\epsilon = \frac{\lambda^*-\lambda}{\lambda^*+\lambda}$, $\xi_{mk} = \frac{1}{\mu_{mk}}$, and let $1\{\delta_{mk} > 0\}$ be a function that is equal to 1 if $\delta_{mk} > 0$ and 0 otherwise. Andradóttir et al. (2003) show that the δ^*_{mk} values are sufficiently tracked to stabilize the system if l_{mk} is chosen to be

$$l_{mk} = \left\lceil \frac{(1-\epsilon)(\sum_{l \in K} \xi_{ml} 1\{\delta^*_{mk} > 0\})\delta^*_{mk}}{\epsilon \xi_{mk}} \right\rceil.$$

Figure 7 shows the schedule produced by the *Round Robin* policy for the five-job example. Assume that the allocation LP results in $\delta^*_{11} = 1$, $\delta^*_{12} = 0$, $\delta^*_{21} = 0.5$, and $\delta^*_{22} = 0.5$. That is, server 1 handles jobs of class 1 only, and server 2 can handle both classes. Further, assume the calculated l_{mk} value to be greater than three for all server and class pairs. In this situation, for our example with at most three jobs per class, the *Round Robin* policy will change classes that a server is processing only if there are no available tasks of that class to start immediately. At time 0, only J{2,1} is present in the system. Since server 2 is the only server to handle class 2 jobs, the job is assigned to begin immediately on server 2. When J{1,1} arrives, server 1 is able to start this job instantly. Server 2 completes J{2,1} at time 6, when there are two jobs of class 1 in queue. Server 2 then begins processing J{1,2} since it arrived first. Server 1 completes J{1,1} at time 9 and starts processing J{1,3}. Server 1 completes J{1,3} before server 2 completes J{1,2}. Although J{2,2} is waiting in queue at time 13, it is not assigned to server 1 because $\delta^*_{12} = 0$. J{2,2} must wait until server 2 is available at time 15. The example schedule ends at time 21 with the completion of J{2,2}.

4.1.2 LPAS

LPAS assigns each arriving job to a machine that has $\delta^*_{mk} > 0$ and is expected to complete the job at the earliest time. The *LPAS* heuristic is similar to *Round Robin*, but does not

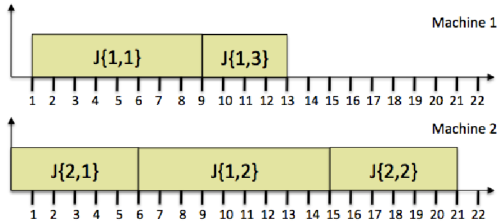

Figure 7: Example schedule produced by the *Round Robin* policy and *LPAS*.

reason about the relative magnitudes of δ^*_{mk} to guarantee stability. Rather, the heuristic uses δ^*_{mk} to reason only about which server-class pairs are efficient assignments. The stability of *LPAS* is an open question, though it has been shown empirically to perform well in terms of the number of jobs in the system (Al-Azzoni & Down, 2008).

Figure 7 also presents the schedule produced by *LPAS* for the five-job example. Similar to the *Round Robin* policy, we assume the allocation LP results in server 1 processing class 1 only, and server 2 handling both classes. At time 0, J{2,1} arrives and is assigned to server 2. J{1,1} arrives at time 1 and can be processed by either server. Starting immediately on server 1 would lead to an earlier completion time for J{1,1}, so we make that assignment. J{1,2} arrives at time 3 and is also considered by both servers. Starting J{1,2} on server 2 at time 6 leads to the earliest completion time, and hence this assignment is made. J{1,3} then arrives at time 5 and can complete earliest if assigned to server 1. Finally, J{2,2} must be assigned to server 2 at the end of the schedule. Therefore, the resulting schedule for this particular example is the same for the *Round Robin* policy and for *LPAS*.

Of the two scheduling models we study, one is a simple dispatch policy similar to that of the queueing policies presented above, while the second makes use of a periodic scheduler.

4.1.3 MinFT

The dispatch policy *MinFT* greedily minimizes the flow time of jobs without exploiting the solution of the allocation LP. In this setting, the flow time of a job j, if it is assigned to server m, is $f_{mj} = \nu_m + \sum_{x \in \Gamma_m} d_{mx} + d_{mj}$, where ν_m represents the total remaining time that server m will be busy with the job currently being served, and $\sum_{x \in \Gamma_m} d_{mx}$ is the sum of the processing times of all jobs belonging to Γ_m, the set of jobs queued for server m. When a job arrives, it is assigned to the server which results in the smallest flow time for that job given the jobs already scheduled. Ties are broken arbitrarily. Jobs are sequenced in first-come, first-served (FCFS) order on each server.

The schedule produced by *MinFT* for the five-job example is shown in Figure 8. At the start of the schedule, J{2,1} is considered by both servers, which leads to the same completion time on either, so *MinFT* arbitrarily chooses one of the servers. In this case, we assume servers are chosen lexicographically, and so the job is assigned to server 1. At time 1, assigning J{1,1} to server 2 will lead to a smaller flow time, and hence this assignment is made. This process continues for each job, which results in the schedule of Figure 8.

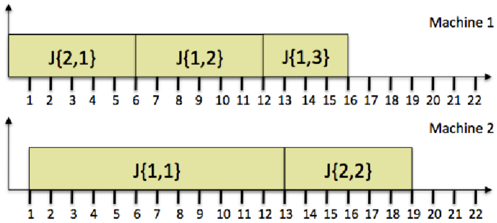

Figure 8: Example schedule produced by *MinFT*.

4.1.4 Makespan

As in the flowshop studied above, the *makespan* model periodically solves static makespan minimization problems. A period is defined as the time from when the scheduler evaluates the system until any server is once again available. Any jobs belonging to a previous period but not yet completed stay assigned as before.

This problem is NP-hard, and so there are no polynomial-time algorithms to minimize the makespan as in the two-machine dynamic flowshop problem. We use mixed-integer programming (MIP) to solve the minimum makespan scheduling problem. At the beginning of every period, the following MIP model is solved to minimize the makespan of the set of unscheduled jobs:

$$
\begin{aligned}
\min \quad & C_{max} & & \\
\text{s.t.} \quad & \sum_{j=1}^{|J|} x_{mj} d_{mj} + \nu_m \leq C_{max}, & m \in M & \quad (4)\\
& \sum_{m=1}^{|M|} x_{mj} = 1, & j \in J & \quad (5)\\
& x_{mj} \in \{0, 1\}, & j \in J;\ m \in M & \quad (6)
\end{aligned}
$$

where

C_{max}: makespan of the period,
x_{mj}: 1 if job j is assigned to server m, 0 otherwise,
ν_m: the remaining time that server m is busy serving jobs belonging to schedules made during previous periods,
J: the set of jobs to be scheduled during the period.

This model minimizes the makespan, C_{max}, which is constrained to be greater than or equal to the maximum scheduled busy period of a server. ν_m is the amount of time that server m will be busy until it is able to serve new jobs, and $\sum_{j=1}^{|J|} x_{mj} d_{mj}$ is the total processing time allotted to the new jobs by server m. The sum of these two terms equals the total time the server will be busy by following the schedule. Constraint (5) ensures that each job is assigned to exactly one server.

The MIP model assigns jobs to servers but does not sequence the jobs. Any order will achieve the same makespan since processing times of all jobs are sequence independent. To allow a direct comparison with the policies defined above, FCFS is used once jobs have been assigned to machines.

