ROBUST SCHEDULES AND DISRUPTION MANAGEMENT FOR JOB SHOPS

by

Deepu Philip

A dissertation submitted in partial fulfillment
of the requirements for the degree

of

Doctor of Philosophy

in

Industrial Engineering

MONTANA STATE UNIVERSITY
Bozeman, Montana

April 2008

APPROVAL

of a dissertation submitted by

Deepu Philip

This dissertation has been read by each member of the thesis committee and has been found to be satisfactory regarding content, English usage, format, citation, bibliographic style, and consistency, and is ready for submission to the Division of Graduate Education.

Dr. Edward L. Mooney

Approved for the Department of Mechanical and Industrial Engineering

Dr. Chris Jenkins

Approved for the Division of Graduate Eduction

Dr. Carl A. Fox

STATEMENT OF PERMISSION TO USE

In presenting this dissertation in partial fulfillment of the requirements for a doctoral degree at Montana State University, I agree that the Library shall make it available for borrowers under the rules of the Library.

If I have indicated my intention to copyright this thesis by including a copyright notice page, copying is allowable only for scholarly purposes, consistent with "fair use" as prescribed in the U.S. Copyright Law.

Requests for permissions for extensive copying or reproduction of this dissertation in whole or in parts may be granted only by the copyright holder.

Deepu Philip

April 18, 2008

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ABSTRACT

This dissertation documents the results of research evaluating policies to schedule for unanticipated disruptions in job shops. The disruptions studied in this research are of two types - machine failure and job release-time. The study was conducted using modified classical job shop problems with the minimize maximum completion time (C_{max}) objective. Best random non-delay schedules (BRS) provided job sequences for each machine. Different slack policies based on frequency, duration and location of slacks in the schedule were used to strategically insert slack in the BRS schedule. The resulting robust schedules proactively managed machine or job release-time disruptions. The change in C_{max} quantified schedule robustness. Simulation was used to generate and modify the BRS schedules with and without slacks and disruptions, simulate disruption events and evaluate schedule performance. It was observed that policies that equally distribute slack to all tasks on heavily utilized machines and those that equally distribute slack to all tasks on all machines performed best. When the number of jobs to process was more than the number of machines and "big jobs" with long processing times on heavily utilized machines were present, the policy distributing slack tasks equally on heavily utilized machines exhibited superior performance. For systems with less variability both policies performed equally well. By comparing the average, minimum and maximum schedule deviations across all policies, the study concluded that strategically distributing slack to tasks on heavily utilized machines results in good robust schedules that can absorb the effects of disruptions.

CHAPTER 1

INTRODUCTION

Scheduling is the process of allocating limited resources to tasks over time to optimize one or more objectives (Baker, 1974). Resources might be machines or people in a shop, material handling systems, etc. Examples of tasks include basic machining operations, moving, transporting, loading, unloading and part programming. Tasks may have earliest start times and due dates and some tasks might have priority over others. Scheduling objectives include minimizing the completion time for a set of tasks (makespan), minimizing lateness, maximizing the number of tasks completed in a given time, minimizing work in process inventory, etc.

Shop scheduling problems usually have n jobs to be processed on m machines with linear task precedences that determine the task processing order within a job. Based on the processing order (or job operational flow), shop scheduling problems are classified into four categories:

1. Flowshop
2. Job shop
3. Dependent shop
4. Open or general shop.

All jobs have an identical processing order in flowshops. In job shop problems the task processing sequences are known for each job, but vary. Also the task sequences are independent of each other. Dependent shops have one or more jobs with the

task processing order dependent on other jobs. An open shop is characterized by the absence of any precedence constraints allowing tasks to be processed in any order.

Operations in industries such as aerospace, heavy machinery manufacture, food processing etc. can often be modeled quite well as a job shop problem. Figure 1 is a schematic representation of a job shop with three different jobs and six departments, or possible job steps.

