Proceedings of the Twenty-Third International Conference on Automated Planning and Scheduling

Hybrid Queueing Theory and Scheduling Models for Dynamic Environments with Sequence-Dependent Setup Times

Tony T. Tran[1] and **Daria Terekhov**[1] and **Douglas G. Down**[2] and **J. Christopher Beck**[1]
[1]Department of Mechanical and Industrial Engineering, University of Toronto, Toronto, Ontario, Canada
{tran, dterekho, jcb}@mie.utoronto.ca
[2] Department of Computing and Software, McMaster University, Hamilton, Ontario, Canada
downd@mcmaster.ca

Abstract

Classically, scheduling research in artificial intelligence has concentrated on the combinatorial challenges arising in a large, static domain where the set of jobs, resource capacities, and other problem parameters are known with certainty and do not change. In contrast, queueing theory has focused primarily on the stochastic arrival and resource requirements of new jobs, de-emphasizing the combinatorics. We study a dynamic parallel scheduling problem with sequence-dependent setup times: arriving jobs must be assigned (online) to one of a set of resources. The jobs have different service times on different resources and there exist setup times that are required to elapse between jobs, depending on both the resource used and the job sequence. We investigate four models that hybridize a scheduling model with techniques from queueing theory to address the dynamic problem. We demonstrate that one of the hybrid models can significantly reduce observed mean flow time performance when compared to the pure scheduling and queueing theory methods. More specifically, at high system loads, our hybrid model achieves a 15% to 60% decrease in mean flow time compared to the pure methodologies. This paper illustrates the advantages of integrating techniques from queueing theory and scheduling to improve performance in dynamic problems with complex combinatorics.

1 Introduction

Many real-world scheduling problems are dynamic: jobs arrive over time and the existence of a job is not known before it arrives. Typical scheduling approaches to dynamic scheduling problems solve a series of inter-dependent static problems (Bidot et al. 2009), allowing the techniques that have been developed for static problems to be applied. However, such methods suffer from sub-optimal long-run solution quality due to optimization over only a short time horizon. Thus, decisions made during one static scheduling period can have unforeseen impact on a later period. The study of queueing theory, in contrast, is primarily concerned with dynamic environments, tending to consider relatively simple combinatorics in order to develop rigorous mathematical results (Gross and Harris 1998).

It has recently been proposed that the hybridization of scheduling and queueing is a fruitful direction for research into problems which are challenging both from combinatorial and stochastic perspectives (Terekhov et al. 2012). While that work demonstrated substantial benefits from considering both scheduling and queueing from a theoretical view, a substantial gap in that paper stems from the fact that no hybrid algorithm was explored. We address a more challenging underlying scheduling problem and develop a hybrid model that outperforms the pure queueing and scheduling approaches stemming from the work of Terekhov et al. (2012).

The specific problem of interest in this paper is a dynamic unrelated parallel machine scheduling problem with sequence-dependent setup times. This problem is dynamic and combinatorially complex, and inspired by a problem from the queueing literature. We present a scheduling model and four modifications to this model using concepts from queueing to create hybrid queueing/scheduling models. We illustrate experimentally that guiding scheduling with queueing theory improves the performance significantly. Our experiments show that a hybrid model can be constructed to provide the lowest mean flow time when compared to pure scheduling or queueing models.

In the following section, the dynamic scheduling problem is defined. The background and motivation for our work is found in Section 3. Section 4 briefly presents a decomposition-based scheduling model for the static problem. Hybridization of the scheduling model to incorporate queueing guidance and apply it to the dynamic version of the problem is proposed in Section 5. Experimental results are then presented, followed by a discussion.

2 Problem Definition

The problem of interest is inspired by Andradóttir, Ayhan, and Down (2003). It consists of M, a set of heterogeneous resources, and K, a set of independent job classes. Jobs arrive over time via an arrival process with rate λ and independent and identically distributed (i.i.d.) inter-arrival times. Each job belongs to a class $k \in K$ with probability pr_k. When a job enters the system and is not immediately served, it waits in a queue of infinite capacity. Each job can be served by a number of resources but must be assigned to only one. We assume that there are no pre-emptions. When a job j of class k is assigned to resource i, a service time of p_{ij} that is i.i.d. with rate $\mu_{ik} = \frac{1}{E[p_{ij}]}$ is required. Both the existence

of a job and its service time become known upon arrival.

At any time when a resource is not busy, it may switch to serving a different class. However, a setup time is incurred when making the change. The nth occurrence of resource i switching from class k to class l has a setup time $s_{ikl}(n)$. Such a definition for setup times differs from standard scheduling definitions which often assume setting up is strictly dependent on the jobs or classes scheduled directly before and after the setup; the same setup between two classes will always be the same. We use a more general queueing theory definition of setup times where switching classes incurs a setup and the magnitude varies each time a setup occurs. This definition follows that of Andradóttir, Ayhan, and Down (2003). The setup times are generated such that they follow the triangle inequality so that the shortest time to change from serving a class k to a class l is a direct switch. Section 6 will provide further details on the generation of setup times.