Figure 9 gives the schedule produced by *makespan* for the five-job example. At time 0, there is only 1 job available, so *makespan* schedules J{2,1}. Assignment to server 1 or 2 leads to the same makespan, so we arbitrarily choose server 1. When J{1,1} arrives, a new sub-problem is created since server 2 is idle and can begin processing jobs immediately. Assigning the arriving job to server 2 leads to minimizing the makespan of sub-problem 1. At time 6, server 1 becomes idle and finds two jobs in queue (J{1,2} and J{1,3}). Minimizing the makespan of sub-problem 2 requires J{1,2} and J{1,3} to be assigned to server 1. The final sub-problem starts at time 13 when server 2 finishes J{1,1}. At this point, there is one job in the queue, and it is assigned to server 2 to minimize the makespan of sub-problem 3.

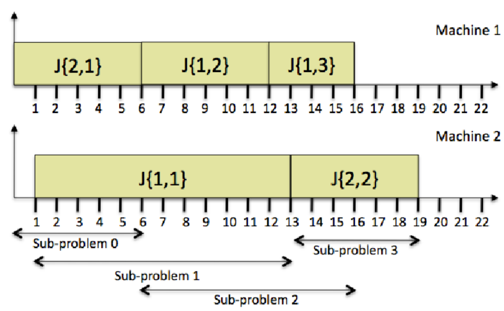

Figure 9: Example schedule produced by the *makespan* model.

MinFT does not reason about future jobs, while *makespan* can be thought of as only indirectly saving resources for future jobs by minimizing the makespan of the current period. The ability of the *makespan* model to deal with complex combinatorics allows it to take advantage of a system's state information, while *Round Robin* and *LPAS* are able to use knowledge of processing and arrival rates to manage the allocation of servers. An approach that is able to reason with both types of information may improve overall performance. Thus, we integrate scheduling and queueing theory on an algorithmic level to create a hybrid model.

4.1.5 Hybrid Model

The hybrid model is a periodic scheduler that uses the *makespan* model as a framework and employs the δ^*_{mk} values from the allocation LP. Recall that these values indicate the best long-run proportional allocation of servers to job classes. The MIP model solved for each period is:

$$\begin{aligned}
\min \quad & \alpha C_{max} + (1-\alpha) \sum_{m=1}^{|M|} \sum_{k=1}^{|K|} c_{mk} \\
\text{s.t.} \quad & \text{Constraints (4) to (6)} \\
& \sum_{j \in S_k} x_{mj} d_{mj} - \delta^*_{mk} \sum_{j=1}^{|J|} x_{mj} d_{mj} \leq c_{mk}, \quad k \in K;\ m \in M \quad (7) \\
& c_{mk} \geq 0, \quad k \in K;\ m \in M \quad (8)
\end{aligned}$$

where
- c_{mk}: the deviation between a realized assignment of server m to class k and what δ^*_{mk} suggests,
- α: an input parameter used to scale the importance of deviation from the δ^*_{mk} values versus makespan,
- S_k: the set of jobs that belong to class k.

This MIP model has a bi-criteria objective of minimizing the makespan and the deviation from the δ^*_{mk} values. The trade-off between makespan and deviation is expressed by α, $0 \leq \alpha \leq 1$. When $\alpha = 1$, the model is the *makespan* model.

Figure 10 illustrates the schedule that the hybrid model might produce for the five-job example assuming the same δ^*_{mk} values as before, i.e., $\delta^*_{11} = 1$, $\delta^*_{12} = 0$, $\delta^*_{21} = 0.5$, and $\delta^*_{22} = 0.5$, and assuming α is chosen to be close to 1. Using a high α value makes our illustration of this model's behaviour simpler. Further, this choice ensures that minimizing makespan is the more important objective, with δ^*_{mk} values being used only for breaking ties when different schedules produce the same makespan. At time 0, sub-problem 0 is solved with only J{2,1} in the system. Although the makespan is the same regardless of which server is responsible for the job, the hybrid model chooses to make an assignment to server 2, because $\delta^*_{12} = 0$ and $\delta^*_{22} = 0.5$. Sub-problem 1 starts at time 1 when J{1,1} arrives and is assigned to server 1. At time 6, J{1,2} and J{1,3} are in queue, and sub-problem 2 starts. J{1,2} is assigned to server 1 and J{1,3} is assigned to server 2 to minimize the makespan of sub-problem 2. At time 12, sub-problem 3 begins as J{2,2} is scheduled on server 2.

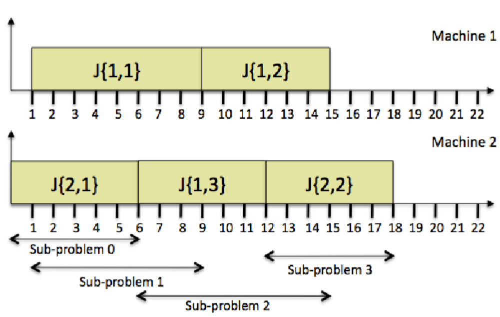

Figure 10: Example schedule produced by the hybrid model.

4.2 Stability

We know from the work of Andradóttir et al. (2003) that the *Round Robin* policy is stable for all $\lambda < \lambda^*$, the maximum stabilizable arrival rate. The stability of the *LPAS* heuristic has not yet been established.

We examine the stability of the two scheduling policies and the hybrid model presented above. The stability conditions for *makespan* and the hybrid model are determined through a comparison with the *Round Robin* policy. The *MinFT* model is shown to *not* guarantee stability under the same conditions as the *Round Robin* policy.

4.2.1 Stability of *Makespan*

Given that *Round Robin* can achieve any desired capacity $\lambda < \lambda^*$, we have the following theorem.

Theorem 3. *If* Round Robin *is stable for a given arrival process with rate λ, then the* makespan *model is also guaranteed to be stable.*

Proof. We prove the stability conditions of *makespan* by assuming that *makespan* is unstable and *Round Robin* is stable, and deriving a contradiction. Let $\hat{J}_t$ be the set of jobs in the system at time t under the *makespan* policy. Our assumption that *makespan* (denoted with a $\hat{}$ in our notation) is unstable means that $\lim_{t\to\infty} E(|\hat{J}_t|) = \infty$.

At some time t', the *makespan* model completes a period and schedules the set $\hat{J}_{t'}$. Denote the earliest start time and latest end time of jobs in $\hat{J}_{t'}$ under the *makespan* policy as $\hat{S}(\hat{J}_{t'})$ and $\hat{F}(\hat{J}_{t'})$, respectively. The *makespan* model will minimize the makespan, $\hat{C}(\hat{J}_{t'}) = \hat{F}(\hat{J}_{t'}) - \hat{S}(\hat{J}_{t'})$. However, the work remaining from previous periods for all servers, represented by ν, must be accounted for. If J' is the set of jobs in the previous period, then the left-over work is bounded by $\nu \leq \max_{m\in M; j\in J'}(d_{mj})$ because any server with residual work greater than the largest of all the processing times would have had a job reassigned to the free machine in the previous sub-problem to reduce the sub-problem makespan. In the worst case, all servers except one are busy for time ν and the next period will have only one server available immediately, delaying the optimal schedule by less than ν time units. We further define the minimum makespan of scheduling $\hat{J}_{t'}$ when residual work can be ignored as $C^*(\hat{J}_{t'})$. It is easy to show that $\hat{C}(\hat{J}_{t'}) \leq C^*(\hat{J}_{t'}) + \nu$. There exists some number of jobs $|\bar{J}|$ where

$$1 \geq \frac{C^*(\bar{J})}{\hat{C}(\bar{J})} \geq \frac{\hat{C}(\bar{J}) - \nu}{\hat{C}(\bar{J})} = 1 - \omega$$

and as $|\bar{J}| \to \infty, \omega \to 0$. This is true because as the number of jobs increases, ν becomes negligible compared to the actual makespan of the schedule. Therefore, as the number of jobs in the system increases without bound, the makespan converges to the optimal. Since throughput of the system is the exit rate of jobs and *Round Robin* can at best match the minimum makespan, after the system has reached some sufficiently large size of $\bar{J}$, the throughput of *makespan* is at least as large as that of the *Round Robin* policy. Therefore, we find a contradiction: *makespan* cannot be unstable when *Round Robin* is stable if the throughput of *makespan* is at least as large as that of the *Round Robin* policy. Thus, it is guaranteed that any stabilizable system is stabilized under *makespan*. □

Note that the dependence on $|\bar{J}| \to \infty$ is an aspect of the proof technique, not a requirement for stability. Stability is a property of the system and does not depend on the number of jobs in the system. However, our proof depends on achieving the minimum makespan for each sub-problem, which requires solving an NP-hard problem. We return to this point in Section 5.

