# Investigating Two-Machine Dynamic Flow Shops Based on Queueing and Scheduling

**Daria Terekhov** and **Tony T. Tran** and **J. Christopher Beck**
Department of Mechanical & Industrial Engineering
University of Toronto, Toronto, Ontario, Canada
{dterekho,tran,jcb}@mie.utoronto.ca

## Abstract

We study two dynamic environments: a two-machine flow shop, which has received significant attention in scheduling research, and a polling system with a flow shop server, an extension of systems typically studied in queueing theory. Based on analysis and experiments, and using ideas from both queueing theory and scheduling, we obtain new insights regarding the solution of dynamic scheduling problems by a periodic scheduling approach. Specifically, we focus on the tradeoff between short-run and long-run objectives, and show that in the polling system, minimizing *makespan* at each scheduling point leads to better mean *flow time* over a long horizon than does minimizing flow time itself or using simple queueing policies. The opposite is true in a dynamic flow shop without a polling structure. We provide an explanation for these results based on a mathematical model of the behaviour of scheduling algorithms in dynamic flow shops. Our work demonstrates that integrating ideas from queueing theory and scheduling can lead to a better understanding of dynamic scheduling.

## Introduction

In a typical real-world scheduling problem, the set of jobs changes dynamically over time and the processing times of jobs are affected by various types of uncertainty. The goal is to determine how the available machine processing time is to be allocated among competing requests with the objective of optimizing the performance of the system. Methods for solving dynamic scheduling problems, in general, must address the combinatorial structure inherent in most interesting scheduling problems, the uncertainty about the evolution of the system, and the effect of current decisions on the future.

Scheduling research has mostly focused on devising effective methods for solving deterministic problems with a complex combinatorial structure, although there has been an increasing interest in modelling and solving scheduling problems in dynamic and uncertain environments (Aytug et al. 2005; Bidot et al. 2009; Suresh and Chaudhuri 1993). In queueing theory, on the other hand, scheduling problems with a simpler combinatorial structure but with stochastic and dynamic characteristics have been studied for a long time (Conway, Maxwell, and Miller 1967).

In this paper, we analyze two dynamic flow shop environments: a novel polling system that is an extension of systems traditionally examined in queueing theory and a dynamic two-machine flow shop, which is important in scheduling research. In both cases, the objective is to minimize the mean flow time of jobs (i.e., the time between the arrival of a job to the system and its completion time). Our experiments demonstrate that in the polling system, a scheduling method that optimizes the makespan at each decision point provides the best performance. In a dynamic flow shop, an approach based on minimizing the mean flow time works best. We develop a mathematical model that represents these two environments and accounts for performance differences in terms of the specific structural properties of the scheduling problems. This analysis shows that short-run and long-run objectives affect these two related environments differently.

Our paper has two main contributions. Most importantly, we demonstrate that combining problem settings and ideas from queueing theory and scheduling can lead to novel insights about scheduling in dynamic environments. Specifically, we obtain a new understanding of the trade-off between short-run and long-run objectives in dynamic scheduling. Secondly, we provide a detailed empirical analysis of the performance of two scheduling methods for a problem from queueing theory as well as a problem from scheduling.

## Motivations

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

Queueing theory has taken the viewpoint that, since it

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

In the scheduling community, a dynamic scheduling problem is generally viewed as a collection of linked static problems. Adopting this view makes the abundance of algorithms developed for static scheduling problems directly applicable as these algorithms can deal with complex combinatorics and can optimize the quality of the schedules for each static sub-problem. However, these approaches tend to overlook the long-run performance and the stochastic properties of the system, with a notable exception being the work of Branke and Mattfeld (Branke and Mattfeld 2002).

Thus, queueing theory and scheduling have differing views of dynamic problems. Our goal is to investigate whether these differences can be leveraged to gain new insights about dynamic scheduling.

## Problem Formulations

We study two related dynamic scheduling environments: a two-machine flow shop and a polling system with a flow shop-like server.

In a two-machine dynamic flow shop, jobs arrive stochastically over time and must be processed first on machine 1 and then on machine 2. The time at which a job arrives and the processing times on both machines are not known until the instant the arrival occurs. Both machines are of unary capacity, and preemptions are not allowed. The buffers in front of machine 1 and machine 2 are both assumed to be of infinite size. The goal of the problem is to assign start times to all jobs on the two machines so that the mean job flow time is minimized. A job's flow time is defined as the difference between the job's completion time on the second machine and its arrival time to the system.

Polling systems are also dynamic environments: jobs arrive at random points in time and, under one set of queueing assumptions, the processing times of these jobs are not known until their arrival. Polling systems usually consist of a single server and multiple job classes. Each arriving job has to wait for processing in the queue corresponding to its specific class. The server is assumed to visit each of these queues in a particular order and serve a subset of the jobs in that queue. A variety of policies stating the order in which the queues should be visited and the number of jobs that should be served on each visit have been considered (Levy and Sidi 1990; Takagi 2000). We assume that the server visits the queues in a cyclic manner. Upon each visit to a queue, the server employs a gated discipline: it processes all jobs that are present at the time of its arrival. Preemptions are not allowed.