The goal is to assign jobs to resources and sequence the jobs so as to minimize the mean flow time: the mean, over all jobs, of the time between when a job arrives to the system and when it completes service.

3 Background and Motivation

Queueing theory and scheduling have both developed techniques to deal with dynamic resource allocation and sequencing. In the scheduling literature, dynamic problems are often handled by periodically solving static, deterministic scheduling problems (Bidot et al. 2009; Ouelhadj and Petrovic 2009). Such approaches apply existing scheduling methods to the combinatorics but are only able to reason about a short time horizon. Because the scheduling approaches typically solve an NP-complete problem during each scheduling period, it is assumed that the "time pressure" of the problem is low. That is, one time unit in the scheduling model represents a reasonable amount of "real time" and therefore we have time to use optimization techniques to improve the schedule. With high time pressure, it is likely that online decisions must be made by a polynomial dispatching rule or policy. We make the assumption of low time pressure in this work.

Queueing theory addresses dynamic problems with policies similar to dispatch rules (Gross and Harris 1998). It often involves theoretical analysis of a system under specific policies and long-run system performance metrics and guarantees. However, the analysis tends to be limited to problems with simple combinatorics.

In the queueing literature, Andradóttir, Ayhan, and Down (2003) study the problem presented above with the addition of rerouting, which allows jobs to change to another job class and stay in the system after service. They compute the maximum arrival rate for which the system can be stabilized, along with an explicit policy that guarantees stability. Stability, in queueing theory, can be informally understood to mean that the expected queue size is finite over an infinite time horizon (Dai 1995).[1] The queueing policy of

[1]We follow Terekhov et al. (2012) in adopting the queueing definition of stability, not others found in the scheduling literature (Bidot et al. 2009; Sotskov et al. 2010).

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

Figure 1: Allocation LP.

Andradóttir, Ayhan, and Down (2003) is not evaluated with respect to common scheduling performance metrics such as flow time. A stable system has better flow time performance than an unstable one. However, knowing that two different policies both stabilize a system does not provide insight about their flow time performance. From the perspective of a customer who wishes to have his/her jobs served in a timely manner, good overall behaviour of a system is not as important as measures like flow time.

Andradóttir, Ayhan, and Down (2003) present a linear programming (LP) model to find the maximum arrival rate, λ^*, for which the system can be stabilized. Figure 1 shows the allocation LP, adjusted to the problem we study. Here,

λ: The arrival rate of jobs,
δ_{ik}: The fractional amount of time that resource i serves jobs of class k.

Constraint (1) ensures that sufficient resources are allocated to each class to guarantee stability. Constraint (2) corresponds to not over-allocating any resource, and non-negativity is enforced by constraint (3). The allocation LP coincides with a fluid representation of the problem (Dai 1995). Such an approach examines the system at very heavy loads and is strictly concerned with system stability - as a consequence, the allocation LP can disregard setup times. As the number of jobs to be served between setups increases, the proportion of time spent setting up becomes negligible; that is,

$$\forall n, \lim_{Z\to\infty} \frac{\sum_{j=1}^{Z} p_{ij} + s_{ikl}(n)}{\sum_{j=1}^{Z} p_{ij}} = 1.$$

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 the required resource allocation proportions, δ^*_{ik}. To then make use of the solution from the LP and sequence jobs, Andradóttir, Ayhan, and Down (2003) develop the *Round Robin* policy: each resource i cyclically visits classes in V_i, where V_i is an ordered list of all classes k with $\mu_{ik}\delta^*_{ik} > 0$. A resource will serve a class $k \in V_i$ until either there are no more jobs of class k waiting or the resource has served l_{ik} jobs. Assume we are given a desired arrival rate λ, $\epsilon = \frac{\lambda^*-\lambda}{\lambda^*+\lambda}$, $m_{ik} = \frac{1}{\mu_{ik}}$, and s_i, the expected sum of setup times for resource i to serve each class in V_i and

cycle back to the initial class. Andradóttir, Ayhan, and Down (2003) show that the δ^*_{ik} values are sufficiently tracked to stabilize the system if l_{ik} is chosen to be

$$l_{ik} = \left\lceil \frac{(1-\epsilon)(s_i + \sum_{l \in K} m_{il} 1\{\delta^*_{ik} > 0\})\delta^*_{ik}}{\epsilon m_{ik}} \right\rceil.$$

Here, $1\{\delta_{ik} > 0\}$ is a function that is 1 if $\delta_{ik} > 0$ and 0 otherwise.

The *Round Robin* policy guarantees stability when λ is less than λ^* by reasoning about the proportion of time spent setting up between classes and serving jobs. As the load on a system becomes high, the number of jobs tends to infinity. At these conditions, the l_{ik} values also become high and the *Round Robin* policy ensures that the time spent setting up between classes is insignificant when compared to the time that resources are busy with jobs. By following the l_{ik} values, the resources are guaranteed to spend the correct proportion of time to manage the demand from arriving jobs. Therefore, the *Round Robin* policy guarantees stability for any system which is stabilizable.