4.2.2 Stability of the Hybrid

We prove stability conditions for the hybrid model in the same way as for *makespan*. We first show that there exists a minimum makespan schedule that tracks the δ^*_{mk} values when the number of jobs in a period is sufficiently large.

Proposition 1. *As the number of jobs to be scheduled in a period approaches ∞, there exists a schedule that has a minimum makespan and tracks the δ^*_{mk} values from the allocation LP.*

Proof. Denote the set of jobs to schedule during a period as J, the set of jobs assigned to server m from class k as Q_{mk}, the resulting makespan of server m as C^*_m, and the makespan of the system over all machines for the set of jobs J as C^*_{max}. The time server m spends on jobs of class k is $\sum_{j \in Q_{mk}} d_{mj}$. The proportion of time each server m spends on a class k, denoted Δ_{mk}, is

$$\Delta_{mk} = \frac{\sum_{j \in Q_{mk}} d_{mj}}{\sum_{k \in K} \sum_{j \in Q_{mk}} d_{mj}} = \frac{\sum_{j \in Q_{mk}} d_{mj}}{C^*_m}.$$

Using the Law of Large Numbers, we know

$$\lim_{|Q_{mk}| \to \infty} \frac{1}{|Q_{mk}|} (\sum_{i \in Q_{mk}} d_{mj}) = \mu^{-1}_{mk}.$$

Taking the limit as the number of jobs in the system, $|J|$, goes to ∞, gives

$$\lim_{|J| \to \infty} \Delta_{mk} = \begin{cases} \lim_{|J| \to \infty} \frac{|Q_{mk}|}{\mu_{mk} C^*_{max}} & \text{when } \lim_{|J| \to \infty} |Q_{mk}| = \infty, \\ 0 & \text{when } \lim_{|J| \to \infty} |Q_{mk}| < \infty. \end{cases} \tag{1}$$

If we multiply Δ_{mk} by μ_{mk} and sum over all machines, we get

$$\lim_{|J| \to \infty} \sum_{m=1}^{|M|} \Delta_{mk} \mu_{mk} = \lim_{|J| \to \infty} \frac{1}{C^*_{max}} \sum_{m=1}^{|M|} |Q_{mk}| = \lim_{|J| \to \infty} \frac{p_k |J|}{C^*_{max}},$$

which is the left-hand side of constraint (1) in the allocation LP. Re-arranging terms leads to

$$\lim_{|J| \to \infty} \sum_{m=1}^{|M|} \frac{\Delta_{mk} \mu_{mk}}{p_k} = \lim_{|J| \to \infty} \frac{p_k |J|}{p_k C^*_{max}} = \lim_{|J| \to \infty} \frac{|J|}{C^*_{max}} \geq \lambda.$$

Here, we see that since the allocation LP aims to maximize λ, it is equivalent to minimizing the makespan C^*_{max}, and therefore a schedule exists with a minimum makespan that also tracks δ^*_{mk}. □

Theorem 4. *If* Round Robin *is stable for a given arrival process with rate λ, then the hybrid model is also stable.*

Proof. The proof is based on the proof of Theorem 3. *makespan* can be replaced by the hybrid model (denoted as h when used as a superscript) and the proof follows except for $C^h(J_t^h) - \nu \not\leq C^*(J_t^h)$. Due to the bi-criteria objective, the model does not guarantee minimum makespan schedules.

If we take $|J| \to \infty$, Proposition 1 states that there is a minimum makespan schedule that will track the δ^*_{mk} values such that c_{mk} goes to 0. Therefore, for a sufficiently large system size, c_{mk} will be small enough to ensure that the makespan of the schedule converges to that of *makespan* and, similarly to Theorem 3, the hybrid model guarantees stability in a stabilizable system. □

4.2.3 Instability of *MinFT*

To show that the *MinFT* model cannot handle all $\lambda < \lambda^*$, we provide a counter-example. Assume a system where there are two servers and two job classes. The arrival rate to the system is $\lambda = 1$ and $p_1 = p_2 = 0.5$. If $\mu_{11} = 10, \mu_{21} = 9, \mu_{12} = 9$, and $\mu_{22} = 10$, then the allocation LP would assign $\delta^*_{11} = 1, \delta^*_{21} = 0, \delta^*_{12} = 0$, and $\delta^*_{22} = 1$ to get $\lambda^* = 20$. In order for the system to be stable for a particular λ, a scheduling algorithm must adequately track the δ^*_{mk} values. As $\lambda \to \lambda^*$, the available freedom for an algorithm to deviate from δ^*_{mk} goes to 0.

Assignments in the *MinFT* model are made greedily to the server that will minimize the job's flow time. Denoting the completion time of the latest job on a server m as Ψ_m, we know that an arriving job j will be served on the faster server m rather than the slower server l unless the inequality $\Psi_m + d_{mj} > \Psi_l + d_{lj}$ is true. When the inequality is true, the *MinFT* model will assign the arriving job to server l since its completion time will be earlier. We define $\omega_j = d_{lj} - d_{mj}$ where $\omega_j << d_{mj}$ since the difference between the processing rates is an order of magnitude smaller than the processing rates themselves.

Consider two cases: (1) $|\Psi_1 - \Psi_2| > \omega_j$, and (2) $|\Psi_1 - \Psi_2| < \omega_j$. In case (1), an arriving job will be assigned to the server with a smaller Ψ_m regardless of class because the other server is busy. The *MinFT* model follows an efficient assignment to use the faster server only in case (2). In case (1) the scheduler will make the efficient assignment 50% of the time because an arriving job has equal probability of belonging to either class. We need to show that if $P(|\Psi_1 - \Psi_2| > \omega_j) > 0$, then the *MinFT* model cannot guarantee stability like the *Round Robin* policy. Assume the inequality is false, i.e., $P(|\Psi_1 - \Psi_2| > \omega_j) = 0$. In this scenario, we know the system at any point in time adheres to $|\Psi_1 - \Psi_2| < \omega_j$. However, if a job arrives, it must be scheduled onto one of the servers, making $|\Psi_1 - \Psi_2| > \min_{m \in 1,2}(d_{mj}) - \omega_j > \omega_j$. This inequality results in a contradiction since, with certainty, the next arriving job will force the system into case (1). Since case (1) occurs with positive probability, an inefficient assignment will occur with some non-zero probability. Denote the probability of inefficient assignments as $b_1 = P(\Psi_1 - \Psi_2 > \omega_j)p_1$ and $b_2 = P(\Psi_1 - \Psi_2 < -\omega_j)p_2$ where $b = b_1 = b_2 \geq (0.5)(0.5)$ because of system symmetry. Then the realized proportions of time each server spends on classes are $\delta_{11} = 1 - b, \delta_{21} = b, \delta_{12} = b$, and $\delta_{22} = 1 - b$. Therefore, the obtainable capacity is

$$10(1 - b) + 9b + 9b + 10(1 - b) = 20 - 2b < 20,$$

because $b > 0$. For this simple system, if $\lambda^* - 2b < \lambda < \lambda^*$, the *MinFT* model cannot guarantee stability.