Unlike in standard queueing models, we assume that the server of the polling model consists of two machines in series (a two-machine flow shop). We consider the problem of scheduling jobs for processing by the two-machine server with the objective of optimizing the mean flow time. Due to the assumption of a gated policy, upon arrival to a queue, the server is faced with a static two-machine flow shop scheduling problem.

## Scheduling in Polling Systems and Dynamic Flow Shops

We solve the two problems described above via periodic scheduling strategies: at a given point in time, we review the jobs present in the system or a particular queue of the system, create a schedule for these jobs, and once this schedule is executed, review the status of the system again. We examine two queueing-based and two scheduling-based methods for creating each sub-schedule in both polling systems and dynamic flow shops. The time at which scheduling happens, however, is different in the two environments. In the polling system, the server switches from one queue to the next only after all of the jobs present upon its arrival to the queue have been processed on *both* machines. The start of every new sub-problem is equal to the completion time of the last job in the previous sub-problem, as shown in Figure 1. In a dynamic flow shop without a polling structure, such an assumption is unreasonable since it would create unnecessary idle time on machine 1. Thus, in a dynamic flow shop, scheduling is performed once all jobs of the previous sub-problem have been processed on the *first* machine, as illustrated in Figure 2. This difference plays a key role in further analysis.

### Methods for Solving Static Sub-problems

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

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

Employing $SPT_{sum}$ means that all jobs present in the queue at the time when the schedule is constructed 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

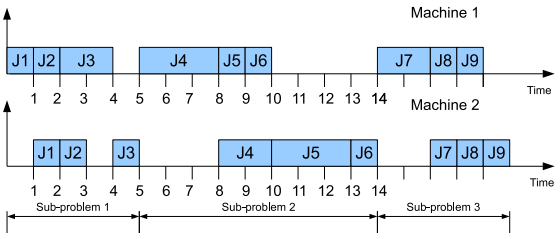


Figure 1: Polling system with 3 sub-problems (each corresponding to a queue visit) and $n = 3$ jobs per sub-problem.

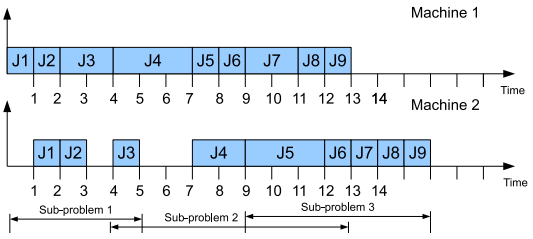


Figure 2: Dynamic flow shop with 3 sub-problems and $n = 3$ jobs per sub-problem.

expected flow time in queueing systems with a single queue or with a polling structure under a cyclic, gated service discipline (Wierman, Winands, and Boxma 2007). Wierman et al. show that such a policy can outperform *FCFS* by as much as 15% in a gated, cyclic polling system.

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

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

Below we present experiments comparing the performance of *FCFS*, $SPT_{sum}$, *makespan* and *completionTime* models in our two problems of interest. The *completionTime* model was implemented via constraint programming in Ilog Scheduler 6.5 and uses the *completion* global constraint (Kovács and Beck 2010). The remaining methods were implemented using C++. 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 for our experiments. With a time limit, this approach is not guaranteed to find the optimal sub-problem schedule. We do not take algorithm run-time into account in any of our results.

## Polling System

In order to develop an understanding of the performance of the four methods in a polling system with a flow shop server, we first simulated five symmetrical systems with $N = 5$ queues and with a fixed arrival rate of $\lambda = 1$ for each queue. In each system, the processing times are exponentially distributed with the same means on both machines. As the mean processing times increase in the different experimental conditions, the load on the system, defined as $N\rho$, where $\rho = \lambda/\mu$ and $\mu$ is the mean processing time both on machine 1 and 2, increases. Thus, in order to observe the variation in performance as the load changes, we considered systems with $\mu \in \{16, 12, 10, 8, 6\}$.

Figure 3 shows the mean flow times for the *completionTime* model with a 1-second time limit, *FCFS*, $SPT_{sum}$ and the *makespan* model as the system load increases. Every point in this figure represents the mean flow time over 100 problem instances, each consisting of 25000 jobs (5000 per queue). The figure shows that, for loads of 0.5 or less, the performance of the four methods is almost identical, although *makespan* has a slight advantage over the remaining three approaches. For loads greater than 0.5, *makespan* achieves the lowest mean flow times. Moreover, the difference in performance between *makespan* and the other methods grows as the load increases. *FCFS* results in the highest flow times, while the relative performance of *completionTime* and $SPT_{sum}$ changes as the loads change. For loads of 0.65 or less, the *completionTime* model has a slight advantage over $SPT_{sum}$; when the load increases to 0.85, the reverse is true. We also simulated an asymmetrical system consisting of five queues with different loads. The results for this system matched the pattern observed in Figure 3.