Al-Azzoni and Down (2008) study a setting similar to the queueing network with flexible servers. In their work, heterogenous computing systems are assigned to job classes using an allocation LP. Their work does not consider setup times or provide stability guarantees, but their heuristic provides improved performance with respect to the long-run number of tasks in the system. Their work shows the potential of using the allocation LP to help improve more common scheduling performance metrics for a dynamic system.

4 A Dynamic Scheduling Model

Our first dynamic scheduling model, *MinMksp*, adopts the periodic scheduling approach. In order to make use of classically developed scheduling techniques in a dynamic environment, a periodic scheduler treats the dynamic system as many static problems in sequence. The scheduler observes the system at a given time instant and forms a schedule with the jobs that are present. Later, the scheduler will once again observe the system and generate a new schedule, incorporating the jobs that have arrived in the interim. This method allows the scheduler to solve static "snapshots" of the dynamic problem. Although periodic scheduling enables us to use classical scheduling tools for dynamic environments, it does not make use of any insight into the dynamics of the system. While solving the static schedule, decisions are made without consideration of the fact that jobs will arrive in the future. Therefore, the *MinMksp* model is a pure scheduling model. We present our hybrid models in Section 5, where the *MinMksp* model provides the framework but scheduling decisions are guided by queueing theory.

We define a scheduling period to be the time from when the scheduler creates a schedule to when any resource has completed serving all jobs that were assigned to it. In the case that the schedule does not assign any jobs to a resource, the period ends with the arrival of a new job that can be executed on any idle resource. All jobs scheduled in the previous period but not yet executed are frozen: they stay assigned to the resource and job sequence as in the previous period's schedule. The scheduling objective is to minimize makespan rather than flow time of each period. Makespan is chosen because of results indicating that short-term makespan is a stronger proxy for long-term flow time than is short-term flow time (Terekhov et al. 2012) and because makespan is empirically much easier to minimize for problems of our scale.

Although optimizing makespan is simpler than optimizing flow time, minimizing makespan for the static version of our problem is still strongly NP-hard because the single machine problem is equivalent to a traveling salesman problem (TSP) (Baker 1974). Due to the difficulty of solving the problem, much of the literature on makespan minimization with setups in a static environment has focused on heuristic approaches (Arnaout, Rabadi, and Musa 2010; Rabadi, Moraga, and Al-Salem 2006). However, we expect that finding optimal periodic solutions will be important: a solution that is sub-optimal by one time unit may result in all jobs in later periods starting one time unit later, leading to a substantial increase in flow time. Logic-based Benders decomposition (LBBD) (Hooker 2005) is a technique that has been successfully applied to a number of static scheduling problems with multiple servers. We make use of a state-of-the-art LBBD due to Tran and Beck (2012) to solve our periodic scheduling problem.

In our LBBD, the problem is decomposed into a master problem and a set of subproblems. A mixed integer programming (MIP) model is used to solve the master problem, while the subproblems are solved with a specialized traveling salesman problem (TSP) solver. In the master problem, jobs are assigned to resources. Once assignments are made, one subproblem will be created for each resource and the TSP solver sequences the assigned jobs to minimize the makespan. The LBBD algorithm then iterates between solving the master problem and subproblems, adding *cuts* from the subproblems to the master problem to create an optimal schedule. We restate the LBBD approach of Tran and Beck (2012) below for our queueing network.

4.1 Master Problem

The MIP formulation of the master problem assigns jobs to a resource and sequences them, relaxing the requirement for a complete sequence. Rather than finding an optimal single sequence of jobs on each resource, the model allows multiple sequences. These sequences share no jobs in common and together include all jobs assigned to the resource and the sum of the incurred setup times is a lower bound on the minimum sum of setup times achievable for the job set if a complete sequence was created. The MIP model is formulated in Figure 2 where:

C_{max}: Makespan of the master problem,
ξ_i: Total setup time incurred from all sequences on resource i,
x_{ij}: 1 if job j is served on resource i,
y_{ijo}: 1 if job o is served directly after job j on resource i,

$s_{ik_jk_o}(n)$: Next setup time for switching from class k_j to class k_o, where k_j is the class that job j belongs to and $n-1$ is how many times a switch between these two classes occured,
ν_i: Remaining service time of previously scheduled jobs on resource i that have not yet completed,
N: Set of jobs that need to be scheduled,
N': Set of jobs that need to be scheduled with the addition of an auxiliary job.

$$\min \quad C_{max}$$

$$\text{s.t.} \quad \sum_{j \in N} x_{ij} p_{ij} + \xi_i + \nu_i \leq C_{max}, \qquad i \in M \quad (4)$$

$$\sum_{i \in M} x_{ij} = 1, \qquad j \in N \quad (5)$$

$$\xi_i = \sum_{j \in N} \sum_{o \in N, j \neq o} y_{ijo} s_{ik_jk_o}(n), \quad i \in M \quad (6)$$

$$x_{ij} = \sum_{j \in N', j \neq o} y_{ijo}, \qquad o \in N'; i \in M \quad (7)$$

$$x_{ij} = \sum_{j \in N', j \neq o} y_{ioj}, \qquad o \in N'; i \in M \quad (8)$$

$$cuts \quad (9)$$

$$x_{ij} \in \{0;1\}, \qquad j \in N; i \in M \quad (10)$$

$$0 \leq y_{ijo} \leq 1, \qquad j,o \in N'; i \in M \quad (11)$$

Figure 2: The Master Problem Formulation.