4.3 Numerical Results

We experimentally compare the mean flow time performance of our proposed models. Two different cases are tested: two job classes and two servers, and four job classes and four servers. For each test case, five different loads between 0.8 and 0.99 of the maximum theoretical load are simulated. At each load, 20 instances are run for 10,000 time units, resulting in a total of 100 simulations per model for each of the two systems.

Job arrivals follow a Poisson process. In order to maintain the relative processing times of a single job on each server, an amount of work for each job, w_j, is generated using an exponential distribution with rate 1; it is then augmented linearly by the processing rate for a server, based on the job's class, to create the processing time $d_{mj} = \frac{w_j}{\mu_{mk}}$. We use processing rates μ_{mk} ranging from one job per time unit to ten jobs per time unit. Both test cases are asymmetric: the processing rates of the classes are different for different servers. Symmetric systems are not examined to emphasize the heterogeneity of the network.

The simulation was implemented in C++. The LP and MIP models use IBM ILOG CPLEX 12.1. All experiments are performed on a Dual Core AMD 270 CPU with 1 MB cache, 4GB of main memory, running Red Hat Enterprise Linux 4. Preliminary results showed that $\alpha = 0.6$ provided the best performance for the hybrid method (Tran, 2011).

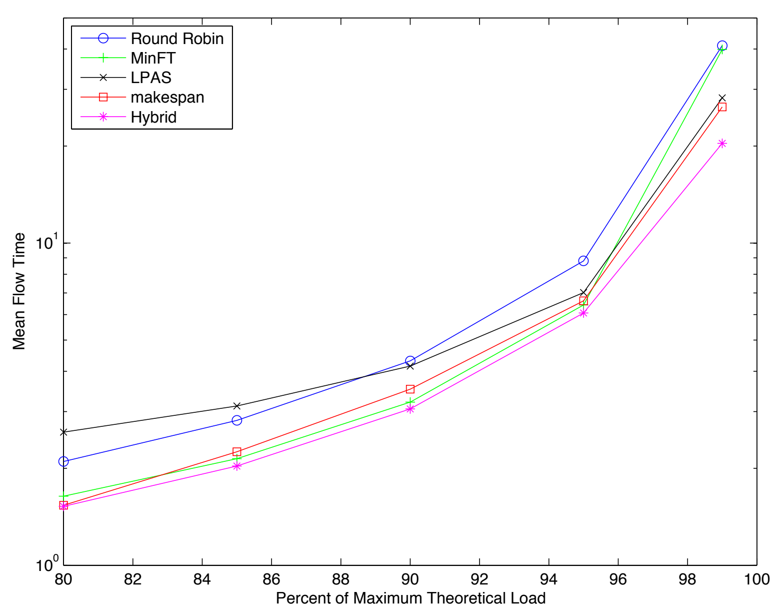

Figure 11: Comparison of mean flow times for a flexible queueing network with two servers and two classes.

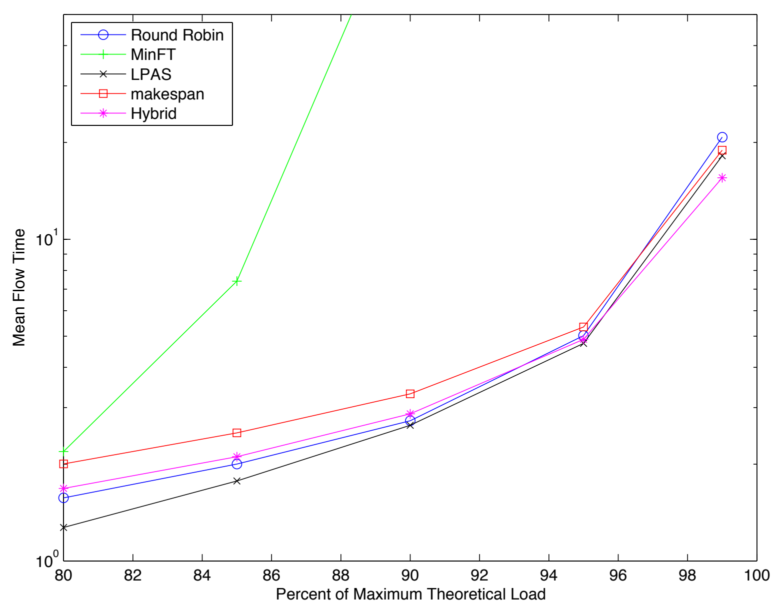

Figure 12: Comparison of mean flow times for a flexible queueing network with four servers and four classes.

Figures 11 and 12 present the mean flow times averaged across all problem instances for every load. Note the log-scale on the y-axes. In Figure 11, we see that the two scheduling models and the hybrid model create better schedules than *Round Robin*.

Increasing the system size produces substantially different results. Figure 12 shows that at lower loads, *LPAS* obtains the lowest mean flow time. This is in contrast to performing worse than *makespan* and the hybrid model at all loads on the smaller system. As the load increases, the *makespan* model performance becomes better than that of *Round Robin* but still does not become as good as that of *LPAS*. The hybrid model is able to obtain performance comparable with *LPAS* at lower loads and provide the best performance at very high loads. Thus, the hybrid model maintains robust performance across varying system loads. Even though the performance of the hybrid falls short of *LPAS* at lower loads, waiting times at low loads are almost negligible. At heavy loads where λ approaches λ^*, the hybrid model outperforms all other algorithms by about 17%.

The good performance of the *MinFT* model is lost in the larger system. At loads of 0.9 and greater, the *MinFT* model is not able to process jobs quickly enough to dissipate a build up of jobs. This is empirical confirmation that *MinFT* is not stable under parameters where other algorithms are stable, e.g., loads greater than 90% in Figure 12.

To investigate whether the strong performance of the hybrid model is indeed due to the guidance of the allocation LP values, we alter the δ^*_{mk} values and plot the performance in Figure 13. If following the queueing guidance is beneficial, deviating from these values should lead to worse schedules. Therefore, we replace all δ^*_{mk} values with $1 - \delta^*_{mk}$. The

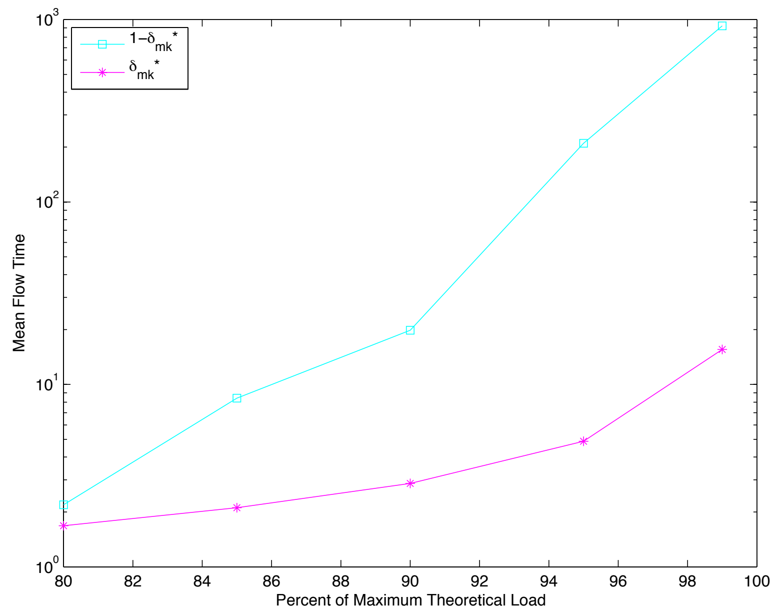

Figure 13: Comparison of mean flow times for the hybrid model guided by δ^*_{mk} and $1-\delta^*_{mk}$ for a flexible queueing network with four servers and four classes.

more resources the allocation LP assigns to a class, the fewer will be assigned in this hybrid model. Although this method allows $\sum_{k\in K}\delta^*_{mk} > 1$, the conceptual goal of scheduling to avoid the allocation LP solution is still maintained, and the validity of the model is not compromised since the resource capacity limit is essential only when solving the fluid model. We find that the incorrectly guided hybrid performs poorly at all loads; at loads above 0.9, the system size grows rapidly and behaves as if it were unstable. We conclude that using proper guidance from the allocation LP is crucial.