## Dynamic Flow Shop

To evaluate the performance of our four methods in a dynamic flow shop, we considered a system with exponential inter-arrival times and exponential processing times with the

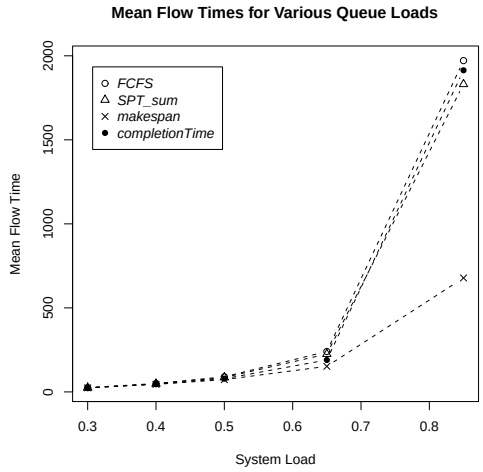


Figure 3: Mean flow times in a polling system with a two-machine flow shop server for *FCFS*, $SPT_{sum}$, *completionTime* and *makespan* models as the system load varies.

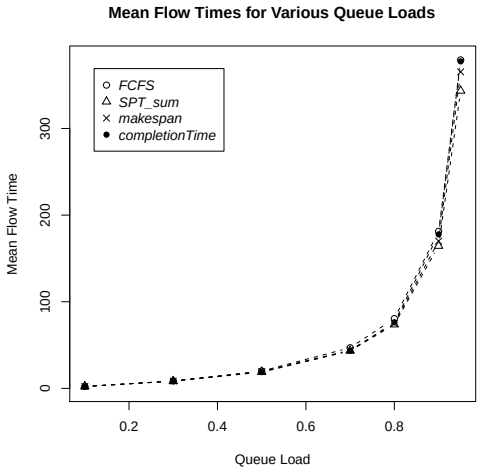


Figure 4: Mean flow times in a dynamic two-machine flow shop for *FCFS*, $SPT_{sum}$, *completionTime* and *makespan* models as the system load varies.

same means on both machines, with the arrival distribution and processing time distributions being independent. We fixed the inter-arrival rate, $\lambda$, to 10, and varied the load on the system by changing the means of the processing time distributions from 100 to 10.53. The results of these experiments are shown in Figure 4. Each point in the figure represents the mean flow time over 100 instances of 55000 jobs each.

Figure 4 shows that the relative performance of the four methods is different without a polling structure. Although there is no significant difference between the methods, $SPT_{sum}$ is slightly better than the rest, especially when the load of the system is at or above 0.9. The mean flow times obtained by the *completionTime* model are comparable to those of *FCFS*, with the *makespan* model being only marginally better.

We have obtained the same results for problems with processing times drawn from the uniform distribution $U[1, 99]$ as well as from exponential distributions with different means on the two machines.

## Makespan vs. Total Completion Time

In our analysis, we focus on understanding the performance differences between the *makespan* and *completionTime* models only. We return to *FCFS* and $SPT_{sum}$ in the Discussion section. Since both approaches are based on sub-problem scheduling, we evaluate the overall impact of the differences between the two in each sub-problem solution. Two objectives are of interest: the total completion time and the makespan. To simplify our analysis, we assume that the arrival processes are such that exactly $n$ jobs are available at the time when a sub-schedule needs to be constructed, and that both models solve exactly the same sub-problems. We discuss these assumptions later in the paper.

### Notation

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

Given that there are $n$ jobs per sub-problem, we assume that the jobs of sub-problem $i$, $i \geq 1$, are indexed from $(i-1)n+1$ to $in$. Thus, for sub-problem $i$, $i \geq 1$, the sum of completion times for the *completionTime* model (i.e., the optimal sum of completion times) is $\sum_{j=(i-1)n+1}^{in} \hat{C}_j$, and the sum of completion times for the *makespan* model is $\sum_{j=(i-1)n+1}^{in} C_j^M$. Similarly, the makespan obtained from solving the $i$th problem using the *completionTime* model is denoted $\hat{C}_{max_i}$, while the makespan from the *makespan* model (i.e., the optimal makespan) is denoted $C_{max_i}^M$. The makespan of an arbitrary schedule is denoted by $C_{max}$.

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

$$\sum_{i=1}^{T} \sum_{j=(i-1)n+1}^{in} C_j^M - \sum_{i=1}^{T} \sum_{j=(i-1)n+1}^{in} \hat{C}_j. \qquad (1)$$

Additionally, denote by $\hat{C}_j^0$ and $C_j^{M0}$ the end times of job $j$ given by the *completionTime* model and the *makespan* model, respectively, under the assumption that the first job

of the sub-problem to which $j$ belongs starts at time 0. For example, in Figure 1, $C_7 = 17$ while $C_7^0 = 3$. We refer to the values of $C_j^0$ as the *initial* completion times and the values of $C_j$ as the *actual* completion times.

## Polling Systems

We view the global polling system schedule as a sequence of $n$-job sub-schedules, which allows us to disregard the information about the queue to which a particular sub-problem belongs and makes our analysis simpler and independent of the polling order.