The makespan on each resource is defined by constraint (4). Constraint (5) ensures that each job is assigned to exactly one resource. The relaxed setup time is calculated in constraint (6) where constraints (7) and (8) enforce the requirement that all assigned jobs must be accounted for in the calculation. ξ_i is a lower bound on the additional time required from setup times of the jobs assigned to resource i since we do not enforce the requirement that the sequence on a single resource is a true feasible schedule; rather, the only requirement is that a setup exists before and after each scheduled job (note that jobs of the same class incur 0 setup time). In this way, a sequence of setups may be a cycle that does not include all jobs assigned to a resource and is therefore infeasible. Thus ξ_i gives a lower bound on the feasible schedule for the set of assigned jobs. To make a single cycle including all jobs be a feasible schedule, we assign an auxiliary job to each resource ($x_{i0} = 1, \forall\, i \in M$) that is defined as the first and last job of a schedule. This auxiliary job has no service time or setup time from other jobs. However, we include a setup time from this job to other jobs depending on the most recent class of jobs served on the resource (i.e., in the previous scheduling period). Thus, a feasible schedule will start at this auxiliary job, serve each assigned job on a resource, and then return back to the auxiliary job to complete the cycle. Constraints (9) are *cuts* added to the master problem from a subproblem each time an infeasible solution is found. During the first iteration of solving the master problem, the set is empty. The final constraints (10) and (11) force the decision variables x_{ij} to be binary and y_{ijo} to be between 0 and 1.

4.2 Subproblems

When an optimal solution to the master problem is found, $|M|$ subproblems are created, one for each resource. The subproblem will sequence the assigned jobs to minimize the makespan.

A TSP formulation is used for the subproblem. Sequencing jobs on a single resource is equivalent to an asymmetric TSP if jobs are represented by nodes and the edges represent the service time of the first job plus the required setup between the jobs. Figure 3 presents the TSP and how a schedule can be defined using the TSP representation. Traveling along an edge (a, b) incurs the service time of job a and the setup time between the class of job a and the class of job b. The makespan of a schedule is defined by a cycle of the TSP from the auxiliary node (0) to all other nodes and back to the auxiliary node. If the order of nodes visited is $(0, 3, 1, 2, 0)$, the distance traveled is $s_{03} + p_3 + s_{31} + p_1 + s_{12} + p_2$. This tour distance is equal to the makespan of the job sequence $(3, 1, 2)$.

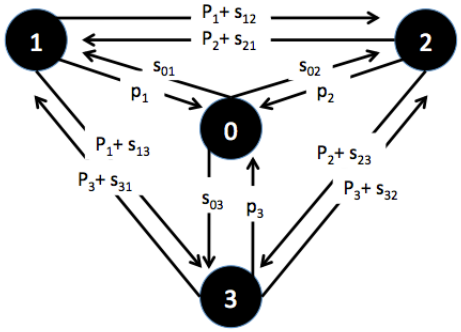

Figure 3: TSP representation.

4.3 Cuts

The solution to each subproblem is compared with C_{max} from the master problem. If a complete sequence for each resource has a makespan equal to or less than the relaxed optimal makespan obtained from the master problem, the relaxed optimal makespan is, in fact, globally optimal. Otherwise, a cut is created for each subproblem with makespan greater than the value of C_{max}.

We use a simple *no good* cut that removes the current master problem solution from the feasible solution space.

Given N_i^h, the set of jobs assigned to resource i in iteration h, the cut for a subproblem i is

$$C_{max} \geq C_{max}^{hi*} - \sum_{j \in N_i^h} (1 - x_{ij}) C_{max}^{hi*}$$

where C_{max}^{hi*} is the optimal makespan for subproblem i in iteration h. This cut implies that if the same assignment is made in future periods, the makespan must be at least as large as the makespan found in the subproblem. In the scenario where the master problem makes a different assignment, one or more x_{ij} terms will be zero and the constraint is redundant.

Tran and Beck (2012) made use of a tighter cut for their problem. Although the cut is also valid for this problem and would lead to faster solve times, we did not implement the cut for our model since the simpler *no good* cut was sufficiently fast.

LBBD iterates between the master problem and the subproblems until either all subproblems have a makespan less than or equal to the C_{max} value of the master problem or the C_{max} value is equal to the best global solution found so far. For the second condition, observe that in each iteration, the subproblems together generate a globally feasible schedule whose makespan is an upper bound on the globally optimal makespan. If a master problem subsequently finds an optimal C_{max} value that is equal to this upper bound, it has proved that no better solutions exist.

5 Hybridizing Scheduling and Queueing

Four hybridizations are proposed: *Tracking*, *Restricted*, *Reschedule*, and *Restricted* + *Reschedule*. The first three approaches each alter the *MinMksp* model to make use of ideas or analysis from queueing theory. *Restricted* + *Reschedule* combines changes proposed by *Restricted* and *Reschedule*.