4.3.1 Performance Analysis

Our numerical results show that the *makespan* model can be improved by incorporating queueing analysis. We can understand this result at low loads by considering a system with two servers and two classes. With equal arrival rates and processing rates $\mu_{11} = 9, \mu_{12} = 2, \mu_{21} = 5$, and $\mu_{22} = 1$, solving the allocation LP gives $\delta^*_{11} = 0, \delta^*_{12} = 1, \delta^*_{21} = 0.5$, and $\delta^*_{22} = 0.5$. If the system is lightly loaded with only two class 1 jobs present, both having processing times equal to the expected value ($\frac{1}{\mu_{m1}}$: $\frac{1}{9}$ for server 1 and $\frac{1}{5}$ for server 2), then *makespan* will schedule one job on each of the servers. The hybrid model would consider the fact that $\delta^*_{11} = 0$ and depending on the α parameter chosen, could decide to place both jobs on the second server in an attempt to perform well in the long run. Although the hybrid is only partially guided by δ^*_{mk}, this guidance is sufficient for performance improvement over *makespan*.

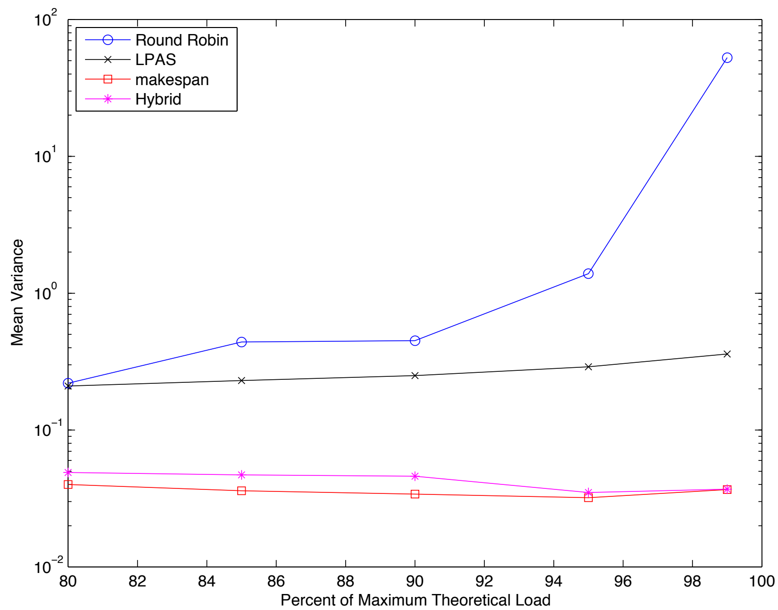

Figure 14: Comparison of class flow time variance for a flexible queueing network with four servers and four classes.

At heavy loads, Proposition 1 states that *makespan* will track δ^*_{mk} and so, contrary to our results, we would expect the *makespan* and hybrid models to perform similarly. However, from the time a server becomes busy until it is once again idle, the first few scheduling periods are expected to have fewer jobs than those in the middle of this time span, because there have not yet been enough arrivals for significant queues to build up. Therefore, the first few periods resemble a lightly loaded system. It is not necessarily the case that *makespan* will track δ^*_{mk} until a larger queue is formed. These early decisions will propagate through to the subsequent periods and affect all later jobs within the busy period. The hybrid model, from the start, is able to make better long-run decisions. Although the hybrid model does not guarantee that it can always make the best choice, we speculate that it does increase the probability that these better choices are made.

In the instances with four servers and four job classes, *LPAS* is better than the hybrid at low loads and worse at high loads. Deeper analysis shows that this performance pattern is largely due to the poor performance of the hybrid policy at low loads. At such loads, the hybrid will make assignments that are inefficient in the long term as there are few jobs in the system. We argued above that the hybrid model is making better assignments than *makespan* because it is guided by the queueing analysis. However, a policy that maintains a more strict adherence to the δ^*_{mk} values, such as *LPAS* and *Round Robin*, reserves servers for job classes that are efficient in the long term. Even though the hybrid model incorporates the δ^*_{mk} values in the decision-making process, it is not guaranteed at low loads that the model will always make globally optimal decisions that adhere to δ^*_{mk}. This is especially true when α is chosen to be high.

The hybrid model outperforms *LPAS* at heavy loads because optimizing the makespan for each period has long-run impact. At heavy loads the hybrid model is able to both match the δ^*_{mk} values and minimize makespan, whereas *Round Robin* can only do the former and *LPAS* cannot guarantee either. If the hybrid model reduces the makespan by t time units compared to *LPAS* or *Round Robin*, all jobs in the next period will be able to start t time units earlier and thereby have a net effect of reducing the mean flow time by approximately t for the subsequent set. The reductions will propagate to subsequent periods until a server is once again idle.

The propagation effect from reducing the makespan of an earlier period is a property of the hybrid model that is inherited from the *makespan* model. As such, we can expect *makespan* to exhibit this behaviour as well. Our numerical results show, as expected, the performance of *makespan* is good at heavy loads. The *makespan* model only lacks the queueing guidance to ensure even higher quality schedules earlier in a busy period. We expected similar results from the *makespan* model in the dynamic flowshop problem of Section 3 as well. However, the experimental results from Section 3 show our assumption to be incorrect. We conjecture that one of the reasons why we see better performance with *makespan* in the queueing network with flexible servers and not in the dynamic flowshop is because in the latter the differences between optimal sub-problem makespans and the makespans found by other methods are not substantial. In the queueing network with flexible servers, the makespan may change more significantly when using different policies. Given the complex nature of these systems, further investigation into performance trade-offs resulting from different system parameters is necessary in future work. We hope that such an investigation will lead to derivations of general dynamic scheduling principles.

4.3.2 Beyond Mean Flow Time

Though our primary quantity of interest is the mean flow time, variance is also an important criterion. From the perspective of a customer who has submitted a job for processing, a high variance indicates that the actual flow time of a given job is unlikely to be accurately predicted by the mean flow time.

Figure 14 shows that the variance in the mean flow time of each job class for four servers and four classes when using *Round Robin* or *LPAS* is much larger than that of the *makespan* and hybrid policies.

We see a substantial difference (note the log scale) among the four algorithms. The larger variance observed in the queueing models occurs because the policies use less information about the state of the system. The policies may overcompensate to serve a job class immediately rather than delaying service to achieve a fairer allocation. In contrast, the scheduling models make use of state information to perform better load balancing and therefore exhibit lower variance.

5. Discussion

Our motivation for studying the integration of scheduling and queueing was that as the two areas address similar problems in different ways, their combination in terms of problem settings, performance measures and algorithms provides a richer set of domains, goals and tools. Given the nascent nature of such a study, we looked at two simple problem settings and demonstrated the following contributions.

1. In both problem settings, we showed that it is possible to establish stability of periodic scheduling approaches, enhancing the periodic scheduling framework with a guarantee of stability that has traditionally been available for queueing approaches only. As far as we are aware, this is the first time that stability has been established for algorithms that use observed job characteristics instead of, or in addition to, stochastic information. Similarly, as far as we are aware, the concept of stability as it appears in queueing theory has not been examined by the combinatorial scheduling community. We believe this is an oversight arising from the area's focus on short-term combinatorics and suggest that, as in queueing theory, stability is an important criterion for any problem that must deal with a stream of arriving resource requests.