Given that the server switches between queues (and sub-problems) at the completion time of the last job on the second machine, every $C_j$ in the dynamic problem can be represented in terms of $C_j^0$ and the sum of the makespans of the preceding sub-problems. That is, if job $j$ belongs to sub-problem $i$, $i \geq 2$, then $C_j = \sum_{k=1}^{i-1} C_{max_k} + C_j^0$: the actual completion time of job $j$ is its initial completion time *shifted* forward in time by the sum of the makespans of all previous sub-problems. For example, in Figure 1 we see that the end time of every job belonging to sub-problem 2 is shifted by 5 time units, while the end times of jobs in sub-problem 3 are shifted by 14 units. Equation (1) can be written in terms of $C_j^0$ and $C_j^{M0}$ as follows:

$$\begin{aligned} & n\sum_{i=2}^{T}\sum_{k=1}^{i-1} C_{max_k}^{M} + \sum_{i=1}^{T}\sum_{j=(i-1)n+1}^{in} C_j^{M0} \\ & -n\sum_{i=2}^{T}\sum_{k=1}^{i-1} \hat{C}_{max_k} - \sum_{i=1}^{T}\sum_{j=(i-1)n+1}^{in} \hat{C}_j^0 \\ & = n[(T-1)\Delta_1(C_{max}) + (T-2)\Delta_2(C_{max}) + \dots \\ & +\Delta_{(T-1)}(C_{max})] + \Delta_1(\sum C_j^0) + \dots + \Delta_T(\sum C_j^0). \quad (2) \end{aligned}$$

Thus, we can calculate the difference in the sum of completion times for $T$ periods by evaluating $T$ static sub-problems under the assumption that each sub-schedule is started at time 0. When $\sum_{i=1}^{T} \Delta_i(\sum C_j) \leq 0$, we know that the *makespan* model is as good or better than the *completionTime* model for the objective of minimizing the total completion time over $T$ time periods. From Equation (2) we see that this condition depends on the values of $n$ and $T$, and the magnitude of the differences in the sums of completion times and in the makespans of the sub-problems. Specifically, as $n$ and $T$ grow, $n[(T-1)\Delta_1(C_{max}) + \dots + \Delta_{(T-1)}(C_{max})]$ decreases since $\Delta_i(C_{max}) \leq 0 \ \forall i$. However, an increase in $T$ also leads to a greater number of $\Delta_i(\sum C_j)$'s in Equation (2), which are generally much larger than the values of $|\Delta_i(C_{max})|$.

In order to better understand Equation (2), we ran experiments with 1000 static flow shop instances consisting of 5 jobs and 1000 instances consisting of 10 jobs. We then created 1000 dynamic problems by taking $T$ instances from each of these sets of static problems and assuming that the order in which these sub-problems are sampled is the order in which they are encountered in the dynamic problem. The optimal makespan of each static problem was found using the *makespan* approach, while the minimum sum of completion times was found by the *completionTime* model, which was run to optimality. The values of $\sum_{i=1}^{T} \Delta_i(\sum C_j)$, obtained using Equation (2), are shown in Figure 5 for increasing values of $T$. Most importantly, the figure empirically demonstrates that, as $T$ goes to infinity, the *makespan* model performs better than the *completionTime* model. A higher $n$ leads to a greater rate of decrease in the value of $\sum_{i=1}^{T} \Delta_i(\sum C_j)$ as $T$ increases.

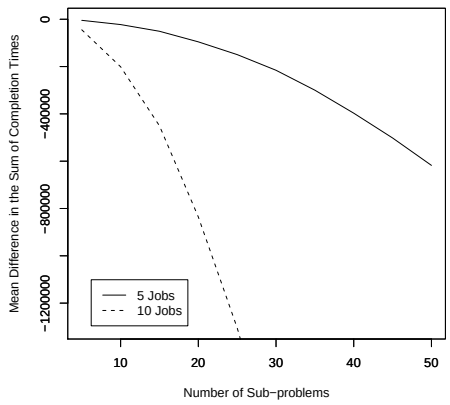


Figure 5: Mean differences in the sum of completion times between the *makespan* model and the *completionTime* model for instances with 5 and 10 job sub-problems.

Therefore, the conclusions of our investigation are consistent with our motivations for considering methods based on optimizing makespan and total completion time. Indeed, as the number of sub-problems increases, the *makespan* model, which considers long-run objectives, achieves significant gains over the *completionTime* model, which optimizes short-run behaviour.

## Dynamic Flow Shop

Unlike in the polling model, in a dynamic two-machine flow shop, jobs that belong to the same sub-problem may be shifted by different amounts, depending on three factors: the sum of makespans of the previous sub-problems, the sum of machine 1 processing times of the previous sub-problems and the sum of idle times preceding the job in the static sub-problem schedule.

Let $p_{1j}$ be the duration of job $j$ on machine 1. Denote by $\xi_j$ the total idle time before job $j$ of sub-problem $i$ minus the machine 1 processing time of the first job in the static schedule for $i$ that starts at time 0. For example, in sub-problem 1 of Figure 2, $p_{11} = 1$, $\xi_1 = \xi_2 = 0$ and $\xi_3 = 1$.