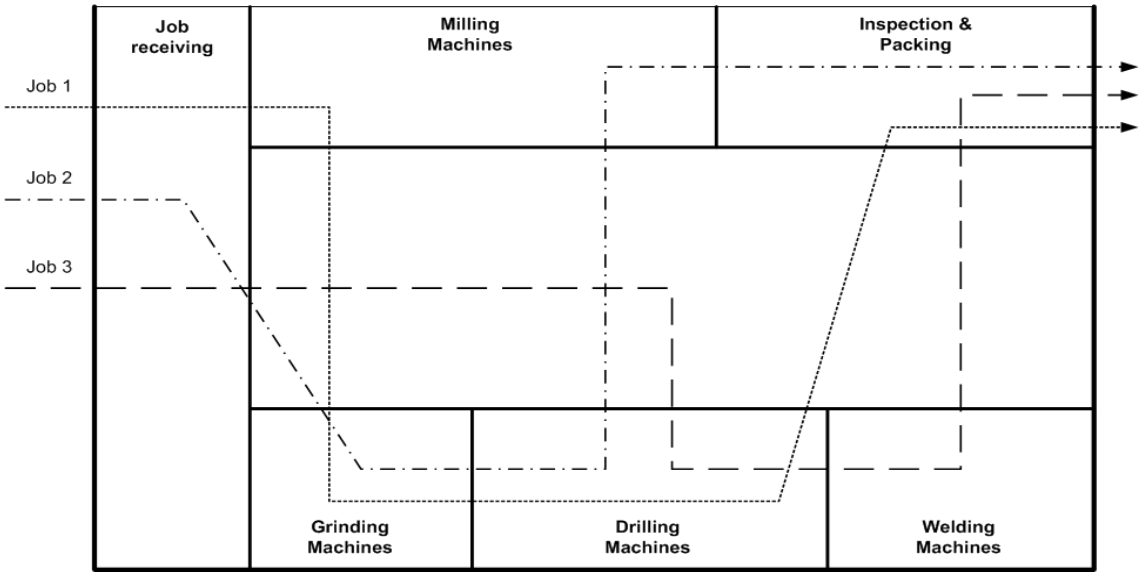

Figure 1: Schematic diagram of a job shop

Job 1 is routed to milling, grinding, drilling, and inspection departments whereas job 2 consists of grinding, drilling, milling and inspection tasks. Job 3 visits drilling, welding and inspection centers before completion. Feasible schedules for a job shop are obtained by satisfying resource and task precedence requirements. Brucker (1998) defined this scheduling approach *off-line scheduling*. The usual off-line scheduling approaches try to pick a schedule from the set of feasible schedules that results in

the best performance for a specific objective, or effectiveness measure. A common objective in job shop problems is minimizing maximum completion time (C_{max}).

However, maximizing profit or resource utilization may result in optimized schedules that fail to accommodate unpredictable events, or *disruptions* (Hopp and Spearman, 2000). In a manufacturing environment, disruptions may be caused by machine breakdowns, raw material shortages, worker unavailability, unanticipated demand changes, rework or quality problems, due date changes, changes in job priority or change in processing times. Disruption Management (DM) refers to strategies for managing the effects of disruptions on a schedule. As shown in Figure 2, disruption management strategies are typically classified as *proactive* or *reactive*.