2. In our second problem setting, a flexible queueing network, but not in our first, a dynamic flowshop, we demonstrated that solution approaches that combine guidance from long-run stochastic reasoning with short-term combinatorial reasoning can perform better than queueing or scheduling approaches alone. However, the differences between our two problem settings, as well as other work in more complex problem domains (Tran, 2011), shows that such a combination is non-trivial and further work is needed to place such hybrid algorithms on a more formal foundation.

We see a variety of additional ways to integrate ideas from queueing theory and scheduling in the future. For example, we would like to: extend our analysis to more general dynamic scheduling environments, such as job shops; compare the methods discussed in this paper with more complex queueing approaches and alternative dynamic scheduling approaches, such as OSCO; investigate stability of dynamic scheduling approaches such as OSCO or those based on approximation algorithms (i.e., polynomial time approximation schemes applied to every sub-problem); and develop more sophisticated hybrid queueing theory and scheduling models.

A broader direction for future work is to more fully investigate the fact that, for a number of applications, it is now reasonable to assume that both data about probability distributions and the actual job characteristics (at arrival time) are available. The best way to integrate these different data sources is an open question that this paper has begun to investigate from the perspective of hybridization of existing tools.

5.1 Distributional Assumptions

Our work assumes that, for each queue, the sequences of processing times for all machines and the sequence of inter-arrival times are independent and identically distributed sequences, and that these sequences are also mutually independent. In many AI applications, such assumptions may not be justified, as the data may be non-stationary (time-dependent) and/or correlated. Both non-stationary models and simple correlation structures are also considered in the queueing theory literature (Prabhu & Zhu, 1989; Massey, 2002; Gupta, Harchol-Balter, Scheller-Wolf, & Yechiali, 2006; Liu & Whitt, 2012).

Importantly, we note that the proof of Lemma 1 holds in a non-stationary setting and that the scheduling methods used in the dynamic flow shop are distribution-independent. For the flexible queueing network, the majority of analysis is based on the allocation LP, which is based on the fluid representation of a system. For fluid analysis of stability, the i.i.d. assumption is not required; instead, it is necessary for the Law of Large Numbers to hold for the inter-arrival and processing time sequences. Thus, our analysis can be extended to including correlation structures as long as the Law of Large Numbers holds. On the algorithmic side, the *Round Robin*, *LPAS* and hybrid methods can be adapted to a non-stationary setting by periodically re-solving the allocation LP and using the updated δ^* values. In fact, this approach is a special case of a general principle that needs to be considered in dynamic scheduling problems: the distributions may need to be updated as new data becomes available. Not surprisingly, in queueing theory as in scheduling, non-stationary situations require periodic approaches.

5.2 Computational Complexity

A substantial difference between periodic scheduling approaches and most traditional queueing theory-based policies is that the scheduling approaches may have to solve NP-hard problems. This is indeed the case for the *completionTime* algorithm for the two-machine dynamic flowshop problem (Section 3) and both the *makespan* and hybrid algorithms for the queueing network (Section 4). Our assumption in this work, consistent with that in related work such as OSCO (Van Hentenryck & Bent, 2006; Mercier & Van Hentenryck, 2007), is that dynamism of the system is slow enough to allow such problem solving. We do not report run-times for solving our problems because we can make them look arbitrarily good or bad by making assumptions about the time granularity. For example, if a time unit in the problem corresponds to one hour, algorithm run-times of 10 or 20 seconds are unlikely to be significant.

In applications with a finer time granularity, we may not be able to solve these problems and traditional queueing policies are more appropriate. However, our work does raise the interesting question of the stability of polynomial-time algorithms that provide approximation guarantees. As our proofs of Theorems 2 and 4 depend on finding optimal makespans, there does not seem to be an easy generalization.

5.3 Online Scheduling

Online scheduling is an alternative approach to dynamic scheduling problems (Pruhs et al., 2004), different from both classical combinatorial scheduling and queueing theory. The online scheduling literature focuses on competitive analysis, proving worst-case bounds on how much worse deterministic and randomized online algorithms are, compared to a full-information algorithm. As with queueing theory, the rigorous mathematical approach of online scheduling tends to limit the combinatorial structure that is addressed. However, unlike queueing theory, it is uncommon to assume knowledge of stochastic distributions from which job arrivals and characteristics are drawn. Indeed, often the results showing differences between deterministic and randomized online algorithms arise from the analysis of systems where an adversary has full knowledge of the online algorithm and can manipulate job characteristics arbitrarily.

While we have chosen not to include online scheduling in this paper, we believe that it is important to understand how results and insights of this area can be integrated with our work in order to obtain an even deeper understanding of dynamic scheduling problems.

5.4 Relevance to AI

Dynamic scheduling requires a series of scheduling and resource allocation decisions to be made online before future tasks arrive and their characteristics, if any, are known. This is the challenge of sequential decision making under uncertainty, a problem that has received a significant amount of attention in AI. Indeed, the requirement for an agent to make decisions and take actions without full knowledge of the future states of the world would appear to be a central requirement for an embodied, intelligent agent. Investigation of sequential decision making in applications of interest to AI include planning and task allocation under uncertainty (Keller & Eyerich, 2012; Alighanbari & How, 2008), land purchases for the conservation of endangered species (Xue, Fern, & Sheldon, 2012), and multi-player and strategy games (Sturtevant, 2008; Balla & Fern, 2009).

Most methodological approaches to such problems in AI rely in some way on Markov Decision Processes (MDPs) (Puterman, 1994), notably in the area of decision-theoretic planning (Boutilier, Dearden, & Goldszmidt, 2000). The challenge arising from direct applications of MDPs is the well-known "curse of dimensionality", where the state space is so large that problems cannot be solved. Substantial work has therefore focused on approaches such as factored MDPs (Meuleau, Hauskrecht, Kim, Peshkin, Kaelbling, Dean, & Boutilier, 1998; Boutilier et al., 2000; Guestrin, Koller, & Parr, 2003), approximate dynamic programming (Powell, 2010), Monte Carlo Tree Search (Chaslot, 2011; Browne, Powley, Lucas, Cowling, Rohlfshagen, Tavener, Perez, Samothrakis, & Colton, 2012; Bellemare, Naddaf, Veness, & Bowling, 2013) and Online Stochastic Combinatorial Optimization (Van Hentenryck & Bent, 2006).

Queueing theory is another approach to sequential decision making under uncertainty, one that emphasizes time and resources and one that, to our knowledge, has not been considered for solving problems studied in AI. Given the concern with time and resources, dynamic scheduling is a natural problem choice for investigation of the incorporation of queueing theory into the toolbox of AI techniques. As the richness of decision-making problems in AI extends to questions of time and resources (e.g., in temporal planning problems, see Coles, Coles, Fox, & Long, 2012) and, in fact, much of the underlying analysis of prescriptive queueing theory approaches is founded on MDPs (Stidham & Weber, 1993; Sennott, 1999; Meyn, 2008), we believe that the application of queueing theory to AI and the hybridization of queueing and scheduling as proposed in this paper, are promising directions for fundamental and applied research.

6. Conclusion

In this paper, we considered the combination of long-run stochastic reasoning with short-term combinatorial reasoning to solve dynamic scheduling problems. More specifically, we investigated the integration of queueing theory and scheduling in two simple scheduling problems: a dynamic two-machine flowshop and a flexible queueing network. We provided both analytical and empirical results.

Analytically, we demonstrated for both of our problem settings that scheduling approaches that make use of the observed job characteristics, as opposed to stochastic information about job classes, can be proved to be stable. That is, if it is possible to manage the system such that the number of jobs waiting for processing remains finite, then a combinatorial scheduling algorithm can do so.