The maximum amount by which a job $j$ in sub-problem $i$ can be shifted depends on whether the first job of $i$ can start on machine 2 exactly at the completion time of the last job of the preceding sub-problem. The job in the first position of the schedule for $i$ is always shifted by this maximum amount. For instance, in Figure 2, $J4$ cannot start at time 5 because the sum of processing times on machine 1 for sub-problem 1 plus $p_{14}$ is greater than 5; thus, the completion time of $J4$ is shifted by the total machine 1 processing time of sub-problem 1, i.e., 4 time units. Operation 2 of $J7$, on the other hand, can start at the completion time of the last

job of the previous sub-problem, time 13. The shift of $J7$ is equal to the sum of the preceding makespans on machine 2 minus, to avoid double-counting, $p_{17}$. Formally, the maximum shift of $i$, $S_i^{max}$, for $i \geq 2$ is:

$$S_i^{max} = \max\{\sum_{g=1}^{(i-1)n} p_{1g}, \sum_{k=1}^{i-1} C_{max_k} - p_{1,(i-1)n+1}\}.$$

Any job belonging to sub-problem $i$ can be shifted by at most the same amount as the first job in $i$, but may be shifted by a smaller amount due to idle times $\xi_j$. The actual shift of job $j$ in sub-problem $i$, $i \geq 2$, is a function of $S_i^{max}$ and $\xi_j$:

$$S_j = \max\{S_i^{max} - \xi_j, \sum_{g=1}^{(i-1)n} p_{1g}\}. \quad (3)$$

The shift of any job $j$ belonging to sub-problem 1 is 0. Hence, in a dynamic flow shop, Equation (1) becomes:

$$\begin{aligned} &\sum_{i=1}^{T}(\sum_{j=(i-1)n+1}^{in} S_j^M + \sum_{j=(i-1)n+1}^{in} C_j^{M0}) \\ &- \sum_{i=1}^{T}(\sum_{j=(i-1)n+1}^{in} \hat{S}_j + \sum_{j=(i-1)n+1}^{in} \hat{C}_j^0) \\ &= [\Delta_1(S) + \Delta_2(S) + \ldots + \Delta_T(S)] \\ &+[\Delta_1(\sum C_j^0) + \ldots + \Delta_T(\sum C_j^0)] \end{aligned} \quad (4)$$

where $\Delta_i(S) = \sum_{j=(i-1)n+1}^{in}(S_j^M - \hat{S}_j)$, and $S_j^M$ and $\hat{S}_j$ are the shift values of job $j$ in the *makespan* and *completionTime* methods, respectively. Similarly to Equation (2), this expression is dependent on the values of $n$, $T$, and, more importantly, the values of $\Delta_i(S)$ and $\Delta_i(\sum C_j^0)$.

Under our assumptions, $\Delta_i(\sum C_j^0) \geq 0$ for all $i$. Thus, in order for the *makespan* model to be better than the *completionTime* model (i.e., for $\sum_{i=1}^{T} \Delta_i(\sum C_j)$ to be negative), the sum of $\Delta_i(S)$ values has to be negative. However, the value of $\Delta_i(S)$ will be negative only if the maximum shift for $i$ resulting from the *makespan* model is smaller than the maximum shift from the *completionTime* model, and there is at least one job for which the actual shift of $j$, $S_j$, is smaller in the solution from the *makespan* model than in the solution from the *completionTime* model. In all other situations, $\Delta_i(S)$ will be $\geq 0$. We provide a complete analysis of these values in the appendix. Essentially, this analysis shows that the difference in the shifts between the two models is not likely to be significant and is not as directly dependent on the value of the makespan as it is in the polling system.

In order to better understand the behaviour of the *makespan* and *completionTime* models in the dynamic flow shop environment, we determined the mean values of $n$, $T$, $\Delta_i(\sum C_j^0)$ and $\Delta_i(S)$ for the problem sets used to create Figure 4, without employing Equation (4). The results are presented in Table 1. Recall that the *completionTime* model was run with a time limit of 1 second.

Firstly, we note that the mean values of $n$ and $T$ were the same for all four solution methods that we have experimented with. Secondly, we see that as the load increases, the number of sub-problems solved decreases, while the sub-problem size increases. These changes coincide with an increase in the mean flow time for all solution methods. Increasing $T$ and $n$ will favourably affect the performance of the *makespan* model only if the values of $\Delta_i(S)$ are large enough to counteract the $\Delta_i(\sum C_j^0)$'s.

Table 1: Mean Statistics for the Dynamic Flow Shop Experiments

| Load | 0.1 | 0.5 | 0.7 | 0.9 | 0.95 |
|---|---|---|---|---|---|
| $n$ | 1.05 | 1.27 | 1.67 | 3.39 | 5.66 |
| $T$ | 52541 | 43287 | 33108 | 16274 | 9959 |
| $\Delta_i(\sum C_j^0)$ | 0.03 | 1.07 | 3.43 | 1.16 | 9.64 |
| $\Delta_i(S)$ | 0.00 | -0.39 | -2.60 | -27.53 | -76.62 |

From Table 1, we see that when the load is 0.1, the mean of $\Delta_i(S)$ is 0. This result makes sense, since the mean of $n$ is 1.05, i.e., the load on the system is so low that both models are essentially solving sub-problems consisting of one job. When the load is greater than 0.1 and less than or equal to 0.7, the values of $\Delta_i(S)$ are negative but smaller in magnitude than the values of $\Delta_i(\sum C_j^0)$. For these loads, the mean flow times obtained by the *completionTime* model are better than those found by the *makespan* model. The reverse is true for loads higher than 0.7.