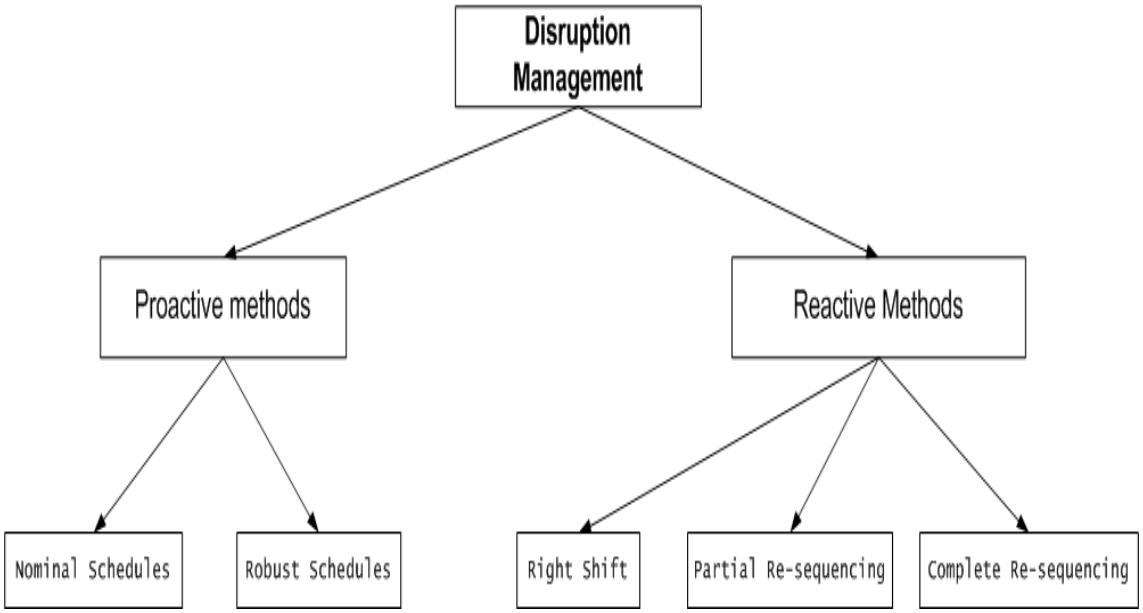

Figure 2: Disruption Management Framework

Proactive disruption management attempts to construct robust schedules that can absorb most of the disruptions, avoiding the need to re-sequence after a disruption. Nominal schedules are baseline schedules obtained with some accidental slack built into it due to the task sequence. Robust schedules modify a good initial schedule by adding intentional slack within the schedule in order to absorb the effects of disruptions. In contrast to proactive disruption management, reactive disruption management repairs the original schedule after the occurrence of disruption (Smith, 1994) while minimizing changes to task sequence and performance measures. Right shift, partial or complete re-sequencing are often used for schedule repair. The right shift method preserves the job sequence, where as the partial and complete re-sequencing methods modify the original job sequence.

This study explored the use of proactive scheduling methods to handle disruptions in job shop environments. Robust schedules were obtained by introducing slack in an optimized schedule according to a *slack policy*, specifying slack and its location in the schedule. The schedules generated by different slack policies were subjected to two main types of random disruptions:

1. Machine breakdowns
2. Job release-time delays.

The policies were evaluated for each disruption type using modified classical job shop problems. The majority of the current research is limited to single breakdown events whereas this study allowed for multiple disruptions to a schedule. Results from the

two studies for the single disruption types were also tested for job shops with multiple disruption types.

Simulation was used to evaluate the effectiveness of the robust schedules obtained with five slack policies for managing the disruption types. Slack was inserted into best random non-delay schedules (BRS) described by Mooney and Philip (2006) to generate the robust schedules used in the study. The robustness of a schedule was quantified by the change in the maximum completion time, C_{max}, when the robust schedule was subjected to disruptions. It was found that slack policies were able to generate robust schedules to handle disruptions.

This chapter briefly introduced the research problem, goals and methodology. The remainder of this dissertation is organized as follows. Chapter 2 explains the concept of machine-robust schedules and how they can be used to proactively manage machine disruptions. Chapter 3 studies the effectiveness of job release-time robust schedules to manage job release-time disruptions. Chapter 4 utilizes the results from chapters 2 and 3 to develop and test practical guidelines for using slack policies to handle both machine and job release-time disruptions. Chapter 5 summarizes the entire study and the main conclusions.

CHAPTER 2

SLACK POLICIES FOR JOBSHOP MACHINE DISRUPTIONS

Abstract : Machine breakdowns can create chaos in shops whose schedules maximize machine utilization. Introducing slack can make the schedule more robust, but at a cost. This paper evaluates the effects of slack policies on the machine-robustness of C_{max} job shop problems. Slack policies are specified by the frequency, duration and location of slack in the schedule. Policies are evaluated using a standard set of job shop problems and the right shift rescheduling algorithm.