Empirically, we demonstrated that for a common scheduling criterion it is possible to create a hybrid algorithm guided by long-term stochastic reasoning and short-term combinatorial reasoning. Furthermore, such an algorithm can be shown to be stable and to empirically out-perform pure scheduling and pure queueing approaches. However, this result was shown for only one of our two problem settings, suggesting that we currently do not have an understanding that will allow us to systematically design such successful hybrids.

We believe that our novel investigation of the integration of problem settings, performance criteria, and algorithms of queueing theory and combinatorial scheduling opens a number of interesting research directions surrounding approaches to dynamic scheduling and, more broadly, to sequential decision making under uncertainty.

Acknowledgements

The authors would like to thank the reviewers and the associate editor for their comments, which helped improve the paper.

Section 3 of this paper is based on previously published workshop and conference papers (Terekhov, Tran, & Beck, 2010; Terekhov, Tran, Down, & Beck, 2012). The work in Section 4 forms part of a Master's dissertation (Tran, 2011) but has not appeared in any peer-reviewed publication.

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

References

Al-Azzoni, I., & Down, D. G. (2008). Linear programming-based affinity scheduling of independent tasks on heterogeneous computing systems. *Parallel and Distributed Systems, IEEE Transactions on*, *19*(12), 1671–1682.

Alighanbari, M., & How, J. P. (2008). A robust approach to the UAV task assignment problem. *International Journal of Robust and Nonlinear Control*, *18*, 118–134.

Andradóttir, S., & Ayhan, H. (2005). Throughput maximization for tandem lines with two stations and flexible servers. *Operations Research*, *53*(3), 516–531.

Andradóttir, S., Ayhan, H., & Down, D. G. (2003). Dynamic server allocation for queueing networks with flexible servers. *Operations Research*, *51*(6), 952–968.

Balla, R.-K., & Fern, A. (2009). UCT for tactical assault planning in real-time strategy games. In *Proceedings of the 21st International Joint Conference on Artificial Intelligence (IJCAI'09)*, pp. 40–45.

Baptiste, P., Le Pape, C., & Nuijten, W. (2001). *Constraint-based Scheduling*. Kluwer Academic Publishers.

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

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

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

Bellemare, M. G., Naddaf, Y., Veness, J., & Bowling, M. (2013). The arcade learning environment: An evaluation platform for general agents. *Journal of Artificial Intelligence Research*, *47*, 253–279.

Bent, R., & Van Hentenryck, P. (2007). Waiting and relocation strategies in online stochastic vehicle routing.. In *Proceedings of the 20th International Joint Conference on Artificial Intelligence (IJCAI'07)*, pp. 1816–1821.

Bidot, J., Vidal, T., Laborie, P., & Beck, J. C. (2009). A theoretic and practical framework for scheduling in a stochastic environment. *Journal of Scheduling*, *12*(3), 315–344.

Bolch, G., Greiner, S., de Meer, H., & Trivedi, K. S. (2006). *Queueing networks and Markov chains: modeling and performance evaluation with computer science applications*. Wiley-Interscience.

Boutilier, C., Dearden, R., & Goldszmidt, M. (2000). Stochastic dynamic programming with factored representations. *Artificial Intelligence*, *121*, 49–107.

Bowman, E. (1959). The schedule-sequencing problem. *Operations Research*, *7*(5), 621–624.

Bramson, M. (2008). Stability of queueing networks. *Probability Surveys*, *5*, 169–345.

Bramson, M., & Williams, R. J. (2000). On dynamic scheduling of stochastic networks in heavy traffic and some new results for the workload process. In *Proceedings of the 39th IEEE Conference on Decision and Control*, Vol. 1, pp. 516–521.

Branke, J., & Mattfeld, D. C. (2002). Anticipatory scheduling for dynamic job shop problems. In *Proceedings of the ICAPS'02 Workshop on On-line Planning and Scheduling*, pp. 3–10.

Branke, J., & Mattfeld, D. C. (2005). Anticipation and flexibility in dynamic scheduling. *International Journal of Production Research*, *43*(15), 3103–3129.

Browne, C., Powley, E., Lucas, S., Cowling, P. I., Rohlfshagen, P., Tavener, S., Perez, D., Samothrakis, S., & Colton, S. (2012). A survey of Monte Carlo tree search methods. *IEEE Transactions on Computational Intelligence and AI in Games*, *4*(1), 1–49.

Brucker, P., Jurisch, B., & Sievers, B. (1994). A branch and bound algorithm for the job-shop scheduling problem. *Discrete Applied Mathematics*, *49*(1), 107–127.

Burke, P. (1956). The output of a queuing system. *Operations Research*, *4*(6), 699–704.

Burns, E., Benton, J., Ruml, W., Yoon, S., & Do, M. B. (2012). Anticipatory on-line planning. In *Proceedings of the Twenty-Second International Conference on Automated Planning and Scheduling (ICAPS'12)*, pp. 333–337.

Carlier, J., & Pinson, E. (1989). An algorithm for solving the job-shop problem. *Management Science*, *35*(2), 164–176.

Chaslot, G. M. J.-B. (2011). *Monte-Carlo Tree Search*. Ph.D. thesis, Universeteit Maastricht.

Chen, H., & Yao, D. D. (1992). A fluid model for systems with random disruptions. *Operations Research*, *41*(2), 239–247.

Coles, A. J., Coles, A. I., Fox, M., & Long, D. (2012). COLIN: Planning with continuous linear numeric change. *Journal of Artificial Intelligence Research*, *44*, 1–96.

Conway, R. W., Maxwell, W. L., & Miller, L. W. (1967). *Theory of Scheduling*. Addison-Wesley.

Crabill, T., Gross, D., & Magazine, M. (1977). A classified bibliography of research on optimal design and control of queues. *Operations Research*, *25*(2), 219–232.

Dai, J. G. (1995). On positive Harris recurrence of multiclass queueing networks: A unified approach via fluid limit models. *The Annals of Applied Probability*, *5*(1), 49–77.

Dai, J. G., & Meyn, S. P. (1995). Stability and convergence of moments for multiclass queueing networks via fluid limit models. *IEEE Transactions on Automatic Control*, *40*(11), 1889–1904.

Dai, J. G., & Weiss, G. (1996). Stability and instability of fluid models for reentrant lines. *Mathematics of Operations Research*, *21*(1), 115–134.

Daniels, R., & Carrillo, J. (1997). β-robust scheduling for single-machine systems with uncertain processing times. *IIE Transactions*, *29*, 977–985.

Davenport, A., Gefflot, C., & Beck, J. C. (2001). Slack-based techniques for robust schedules. In *Proceedings of the Sixth European Conference on Planning (ECP-2001)*.

Down, D., & Meyn, S. (1994). A survey of Markovian methods for stability of networks. In *11th International Conference on Analysis and Optimization of Systems: Discrete Event Systems*, pp. 490–504. Springer.

Down, D. G., & Karakostas, G. (2008). Maximizing throughput in queueing networks with limited flexibility. *European Journal of Operational Research*, *187*(1), 98–112.

El-Bouri, A., Balakrishnan, S., & Popplewell, N. (2008). Cooperative dispatching for minimizing mean flowtime in a dynamic flowshop. *International Journal of Production Economics*, *113*(2), 819–833.

Fox, M. S. (1987). *Constraint-directed Search: A Case Study of Job-Shop Scheduling*. Morgan-Kaufmann Publishers Inc.

French, S. (1982). *Sequencing and Scheduling: An Introduction to the Mathematics of the Job-shop*. Ellis Horwood.

Gross, D., & Harris, C. (1998). *Fundamentals of Queueing Theory*. John Wiley & Sons, Inc.

Guestrin, C., Koller, D., & Parr, R. (2003). Efficient solution algorithms for factored MDPs. *Journal of Artificial Intelligence Research*, *19*, 399–468.