The fact that $|\Delta_i(S)| > \Delta_i(\sum C_j^0)$ for higher loads may seem to contradict our initial intuition that $|\Delta_i(S)|$ values should generally be smaller than the $\Delta_i(\sum C_j^0)$'s. This discrepancy can be explained by the fact that our results are based on the *completionTime* model being run with a time limit of 1 second. At higher loads, when the average sub-problem size is greater, the *completionTime* model frequently provides a schedule that is worse than the one provided by the *makespan* model (if no feasible solution is found within the time limit, a *FCFS* schedule is used; otherwise, the schedule found may still be far from the optimal and worse than the one provided by the *makespan* model).

Our results show that the main parameters that influence the relative performance of methods based on makespan and total completion time minimization are the magnitudes of $\Delta_i(S)$ and $\Delta_i(\sum C_j^0)$. We know that when the *completionTime* model is run to optimality, $\Delta_i(\sum C_j^0)$ will tend to be greater than $|\Delta_i(S)|$; hence, theoretically, the *completionTime* model is a better choice for solving the dynamic flow shop problem than the *makespan* approach. Practically, however, if the *completionTime* model is not given enough time, it may produce solutions that lead to lower $\Delta_i(\sum C_j^0)$, in which case the effect of $|\Delta_i(S)|$ may be significant enough to allow the *makespan* model to achieve better overall mean flow times.

## Discussion

### Polling System vs. Dynamic Flow Shop

Equations (2) and (4), together with our empirical results, allow us to explain the differences in the behaviour of the *makespan* and the *completionTime* models in the polling system and in the dynamic flow shop.

In the case of the polling system, our analysis demonstrates a conflict between short-run and long-run objectives. That is, minimization of the sum of completion times at each scheduling point results in the best performance for the current sub-problem, but leads to poor performance in the long-run. Minimization of the makespan, on the other hand, leads

to sub-optimal sum of completion time values for each sub-problem, but results in significant overall mean flow time improvements. In the dynamic two-machine flow shop there is no conflict between the way in which we have attempted to optimize long-run and short-run objectives – minimizing the total completion time of each sub-problem also leads to better mean flow time in the long-run than does minimizing makespan.

### Sub-problem Size Assumptions

One simplifying assumption that we made in the above analysis is that every sub-problem consists of exactly $n$ jobs. Suppose now that the number of jobs changes, and denote the number of jobs in sub-problem $i$ by $n_i$. If both the *makespan* and the *completionTime* models encounter exactly the same sub-problems, then for every sub-problem $i$, one needs only to substitute $n_i$ for the corresponding occurrence of $n$ in Equations (2) and (4) in order to make them valid. $n_i$ jobs would be used in the calculations of $\Delta_i(\sum C_j^0)$, $\Delta_i(C_{max})$ and $\Delta_i(S)$ for every $i$ as well.

Consider now our assumption that the two models solve exactly the same sub-problems. In the dynamic flow shop, this is in fact always true, since a new schedule is created when the last job in the previous sub-problem is completed on machine 1, which occurs at the same time in both the *completionTime* and the *makespan* models. In the polling system, the scheduling time point directly depends on the makespans of all previous sub-problems. Thus, a difference between the makespans resulting from the *makespan* and the *completionTime* models may imply that the *makespan* model will solve more sub-problems with a smaller number of jobs. However, this situation can happen only if the arrival rate and $|\Delta_i(C_{max})|$ values are sufficiently high: there has to exist some sub-problem $i$ such that there is at least one job in the queue at the completion of this sub-problem in the *makespan* model and there is an arrival in the time interval between $\sum_{k=1}^{i} C^M_{max_k}$ and $\sum_{k=1}^{i} \hat{C}_{max_k}$. For the polling model, a higher $T$ and a higher $|\Delta_i(C_{max})|$ value will imply an even greater advantage for the *makespan* model; a smaller $n$ may counteract those effects, however. Equation (2) would not be directly applicable if the two models encountered different problems, but it would be an approximation of the actual difference in the sum of completion times. Moreover, the insights gained from this equation would still be applicable.

Therefore, our assumptions regarding the fixed sub-problem size for both models greatly simplify the analysis without taking away from the generality of its conclusions.

### *SPT$_{sum}$* and *FCFS*

For the dynamic flow shop scheduling problem, experiments have shown that $SPT_{sum}$ has an advantage over the other methods for loads greater than 0.7. This performance is due to the $SPT_{sum}$ model finding better solutions for the total completion time objective than all the other methods. These observations are supported by the results of Xia et al. (Xia, Shanthikumar, and Glynn 2000), who have shown that $SPT_{sum}$ is asymptotically optimal for the average completion time objective as the number of jobs in a two-machine flow shop increases. *FCFS*, on the other hand, is the worst performer, due to the fact that it does not use processing time information, whereas the other methods do. Further investigation has shown that *FCFS* finds sub-problem solutions that are poor both in terms of their makespan and their total completion time values.