Keywords : disruption, robust schedule, slack, shop scheduling, slack policies, simulation, job shops.

Introduction

Businesses expend resources and efforts generating schedules prior to starting activities based on demand, competition, available resources and operational requirements to maximize profits. The method of developing such schedules is known as off-line scheduling or off-line optimal scheduling (Brucker, 1998). Off-line optimal scheduling is used in many industries including:

- Manufacturing - shop scheduling
- Service - manpower scheduling
- Construction - project scheduling.

Optimal schedules minimize slack in the schedule to maximize resource utilization and thus profit. The deterministic optimal scheduling approach assumes that there is no variability or uncertainty associated with the system. In the real world, however, all systems have uncertainty and variation associated with them.

Disruptions are unpredictable events that occur due to variability in the system (Hopp and Spearman, 2000). Thus, disruptions unavoidably bring changes to the environment of the operation assumed for off-line optimal scheduling. Outcomes of disruptions to optimal schedules include revenue loss due to missed delivery dates, loss of customer good will and the cost of re-scheduling (Baker and Scudder, 1990). Kandasamy et al. (2004) showed that when an off-line optimal schedule is implemented in a dynamic and unpredictable environment, the difference between expected performance and actual performance is often high because the off-line schedule cannot accommodate the uncertainties during execution.

Since many manufacturing facilities can be modeled as a job shop (Sule, 1997), this study focused on job shop problems. The proactive scheduling method used in this study is the application of robust schedules to handle machine disruptions. This study presents a new approach for developing various slack policies to generate robust schedules. Also, the effectiveness of these slack policies in handling machine breakdowns is evaluated by right-shifting tasks when necessary.

The general approach used in this research can be summarized in four steps:

1. Extend the SIMPL simulation library developed by Mooney (2005) to simulate

a job shop with a predetermined sequence, handle disruption events and collect statistics.

2. Adapt the classical job shop problems used by Applegate and Cook (1991) to include disruption events.

3. Develop robust schedules using various slack policies to handle machine disruptions.

4. Analyze the effectiveness of the slack policies with simulation experiments.

The rest of this paper is organized as follows. The next section discusses the current research and practices in disruption management using reactive and proactive methods. The slack policies and test problems used in the study are described next. We conclude with the results of the experiments and general findings of the research.

Related Work

Recent research on techniques for handling schedule disruptions involves either *proactive* or *reactive* methods (Vieira G. and Lin, 2003). Early disruption management studies focused on reactive disruption management by rescheduling following a disruption. Bean et al. (1991) pioneered work in reactive disruption management, establishing the importance of schedule deviation costs. The authors proposed the concept of match-up scheduling where the goal was to develop a new schedule with a finite recovery time after disruption. Akturk and Gorgulu (1999) extended

the match-up schedule approach to a modified flow shop environment. Mason et al. (2004) studied various rescheduling strategies for minimizing total weighted tardiness in the semiconductor manufacturing industry. The authors found that rescheduling strategies can minimize the effect of disruptions when setup costs are not present.

The airline industry is perhaps the largest implementor of reactive disruption management techniques to recover from schedule disruptions. Teodorovic and Guberinic (1984) used the re-assignment method to handle small perturbations in airline schedules after a disruption. Clauser et al. (2001) studied reactive disruption management in project scheduling in the ship building industry. The authors found that proper rescheduling methods can help in speedy recovery of the original schedule.

Zhu et al. (2005) proposed a mixed integer programming/constraint propagation procedure to solve the general project schedule recovery problem. The authors provided a formal definition of disruption in project scheduling as the stage at which the differences between planned and actual costs, activity durations and resource requirements were noticeable. The authors used the recovery problem approach proposed by Eden et al. (2000), where the aim was to minimize the deviation from the original schedule and the objective was to minimize the project completion time, or makespan.