5.1 Tracking

Our first approach to integrating long-term guidance into the myopic scheduling problem is to replace the objective function of the *MinMksp* model by one that accounts for the dynamics of the system. We wish to update the periodic scheduler by using a bi-criteria objective of minimizing a linear combination of the makespan and the deviation from δ_{ik}^* found by the allocation LP. We adapt the *MinMksp* model by changing the LBBD master problem in Figure 2 to the one shown in Figure 4. Here, we define the parameters:

- α: A parameter used to scale the importance of deviation from δ_{ik}^* versus the makespan,
- Δ_{ik}: Deviation between a realized assignment of resource i to class k and what δ_{ik}^* suggests,
- S_k: The set of jobs that belong to class k.

The Δ_{ik} values are defined by the master problem solution and no changes are needed in the subproblems.

By using the *Tracking* model, we are able to guide the *MinMksp* scheduling model with the high level analysis used in *Round Robin* as defined in Section 3. The *Tracking* model attempts to imitate the short-term behaviour of *MinMksp* and the long-term behaviour of *Round Robin*.

$$\begin{aligned} \min \quad & \alpha C_{max} + (1 - \alpha) \sum_{i \in M} \sum_{k \in K} \Delta_{ik} \\ \text{s.t.} \quad & \text{Constraints (4) to (11)}, \\ & \sum_{j \in S_k} x_{ij} p_{ij} - \delta_{ik}^* \sum_{j \in N} x_{ij} p_{ij} \leq \Delta_{ik}, \\ & \qquad i \in M, k \in K \quad (12) \end{aligned}$$

Figure 4: Updated Master Problem Formulation for Tracking.

5.2 Restricted

When the job arrival rate is low, resources are expected to have idle periods since the supply of resources is greater than the demand from jobs. Low resource usage causes the *MinMksp* model to make assignments that are inefficient from a long-term perspective. For example, Figure 5 presents two feasible schedules. Each job is defined by (j,k), the job's index and class. Schedule 5.a is the *MinMksp* schedule while schedule 5.b is a schedule with longer makespan. However, in the former schedule a setup is incurred between jobs 4 and 3, and that resource capacity cannot be regained. In contrast, the scheduling period for schedule 5.b ends at the end of job 4. If a new job of class 1 arrives before job 4 completes, we could have one of the two schedules in Figure 6. With the new job, we can see that 5.b is a much better schedule.

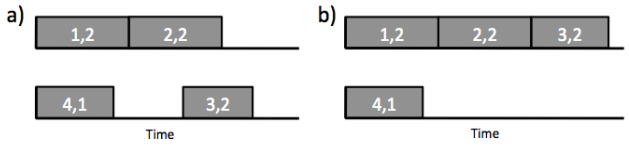

Figure 5: Possible schedules for a period.

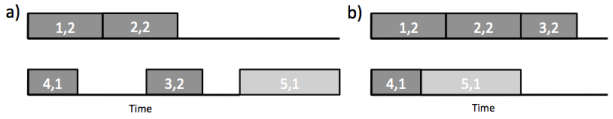

Figure 6: Possible schedules for the subsequent period.

The *MinMksp* model does not reason about jobs that may arrive in the future, and we have observed that in scheduling for short-term objectives, it often falls into the trap of Figure 6.a. To reduce the probability of such assignments, the *Restricted* model allows a job to be assigned to a resource only if the corresponding δ_{ik}^* value is non-zero; where $\delta_{ik}^* = 0$, the corresponding x_{ij} is set to 0. Therefore, if the allocation LP solution states that the decision to assign class 2 jobs to the bottom resource is inefficient, then the schedule in Figure 5.a cannot occur. With such a restriction, the schedule

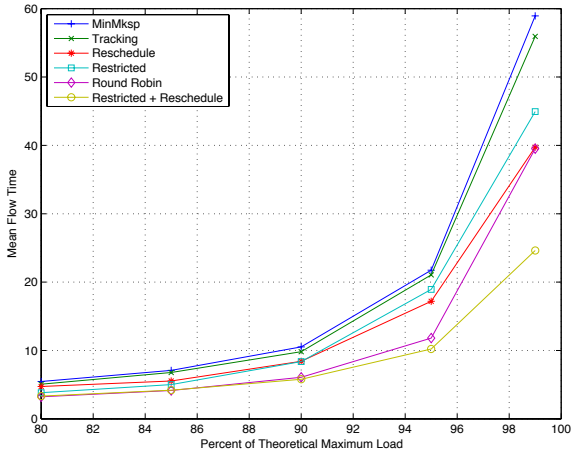

Figure 7: Comparison for a system with two resources and four job classes with short setup times.

from Figure 6.a would not occur.

5.3 Reschedule

The *MinMksp* model ignores job arrivals until the end of each scheduling period. However, the new jobs change the scheduling problem and are likely to reduce the quality of the scheduling decisions that have already been made. In contrast, the *Round Robin* policy immediately accounts for new jobs. In particular, a resource does not switch classes until after l_{ik} jobs have been served or the class queue is empty. A new job results in a non-empty queue, which is considered before making any switches.