Our results indicate that, in a polling system with a flow shop-like server, an approach that would find, for each sub-problem, the minimum makespan schedule with the best total completion time value would outperform the methods we presented here. This hybrid method would have the long-run focus of the *makespan* model, but would also be better than *makespan* in the short-run. In a dynamic flow shop, an approach that finds the optimal completion time schedule with the smallest makespan is of interest. However, our results imply that this approach would provide only marginal improvements over the *completionTime* model. Investigation of these hybrid methods is part of our future work.

## Related Work

From the queueing perspective, our paper is related to the literature on tandem queues (Towsley and Baccelli 1991) and polling systems (Takagi 2000; Levy and Sidi 1990), while from the scheduling perspective, it is related to research on flow shop scheduling (Hejazi and Saghafian 2005; Della Croce, Narayan, and Tadei 1996; Xia, Shanthikumar, and Glynn 2000) and scheduling in dynamic and uncertain environments (Aytug et al. 2005; Bidot et al. 2009). Thus, we see a variety of ways for integration of ideas from these areas in the future. For example, we would like to look at the optimization of polling order; 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 develop hybrid queueing/scheduling methods.

## Conclusion

We have considered the problem of minimizing the mean flow time in two-machine dynamic flow shop environments from the perspectives of queueing theory and scheduling. Queueing theory approaches tend to optimize the long-run expected performance of the system, while scheduling focuses on actual short-run objectives. Taking both viewpoints into account, we show that, in a polling system, a method which incrementally constructs the schedule by minimizing the makespan of each subsequent set of jobs results in a better mean flow time than does a method that minimizes the mean flow time itself. In a dynamic flow shop, the reverse is true. We provide an analysis of the differences of scheduling methods in these two environments. This paper demonstrates that combining ideas from scheduling and queueing theory can lead to new insights about dynamic scheduling.

## Appendix

Our analysis of $\Delta_i(S)$ for the dynamic flow shop is divided into four cases based on the values of $S_i^{maxM}$ and $\hat{S}_i^{max}$.

**Case 1:** $S_i^{maxM} = \hat{S}_i^{max} = \sum_{g=1}^{(i-1)n} p_{1g}$. Since $S_j = \max\{S_i^{max} - \xi_j, \sum_{g=1}^{(i-1)n} p_{1g}\}$ and $\xi_j \geq 0$, the values of $S_j^M$ and $\hat{S}_j$ are both equal to $\sum_{g=1}^{(i-1)n} p_{1g}$. Because this is true for all jobs belonging to sub-problem $i$, $\Delta_i(S) = 0$.

**Case 2:** $S_i^{maxM} = \sum_{g=1}^{(i-1)n} p_{1g}$ and $\hat{S}_i^{max} = \sum_{k=1}^{i-1} \hat{C}_{max_k} - \hat{p}_{1,(i-1)n+1}$. As in case 1, $S_j^M = \sum_{g=1}^{(i-1)n} p_{1g}$ for all $j$ in sub-problem $i$. $\hat{S}_j$ can be:

**(a)** $\hat{S}_j = \max\{\hat{S}_i^{max} - \hat{\xi}_j, \sum_{g=1}^{(i-1)n} p_{1g}\} = \hat{S}_i^{max} - \hat{\xi}_j$

$$= \sum_{k=1}^{i-1} \hat{C}_{max_k} - \hat{p}_{1,(i-1)n+1} - \hat{\xi}_j$$

In this situation, $S_j^M - \hat{S}_j = \sum_{g=1}^{(i-1)n} p_{1g} - (\sum_{k=1}^{i-1} \hat{C}_{max_k} - \hat{p}_{1,(i-1)n+1} - \hat{\xi}_j) \leq 0$ (since $\sum_{k=1}^{i-1} \hat{C}_{max_k} - \hat{p}_{1,(i-1)n+1} - \hat{\xi}_j$ is $\geq \sum_{g=1}^{(i-1)n} p_{1g}$). Thus, employing the *makespan* model would lead to a smaller shift value for job $j$ than using the *completionTime* model.

**(b)** $\hat{S}_j = \max\{\hat{S}_i^{max} - \hat{\xi}_j, \sum_{g=1}^{(i-1)n} p_{1g}\} = \sum_{g=1}^{(i-1)n} p_{1g}$, and thus $S_j^M - \hat{S}_j = 0$.

Overall in case 2, we see that $\Delta_i(S) \leq 0$ and the magnitude of $\Delta_i(S)$ depends on the number of jobs satisfying the conditions of **(a)** and **(b)**.