Kohl et al. (1999) studied the expensive tools developed to generate optimal schedules for various industries and documented the difficulty of implementing the optimal schedules on a daily basis. Their study focused on the airline industry, where airlines attempted to develop an optimal flight schedule with very little slack to

accommodate disruptions. The authors also investigated the disruptions that occur in the airline industry due to resource breakdowns and the *ripple-down effect* (change in the time of scheduled operations following a disruption event) in a well-optimized schedule. Dorndorf et al. (2007) also studied the same phenomena for the flight gate scheduling problem and named it the *knock-on effect*. The authors identified that multiple disruptions results in a larger ripple-down or knock-on effect in a schedule.

Yang and Yu (2005) studied disruptions due to various internal and external factors in a typical production environment. The major factors identified were machine breakdowns, raw materials availability, raw material quality, shipment delays, unanticipated employee absences, stochastic demand fluctuations, power outages, environmental factors, and weather. The authors observed that when a heavily optimized plan was used, the proper and timely management of disruptions was almost impossible without rescheduling the entire shop.

Proactive disruption management is an elegant alternative to reactive methods. One of the first proactive methods was the use of probabilistic models to deal with disruptions. Birge and Louveaux (1997) proposed an anticipative approach along with the application of stochastic programming to plan for disruptions. Porteus (1990) and Putterman (1990) proposed Markov decision techniques to obtain contingency plans to accommodate unplanned disruptions. Putterman (1990) also showed that the contingency plans derived by the Markov decision process are not very effective in dealing with completely random disruptions. Kouvelis and Yu (1997) proposed

the *robust optimization* approach where the worst case production plan was optimized. The authors also noted that the resulting solutions were too conservative to be practical for many applications.

Sorensen and Sevaux (2003) proposed a meta-heuristic optimization with Monte Carlo sampling of stochastic parameters that control disruptions in the stochastic vehicle routing problem. Lambert et al. (1993) developed stochastic vehicle routing formulations where fluctuations due to uncertainty were handled using stochastic travel times. Their findings suggested that the two methods provide some robustness to handle these disruptions in the final vehicle routing.

Mehta and Uzsoy (1999) studied predictive scheduling methods for handling single machine breakdowns. The authors extended the study to job shop problems with machine breakdowns. Igal et al. (1989) studied the time to repair and time between failures for a single machine scheduling problem. The authors identified that more the variation associated with the disruption event, the harder it is to develop robust schedules to manage the disruption.

There has been some preliminary work on slack-based robust schedule generation. Natale and Stankovic (1994) first proposed the widely used method of adding equal slack to all available tasks. Kandasamy et al. (2004) extended this equal slack-based approach to handle disruptions in the task scheduling problem. The authors distribute the slack equally among the tasks while trying to satisfy the task timing requirements. This method, known as *temporal flexibility*, aims to absorb disruptions

due to the uncertainties in task execution times.

Davenport et al. (2001) proposed a method of introducing slack by adding the expected repair time of the resource used by a task to the execution time of that task. Gao (1995) proposed a similar approach where slack was built into the processing time of each task. Both authors then used standard techniques to generate robust schedules at the expense of makespan and due dates.

It is evident from the discussion above that many robust scheduling approaches have been studied in various problem domains. However, the literature reveals little work on proactive slack-based robust scheduling for job shops. The next section defines the slack policies for generating job shop schedules that are robust with respect to common machine disruptions such as breakdowns. Such schedules are called *machine-robust schedules*.

Slack Policies

According to Kouvelis and Yu (1997) the best robust schedule is the schedule with the best worst case performance. Dorndorf et al. (2007) argue that the best robust schedules are sub-optimal schedules where no rescheduling is necessary when a disruption occurs. However, a schedule incorporating a lot of slack is not necessarily very robust (Kouvelis and Yu, 1997), because the availability of slack by the disruption event is also as important as the amount of slack. Hence, we propose strategically distributing slack within a good sub-optimal schedule to absorb disruptions.