To make *MinMksp* more responsive, we modify the definition of a scheduling period. Observations of *Round Robin* suggest that the more frequent consideration of arriving jobs results in the ability to opportunistically continue service of jobs in a particular class, avoiding a costly setup. For example, if at the beginning of a period there are three jobs of a class, then at best, the *MinMksp* model could schedule these three jobs together. For a period with many jobs, it is likely that the server must serve more jobs upon completion of these three jobs, thereby incurring a setup time. If, during service, a new job of the same class arrives, setup time can be saved by executing the new job together with the three existing jobs. We therefore redefine the end of a scheduling period to be the earliest time that a resource has no more jobs to execute or is scheduled to switch to another job class. If the end of a period occurs because a resource is switching classes, we will then examine every other resource. Those with jobs still yet to be served can either be committed, which means that the resource will continue to serve the job following the original schedule, or removed from the current schedule to be rescheduled with the new jobs. If a resource has incomplete jobs, we commit only those belonging to the same class as the current job being served. Any other jobs not belonging to the class of jobs currently being served by a resource are pooled with the newly arrived jobs and rescheduled. Here, ν_i is calculated to be the remaining time to serve only the jobs committed to resource i. Rescheduling in such a manner allows the scheduler to consider if it is worthwhile to incur a setup given the most up-to-date information on the existing jobs. As a result, each resource serves only one job class for any given scheduling period.

Unfortunately, this change may result in starvation for a job class. At heavy loads it may be optimal in the short-term to continue executing jobs of the same class, thereby avoiding all setup times but failing to ever serve jobs in some classes. To avoid such starvation, we use l_{ik} of the *Round Robin* policy to limit the number of consecutive jobs of class k that resource i can serve. Service is halted after the l_{ik}th job and rescheduling occurs immediately with changes to the optimization model. We set the corresponding y_{ijo} values to 0 in the LBBD master problem and remove edges linking the source node to all jobs in class k. These changes remove the ability of LBBD to consider serving a job from the offending class first in the following period.

5.4 Restricted + Reschedule

We also incorporate both the *Restricted* and *Reschedule* changes to *MinMksp*. The two modifications are motivated by different observed strengths of the *Round Robin* policy: the ability to incorporate system dynamics at a high level (allocation LP) and the use of more up-to-date information than the *MinMksp* model. The hybrid model incorporating both changes makes assignments with positive δ^*_{ik} values only and ends a scheduling period prior to performing a setup.

6 Empirical Investigations

We test the models by simulating two systems: a system with two resources and four job classes and another one with four resources and four job classes. In the first system, the resources must switch between classes more often than in the second system. Furthermore, each system is tested under two setup time configurations: long and short. When setup times are long, the setup is in expectation an order of magnitude larger than the mean service times. For short setup times, the mean setup and service times are equal in magnitude. The difference in setup times is used to understand the effects of using the scheduling algorithms which make direct use of the exact setup times with the *Round Robin*, which does not. One would expect that ignoring setup times in the allocation LP will lead to much worst performance as the magnitude of setup times increases. Test cases are asymmetric: the service rates of the classes differ for each resource. Symmetric systems are not examined to emphasize the heterogeneity of the system. In the case of a homogenous system, we can see that any assignment of δ_{jk} in the allocation LP will lead to the maximum λ and assignment decisions will have less of an impact on overall system performance.

For each of the test cases, five loads between 0.8 and 0.99 of the maximum theoretical load the system can handle (i.e., λ^*) are simulated. At each load, 20 instances are tested for 10,000 time units to create a total of 400 simulations for each of the six models. Based on preliminary experiments, we choose to use $\alpha = 0.6$ for the *Tracking* model.

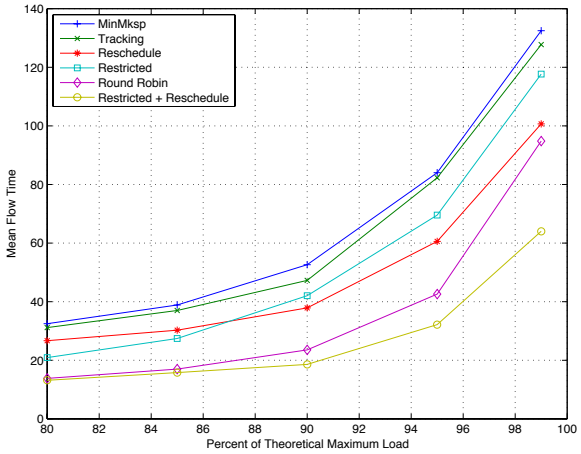

Figure 8: Comparison for a system with two resources and four job classes with long setup times.

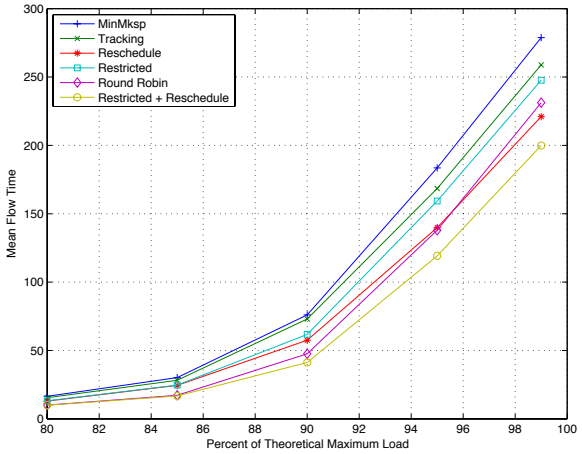

Figure 9: Comparison for a system with four resources and four job classes with short setup times.