**Case 3:** $S_i^{maxM} = \sum_{k=1}^{i-1} C_{max_k}^M - p_{1,(i-1)n+1}^M$ and $\hat{S}_i^{max} = \sum_{g=1}^{(i-1)n} p_{1g}$. This case is possible only if $p_{1,(i-1)n+1}^M - \hat{p}_{1,(i-1)n+1} \leq 0$. There are two sub-cases:

**(a)** $S_j^M = \max\{S_i^{maxM} - \xi_j^M, \sum_{g=1}^{(i-1)n} p_{1g}\}$

$$= S_i^{maxM} - \xi_j^M = \sum_{k=1}^{i-1} C_{max_k}^M - p_{1,(i-1)n+1}^M - \xi_j^M$$

Therefore, $S_j^M - \hat{S}_j = (\sum_{k=1}^{i-1} C_{max_k}^M - p_{1,(i-1)n+1}^M - \xi_j^M) - \sum_{g=1}^{(i-1)n} p_{1g} \geq 0$.

**(b)** $S_j^M = \max\{S_i^{maxM} - \xi_j^M, \sum_{g=1}^{(i-1)n} p_{1g}\} = \sum_{g=1}^{(i-1)n} p_{1g}$ and $S_j^M - \hat{S}_j = 0$.

In case 3, therefore, $\Delta_i(S) \geq 0$, and as in case 2, the magnitude of $\Delta_i(S)$ depends on the number of jobs satisfying the conditions of **(a)** and **(b)**.

**Case 4:** $S_i^{maxM} = \sum_{k=1}^{i-1} C_{max_k}^M - p_{1,(i-1)n+1}^M$ and $\hat{S}_i^{max} = \sum_{k=1}^{i-1} \hat{C}_{max_k} - \hat{p}_{1,(i-1)n+1}$. There are four sub-cases:

**(a)** $S_j^M = \hat{S}_j = \sum_{g=1}^{(i-1)n} p_{1g}$, or $S_j^M - \hat{S}_j = 0$, which would happen if $\xi_j^M \leq \sum_{k=1}^{i-1} C_{max_k}^M - p_{1,(i-1)n+1}^M - \sum_{g=1}^{(i-1)n} p_{1g}$ and $\hat{\xi}_j \leq \sum_{k=1}^{i-1} \hat{C}_{max_k} - \hat{p}_{1,(i-1)n+1} - \sum_{g=1}^{(i-1)n} p_{1g}$.

**(b)** $S_j^M = \sum_{g=1}^{(i-1)n} p_{1g}$ and $\hat{S}_j = \sum_{k=1}^{i-1} \hat{C}_{max_k} - \hat{p}_{1,(i-1)n+1} - \hat{\xi}_j$, or $S_j^M - \hat{S}_j = \sum_{g=1}^{(i-1)n} p_{1g} - (\sum_{k=1}^{i-1} \hat{C}_{max_k} - \hat{p}_{1,(i-1)n+1} - \hat{\xi}_j) \leq 0$.

**(c)** The reverse of **(b)**. $\hat{S}_j = \sum_{g=1}^{(i-1)n} p_{1g}$ and $S_j^M = \sum_{k=1}^{i-1} C_{max_k}^M - p_{1,(i-1)n+1}^M - \xi_j^M$, or $S_j^M - \hat{S}_j = (\sum_{k=1}^{i-1} C_{max_k}^M - p_{1,(i-1)n+1}^M - \xi_j^M) - \sum_{g=1}^{(i-1)n} p_{1g} \geq 0$.

**(d)** $S_j^M = \sum_{k=1}^{i-1} C_{max_k}^M - p_{1,(i-1)n+1}^M - \xi_j^M$ and $\hat{S}_j = \sum_{k=1}^{i-1} \hat{C}_{max_k} - \hat{p}_{1,(i-1)n+1} - \hat{\xi}_j$, or $S_j^M - \hat{S}_j = \sum_{k=1}^{i-1} C_{max_k}^M - p_{1,(i-1)n+1}^M - \xi_j^M - (\sum_{k=1}^{i-1} \hat{C}_{max_k} - \hat{p}_{1,(i-1)n+1} - \hat{\xi}_j)$, which may or may not be positive.

In case 4, $\Delta_i(S)$ would be negative only if the shift savings achieved by the solutions of sub-problems that satisfy the conditions of **(b)** and **(d)** outweigh the cumulative effect of the rest of the sub-problems.

## References


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

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

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

Conway, R. W.; Maxwell, W. L.; and Miller, L. W. 1967. *Theory of Scheduling*. Addison-Wesley.

Della Croce, F.; Narayan, V.; and Tadei, R. 1996. The two-machine total completion time flow shop problem. *European Journal of Operational Research* 90(2):227–237.

Hejazi, S. R., and Saghafian, S. 2005. Flowshop scheduling problems with makespan criterion: a review. *International Journal of Production Research* 43:2895–2929.

Kovács, A., and Beck, J. 2010. A global constraint for total weighted completion time for unary resources. *Constraints*. To Appear.

Levy, H., and Sidi, M. 1990. Polling systems: Applications, modeling, and optimization. *IEEE Transactions on Communications* 38(10):150–1760.

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

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

Takagi, H. 2000. Analysis and application of polling models. In *Performance Evaluation: Origins and Directions*, Lecture Notes in Computer Science. Springer. 423–442.

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

Wierman, A.; Winands, E.; and Boxma, O. 2007. Scheduling in polling systems. *Performance Evaluation* 64:1009–1028.

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