Distributing slack in a schedule is comparable to leaving *holes* in a facility layout, where extra empty space is left to accommodate future growth or changes in department sizes with respect to layout. The effect of department changes on layout is equivalent to disruption effects on a schedule. If a facility has just enough space to accommodate its employees (equivalent of optimal schedule), any future expansion or staffing changes will necessitate a reprogramming of a large amount of space.

As mentioned in the previous section, machine-robust schedules are used in this study to manage machine disruptions. To obtain a machine-robust schedule we specify the distribution of slack in the schedule with a *slack policy*. Slack policies specify the number, length and placement of dummy, or slack, tasks in the schedule with three slack policy parameters:

1. Frequency - number of slack tasks (pseudo tasks) inserted in the schedule,

2. Duration - length of individual slack tasks,

3. Location - distribution of slack tasks in the schedule by machine.

Slack policies used in this study were based on previous studies conducted by Kandasamy et al. (2004) for task scheduling problems, Mehta and Uzsoy (1998) and Davenport et al. for job shop problems (2001). Davenport et al. (2001) showed that the frequency and duration of the slack tasks should be a function of the Mean Time Between Failures (MTBF) and Mean Time To Repair (MTTR) of machines for general production systems. The location of the slack is also an important parameter

that has not been significantly studied.

Slack frequency, duration and location are determined indirectly by specifying total slack and selecting one or more machines (along with tasks on them) for slack insertion. The total slack is set equal to the expected total downtime for machine disruptions, i.e. $E[\text{Total Slack}] = \frac{1}{MTBF} * E[MTTR]$. This total slack time is then distributed equally to the selected subset of machines (location) and their tasks by placing dummy slack tasks ahead of each task in the sequence. Table 1 summarizes the subset of machines chosen (location) for each slack policy.

Table 1: Location values for slack policies for machine disruptions

Policy	Machine selection for slack distribution
P1	One random machine from the set of machines
P2	On all machines
P3	The most heavily utilized machine
P4	On one of the heavily utilized (near-bottleneck) machines
P5	On all heavily utilized (near-bottleneck) machines

Policy P1 selects a machine at random from the set of machines, whereas policy P4 selects one machine randomly from the set of heavily utilized machines. Policy P2 distributes slack on all machines, whereas policy P5 distributes slack to all heavily utilized machines. Policy P3 distributes slack only to the most heavily utilized machine. In general, the more machines chosen, the higher the slack frequency.

Sufficient slack at the time of a disruption can cushion the schedule from the disruption, minimizing the ripple-down or knock-on effects. The right shift approach by Kohl et al. (1999) was used to evaluate the effectiveness of the slack policies.

Experiments comparing slack policies for classic job shop problems allowed to evaluate the importance of policy parameters.

Test Problems

Experiments for this study applied the policies discussed in the previous section to the classical job shop test problems used by Applegate and Cook (1991), modified to include machine breakdowns. These well-known problems are from several sources:

1. Adams et al. (1998) - 'abz' problems,
2. Carlier and Pinson (1989) - 'car' problems,
3. Muth and Thompson (1991) - 'mt' problems,
4. Bonn (1991) - 'orb' problems,
5. Lawrence et al. (1991) - 'la' problems.

Each problem case considered in this study is specified by the class (author) and the base instance with the disruption pattern added to it. Each base instance has a specified number of jobs (N), number of machines (M), processing time durations (p_{ij}'s) and routes, specified by the original problem. To these problem instance parameters, we added the disruption parameters Mean Time Between Failure (MTBF), Mean Time To Repair (MTTR), Location (Machine) of Disruption (LOD) and Slack Ratio (SR). The goal of this study is to find significant interactions between important problem factors and policy factors.

There are seven parameters that are applicable to a particular problem instance. All the seven parameters were not varied in the study. One parameter was eliminated and one parameter was fixed and the remaining five were varied as factors in the experiment. The experiment was conducted as a 2^k factorial experiment. First we discuss the eliminated and fixed parameters before discussing those treated as experimental factors.