Service times are exponentially distributed and arrivals follow a Poisson process. The service rates vary depending on class and resource pairs. The magnitude of service rates ranges between 1 and 9 jobs per time unit. To generate setup times, we randomly assign each class two positions on a Cartesian plane using a uniform distribution along the x and y axes. Given two coordinates (x_{k1}, y_{k1}) and (x_{k2}, y_{k2}) for each class k, the setup time from class a to class b is the straight line distance from (x_{a1}, y_{a1}) to (x_{b1}, y_{b1}). Sequencing classes in the reverse direction leads to a different setup time equal to the straight line distance between (x_{a2}, y_{a2}) and (x_{b2}, y_{b2}). Once a resource switches classes, new positions are assigned to calculate the setup times for the next setup occurrence. This approach maintains the triangle inequality when setup times are sequence dependent.

The simulation is written in C++ and run on an Intel Pentium 4 CPU 3.00GHz, 1 GB of main memory, running Red Hat 3.4.6-3. The MIP in LBBD is solved using IBM ILOG CPLEX 12.1 and the TSP solver is tsp_solve.[2]

Figures 7-10 show the performance of our policies for the four test cases. We see that *MinMksp* performs the worst for all systems under all loads. The *Tracking* model marginally improves over the *MinMksp* model. Surprisingly, the *Round Robin* policy, which is not designed with flow time in mind, exhibits very good flow time performance. In fact, while the *Restricted* and *Reschedule* models show improvement over *MinMksp*, they are both worse than *Round Robin*. This implies that following an efficient pairing of jobs and resources, according to the allocation LP, is important when scheduling jobs.

The *Restricted* + *Reschedule* model is the best performer overall. Other than in a few cases at very low loads where the *Round Robin* policy slightly outperforms the *Restricted* + *Reschedule* model, the lowest average flow time is achieved by *Restricted* + *Reschedule*. At high loads, the *Restricted* + *Reschedule* hybrid realizes mean flow time performance from 15% to 60% better than the next best algorithm. Under light system loads, for all but the smaller two-server system, the hybrid is able to outperform *Round Robin* by 0.2% to 5%.

[2]tsp_solve is a TSP solver in C++ available online at http://www.cs.sunysb.edu/∼algorith/implement/tsp/implement.shtml.

7 Discussion

Using the δ^*_{ik} values to guide the *MinMksp* model improves performance. The two models that directly make use of the δ^*_{ik} values are *Tracking* and *Restricted*. In both, we see lower mean flow time compared with *MinMksp* for all tested cases, though the gains observed for *Tracking* are marginal. Our intuition was that since the *Tracking* model more closely mimics the allocation LP solution, performance should be better than that of the *Restricted* model. We see that the opposite is true. We can understand these results by examining the impact of the setup times on the *Tracking* approach. Using the *Tracking* model, assignments where $\delta^*_{ik} = 0$ are still possible but costly due to setup times that are incurred over multiple scheduling periods. Since such assignments are expected to occur infrequently in *Tracking*, the number of jobs from class k assigned to resource i in a single period will be small. In an extreme, but not too uncommon, case where a single such assignment is made, two setups are incurred: one before the job and one after, with the second setup occurring in the next scheduling period as in Figure 6.a. If the setup times are an order of magnitude greater than the service times, the time used to serve this single job could have been more efficiently allocated to roughly twenty other jobs.

Turning to the *Restricted* and *Reschedule* models, we see that the former performs well at lower loads, while the latter is better at higher loads. This pattern occurs since at low loads, the scheduler is expected to make more inefficient assignments when there is naturally more idle time for resources. As a result, the *Restricted* model has more impact

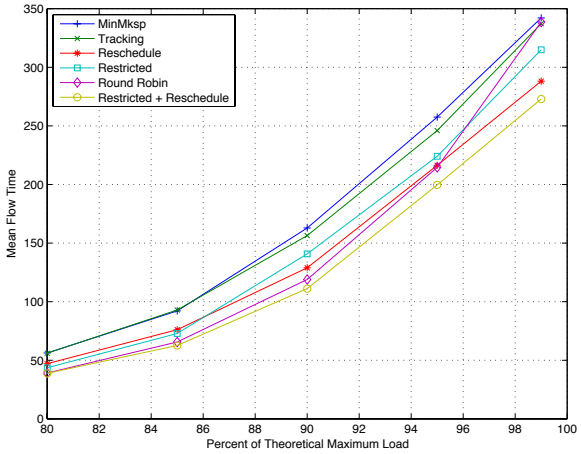

Figure 10: Comparison for a system with four resources and four job classes with long setup times.

at low loads. As the load increases, the scheduler has less slack and the restrictions have less impact as *MinMksp* naturally mimics δ^*_{ik}. The *Reschedule* model, in contrast, makes more changes to previous schedules at higher loads where there is a higher probability that jobs arrive during a period.