Gupta, V., Harchol-Balter, M., Scheller-Wolf, A., & Yechiali, U. (2006). Fundamental characteristics of queues with fluctuating load. *ACM SIGMETRICS Performance Evaluation Review*, *34*(1), 203–215.

Herroelen, W., & Leus, R. (2005). Project scheduling under uncertainty: Survey and research potentials. *European Journal of Operational Research*, *165*(2), 289–306.

Jackson, J. (1957). Networks of waiting lines. *Operations Research*, *5*(4), 518–521.

Keller, T., & Eyerich, P. (2012). PROST: Probabilistic planning based on UCT. In *Proceedings of the 22nd International Conference on Automated Planning and Scheduling (ICAPS'12)*, pp. 119–127.

Kendall, D. G. (1953). 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.

Kovács, A., & Beck, J. C. (2011). A global constraint for total weighted completion time for unary resources. *Constraints*, *16*(1), 100–123.

Kumar, P. R. (1994). Scheduling semiconductor manufacturing plants. *IEEE Control Systems Magazine*, *14*(6), 33–40.

Leon, V. J., Wu, S. D., & Storer, R. H. (1994). Robustness measures and robust scheduling for job shop. *IIE Transactions*, *26*(5), 32–43.

Liu, Y., & Whitt, W. (2012). The $G_t/GI/s_t + GI$ many-server fluid queue. *Queueing Systems*, *71*(4), 405–444.

Manne, A. (1960). On the job-shop scheduling problem. *Operations Research*, *8*(2), 219–223.

Massey, W. A. (2002). The analysis of queues with time-varying rates for telecommunication models. *Telecommunication Systems*, *21*(2–4), 173–204.

Mercier, L., & Van Hentenryck, P. (2007). Performance analysis of online anticipatory algorithms for large multistage stochastic integer programs. In *Proceedings of the 20th International Joint Conference on Artificial Intelligence*, pp. 1979–1984. Morgan Kaufmann Publishers Inc.

Meuleau, N., Hauskrecht, M., Kim, K. E., Peshkin, L., Kaelbling, L. P., Dean, T., & Boutilier, C. (1998). Solving very large weakly coupled Markov decision processes. In *Proceedings of the 15th National Conference on Artificial Intelligence (AAAI'98)*.

Meyn, S. P. (2008). *Control Techniques for Complex Networks*. Cambridge University Press.

Nowicki, E., & Smutnicki, C. (1996). A fast taboo search algorithm for the job shop problem. *Management Science*, *42*(6), 797–813.

Nowicki, E., & Smutnicki, C. (2005). An advanced tabu algorithm for the job shop problem. *Journal of Scheduling*, *8*, 145–159.

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

Park, B. Y. (1988). An evaluation of static flowshop scheduling heuristics in dynamic flowshop models via a computer simulation. *Computers & Industrial Engineering*, *14*(2), 103–112.

Petrick, R. P. A., & Foster, M. E. (2013). Planning for social interaction in a robot bartender domain. In *Proceedings of the 23rd International Conference on Automated Planning and Scheduling*, pp. 389–397.

Pinedo, M. L. (2003). *Scheduling: Theory, Algorithms, and Systems* (2nd edition). Prentice-Hall.

Powell, W. (2010). Merging AI and OR to solve high-dimensional stochastic optimization problems using approximate dynamic programming. *INFORMS Journal on Computing*, *22*(1), 2–17.

Prabhu, N. U., & Zhu, Y. (1989). Markov-modulated queueing systems. *Queueing Systems*, *5*(1–3), 215–245.

Pruhs, K., Sgall, J., & Torng, E. (2004). Online scheduling. In Leung, J. Y.-T. (Ed.), *Handbook of Scheduling: Algorithms, Models and Performance Analysis*, chap. 15. CRC Press.

Puterman, M. L. (1994). *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. John Wiley & Sons, Inc.

Reich, E. (1957). Waiting times when queues are in tandem. *The Annals of Mathematical Statistics*, *28*(3), 768–773.

Ross, S. M. (2003). *Introduction to Probability Models*, chap. 6 – Continuous-Time Markov Chains, pp. 349–399. Academic Press.

Sarper, H., & Henry, M. C. (1996). Combinatorial evaluation of six dispatching rules in a dynamic two-machine flow shop. *Omega*, *24*(1), 73–81.

Sennott, L. I. (1999). *Stochastic Dynamic Programming and the Control of Queueing Systems*. John Wiley & Sons, Inc.

Sotskov, Y. N., Sotskova, N. Y., Lai, T.-C., & Werner, F. (2010). *Scheduling Under Uncertainty: Theory and Algorithms*. Belorussian Science.

Stidham, Jr., S., & Weber, R. (1993). A survey of Markov decision models for control of networks of queues. *Queueing Systems*, *13*, 291–314.

Sturtevant, N. (2008). An analysis of UCT in multi-player games. In *Proceedings of the Sixth International Conference on Computers and Games (CG2008)*, pp. 37–49.

Tadj, L., & Choudhury, G. (2005). Optimal design and control of queues. *Sociedad de Estadìstica e Investigación Operativa, Top*, *13*(2), 359–412.

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

Terekhov, D. (2013). *Integrating Combinatorial Scheduling with Inventory Management and Queueing Theory*. Ph.D. thesis, Department of Mechanical and Industrial Engineering, University of Toronto.

Terekhov, D., Beck, J. C., & Brown, K. N. (2009). A constraint programming approach for solving a queueing design and control problem. *INFORMS Journal on Computing*, *21*(4), 549–561.

Terekhov, D., Tran, T. T., & Beck, J. C. (2010). Investigating two-machine dynamic flow shops based on queueing and scheduling. In *Proceedings of ICAPS'10 Workshop on Planning and Scheduling Under Uncertainty*.

Terekhov, D., Tran, T. T., Down, D. G., & Beck, J. C. (2012). Long-run stability in dynamic scheduling. In *Proceedings of the 22nd International Conference on Automated Planning and Scheduling (ICAPS'12)*, pp. 261–269.

Towsley, D., & Baccelli, F. (1991). Comparisons of service disciplines in a tandem queueing network with real time constraints. *Operations Research Letters*, *10*(1), 49–55.

Tran, T. T. (2011). Using queueing analysis to guide combinatorial scheduling in dynamic environments. Master's thesis, Department of Mechanical and Industrial Engineering, University of Toronto.

Van Hentenryck, P., & Bent, R. (2006). *Online Stochastic Combinatorial Optimization*. MIT Press.

Widmer, M., & Hertz, A. (1989). A new heuristic method for the flow shop sequencing problem. *European Journal of Operational Research*, *41*(2), 186–193.

Wierman, A., Winands, E., & Boxma, O. (2007). Scheduling in polling systems. *Performance Evaluation*, *64*, 1009–1028.

Wu, C. W., Brown, K. N., & Beck, J. C. (2009). Scheduling with uncertain durations: Modeling β-robust scheduling with constraints. *Computers & Operations Research*, *36*(8), 2348–2356.

Xia, C. H., Shanthikumar, J. G., & Glynn, P. W. (2000). On the asymptotic optimality of the SPT rule for the flow shop average completion time problem. *Operations Research*, *48*(4), 615–622.

Xue, S., Fern, A., & Sheldon, D. (2012). Scheduling conservation designs via network cascade optimization. In *Proceedings of the 26th AAAI Conference on Artificial Intelligence (AAAI'12)*, pp. 391–397.

Yoon, S. W., Fern, A., Givan, R., & Kambhampati, S. (2008). Probabilistic planning via determinization in hindsight. In *Proceedings of the 23rd AAAI Conference on Artificial Intelligence (AAAI'08)*, pp. 1010–1016.