Based on previous studies by Davenport et al. (2001), Kandasamy et al. (2004) and (Gao, 1995) on slack-based robust schedules in disruption management, the mean processing time parameter ($\bar{P}$) was eliminated. It is generally accepted that disruption events are not related to mean processing times (Davenport et al. (2001)). The processing times (and $\bar{P}$) for the test problems are defined by the instances which we did not vary.

The MTBF and MTTR defines the disruption pattern for a problem, where $MTBF$ determines the expected number of breakdowns in a given schedule and MTTR determines the duration of the breakdowns. The Time Between Failures (TBF) is a random variable in this study. Previous studies of machine disruptions on the single machine problem by Igal et.al (1989) used a single disruption event with a constant time between failure. Mehta and Uzsoy (1998) also studied the effect of a single disruption event on job shops. They concluded that the time between failures in job shops can be best modeled using an exponential distribution with a mean equal to the MTBF and suggested using MTBF around 40% of the maximum of the total

processing times for individual jobs to ensure at least one machine disruption event in the schedule. Hence, for this research MTBF was set at 40% of the maximum of the total processing times for individual jobs.

The problem parameters kept as experimental factors used in this study are listed in Table 2. Factors one through five were considered by different authors in previous

Table 2: Job shop machine disruption study factor list

Parameter	Parameter Name	Description
1	N	Number of jobs
2	M	Number of machines
3	SR	Ratio of total slack to total processing time
4	MTTR	Mean Time To Repair
5	LOD	Location Of Disruption

studies as important factors in different problem scenarios. Preliminary analysis showed that these five parameters do have the capability to influence the robustness of a particular machine-robust schedule. Hence, all the five were considered as factors for the 2^k factorial experiments. The levels to all of these factors are discussed below.

Factors N and M determine the problem size. The levels of N and M were chosen by the Median Average Technique (MAT) (Montgomery, 2001). Compared to the conventional mean averaging method, the median average is less sensitive to noise created by outlying values in the data set (Devore, 2004). Hence, the MAT method is found useful when the values in the data set of the parameters are unevenly distributed (Schillings, 2008).

The possible values are $N = \{6, 7, 10, 11, 13, 15, 20\}$ across all five job shop problem classes. By MAT, the averaged median becomes 11.7 and the closest value to the median average is chosen as the splitting point, because of the odd number of possible values. Hence the high and low values chosen for factor N were $N \leq 11(-)$ and $N \geq 12(+)$.

Similarly the possible values of M are $M = \{4, 5, 6, 7, 10, 15\}$, across all five job shop problem classes. The MAT gives a value of 6.5, resulting in an equal split of the data points, because of the even number of possible values. The high and low values for factor M were $M \leq 6(-)$ and $M \geq 7(+)$.

The levels for the SR were varied based on results from previous studies. In the studies of Davenport et al. (2001) and Kandasamy et al. (2004) the total duration of slack was considered to be an important factor. Also Dorndorf et al. (2007) used the ratio of available slack in an airline schedule to the schedule makespan to measure the robustness of the given schedule. The SR factor normalizes the introduced slack for comparison between instances. Hence, the levels of factor SR was set to $SR \approx 25\%(-)$ and $SR \approx 50\%(+)$.

Repair or disruption recovery times were sampled from an exponential distribution. With $MTTR$ factor set as suggested by Igal et al. (1989), repair times were assumed to be independent of time between failures for the single machine scheduling problem.

McKay et al. (1989) used a uniform distribution to sample the repair times. Yang and Yu (2005) also argued that even though random values for time to repair models the worst case repair time variability, in most situations the MTTR can be modeled within certain limits. Davenport et al. (2001) argued that the time it takes to repair the machine after a breakdown in comparison with the average processing time is what is important. The same hypothesis was reached during discussions with the committee chair (Mooney, 2007). Hence, for simulating the machine repair times, we sampled the time to repair from an exponential distribution whose mean was obtained from two different uniform distributions (McKay K. N. and Safayeni, 1989), (Yang J. and Yu, 2005). The levels for the factor MTTR were set at $MTTR = UNIFORM(0.25 * \bar{P}, 0.45 * \bar{P})(-)$ and $MTTR = UNIFORM(0.25 * \bar{P}, 0.45 * \bar{P})(+)$ where $\bar{P}$ denoted the mean processing time for the individual problem instance.