Combining the modifications in *Restricted* + *Reschedule* leads to the best performance. Alone, the *Restricted* model commits to more setup times than required because a schedule does not change when new jobs arrive. The *Reschedule* model may assign a job to a resource where $\delta^*_{ik} = 0$ and schedule it at the beginning of the period. Rescheduling prior to a setup time is then too late because the inefficiently assigned job, with $\delta^*_{ik} = 0$, has already been served and a setup is required to switch back to a more efficiently assigned job class. Therefore, the two models complement each other and deal with the dynamic properties of the system in different ways. The *Restricted* + *Reschedule* model is able to realize up to 60% better mean flow times than the next best approach at high system loads.

Finally, the performance of *Round Robin* comes as a surprise for two reasons, the relatively strong performance overall and the large drop in performance at high loads in Figure 10. The reason why the strong performance is not expected is because flow time is not considered as an objective and setup times are even ignored by assuming an infinite horizon. The fact that it is able to achieve performance better than all but one of our models suggests that a deeper investigation of its long- and short-term behaviour is merited. These results suggest that getting the long-run proportion of efficient allocations right is important for flow time performance as well. The performance drop at high loads for *Round Robin* in Figure 10 is surprising since we expect that if *Round Robin* were to perform well anywhere along the spectrum of system loads, it would be at higher loads. The allocation LP and calculation for l_{ik} in the *Round Robin* policy assume the system is at heavy loads and is designed to handle the system at these specific loads. One would believe that the fluid model should be most accurate as the load of the system approaches 1, but there does not seem to be any indication that the *Round Robin* policy is more accurate at higher loads as it is in fact one of the worst performers in Figure 10. This suggests that there are additional factors that have significant impact on flow time performance which the fluid solution does not capture. The loss of performance may also be due to the fact that *Round Robin* is a static policy only concerned with stability. As loads increase, it may be beneficial to make use of more state information to improve flow time performance.

An issue not addressed in our paper is the guarantee of stability in the system. Unlike the *Round Robin* policy, none of the other models guarantee that the system remains stable for any $\lambda < \lambda^*$. The stability conditions of a periodic scheduler in dynamic flow shop environments have been previously shown (Terekhov et al. 2012; Terekhov, Down, and Beck 2012). We expect that the models presented also stabilize the system when $\lambda < \lambda^*$, given that the experimental results exhibit stable behaviour for all models up to loads of 0.99. Further, one could intuitively see how minimizing makespan of a period will maximize throughput for the jobs belonging to the period. However, the interplay between maximizing throughput at a myopic scale and at the long-run system scale is not completely apparent. Thus, a formal proof is difficult and so we leave the stability conditions of our hybrid models for future work.

8 Conclusion

We studied a dynamic scheduling problem with sequence-dependent setup times. Building on recent work that seeks to combine queueing theory and scheduling approaches to address both stochastic and combinatorial challenges, four hybridizations of queueing and scheduling approaches were presented. Experimental evidence showed that each of the modifications improved on the standard scheduling model, but failed to perform as well as a simple policy from the queueing literature. We demonstrated that a combination of two hybridizations resulted in a model that achieved up to 60% better observed mean flow time performance than the pure scheduling or queueing-based policy. The strong performance of the queueing policy for an optimization criteria that it does not consider is surprising and deserves further research.

We believe that this paper contributes to ongoing research into the advantages of integrating techniques from queueing theory and scheduling to solve dynamic scheduling problems with complex combinatorics.

Acknowledgements

This research was supported by the Natural Sciences and Engineering Research Council of Canada (NSERC).

References

Al-Azzoni, I., and 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.

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

Arnaout, J. P.; Rabadi, G.; and Musa, R. 2010. A two-stage ant colony optimization algorithm to minimize the makespan on unrelated parallel machines with sequence-dependent setup times. *Journal of Intelligent Manufacturing* 21(6):693–701.

Baker, K. R. 1974. *Introduction to Sequencing and Scheduling*. John Wiley & Sons.

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.

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.

Gross, D., and Harris, C. 1998. *Fundamentals of Queueing Theory*. Wiley Interscience.

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

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

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

Sotskov, Y. N.; Sotskova, N. Y.; Lai, T.; and Werner, F. 2010. *Scheduling under uncertainty: Theory and algorithms*. Belorusskaya Nauka.

Terekhov, D.; Tran, T.; Down, D. G.; and Beck, J. C. 2012. Long-run stability in dynamic scheduling. In *Proceedings of the 22nd International Conference on Automated Planning and Scheduling*, 261–269.

Terekhov, D.; Down, D. G.; and Beck, J. C. 2012. Stability of a polling system with a flow-shop server. Technical Report MIE-OR-TR2012-01, Department of Mechanical and Industrial Engineering, University of Toronto. Available from http://www.mie.utoronto.ca/labs/ORTechReps/.

Tran, T. T., and Beck, J. C. 2012. Logic-based Benders decomposition for alternative resource scheduling with sequence dependent setups. In *Proceedings of the 20th European Conference on Artificial Intelligence (ECAI 2012)*, 774–779.