The *LOD* factor determines which machine breaks down during the schedule execution. In single machine problems this factor is not considered as there is only one resource available. Studies in semiconductor manufacturing by Barahona et.al (2005) modeled the breakdown of the most heavily utilized resource. Mehta and Uzsoy (1998) studied the breakdown of any machine, but limited themselves to single breakdown event during a schedule. Kouvelis and Yu (1997) argued that the breakdown of the bottleneck machine represents the worst case scenario and the best robust schedule has the best worst case performance. Since this study aims to evaluate the effect of multiple machine breakdowns on robust schedules, the levels for the location

of disruption factor were set to *Any machine other than most heavily utilized (-)* and *Most heavily utilized machine (+)*.

Table 3 summarizes the levels for all problem factors used in this study. Using

Table 3: Factors and levels for job shop machine disruption study

Factor	Factor	Low (-)	High (+)
1	N	$N \leq 11$	$N \geq 12$
2	M	$M \leq 6$	$M \geq 7$
3	SR	$SR \approx 25\%$	$SR \approx 50\%$
4	MTTR	$MTTR = UNIFORM(0.25 * \bar{P}, 0.45 * \bar{P})$	$MTTR = UNIFORM(0.25 * \bar{P}, 0.45 * \bar{P})$
5	LOD	Any machine other than most heavily utilized	Most heavily utilized machine

these factors and levels, a factorial analysis was conducted to identify the significant factors with respect to machine-robust schedules in handling machine disruptions. Every problem case evaluated in the study was replicated twice by using two different random number seeds to get two different disruption patterns. The next section of this document summarizes the experimental results and analysis.

Experimental Results

Having defined the policies and problem factors for the computational experiment in the previous section, this section describes the experimental methods and results. As mentioned in the introduction, the study used a general purpose simulation to generate robust schedules, collect statistics collection and compute the makespan for the problem and policies. All cases evaluated slack-based proactive disruption management, and there was no re-sequencing after a machine disruption. Feasibility was maintained by right-shifting tasks as proposed by Kohl et al. (1999) to evaluate the

effectiveness of the slack policies. Right shifting quantified the amount of disruption absorbed by the strategically distributed slacks in the schedule. If the slack policy results in a robust schedule then the net ripple-down effect as measured by the change in makespan is equal to zero. Any increase in the schedule's C_{max} implies the failure of the slack policy to fully accommodate the machine disruptions.

Experiments were run using a three step procedure for each problem as shown in Figure 3. First, the simulation was used to obtain a baseline schedule without

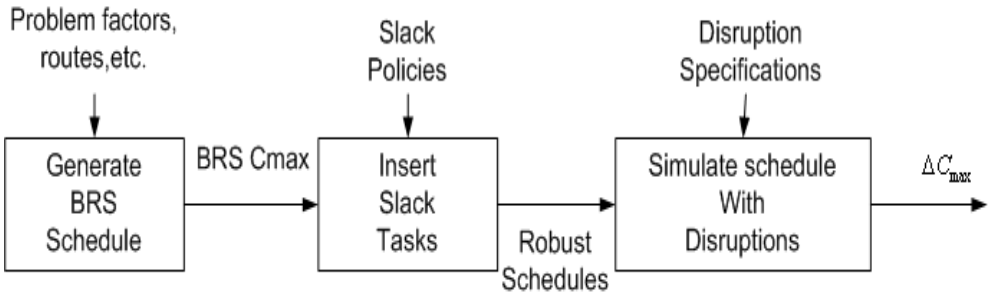

Figure 3: Experimentation process for machine disruptions

disruptions. All problems were scheduled initially without slack or disruptions using Best Random non-delay Schedule (BRS) approach proposed by Mooney and Philip (2006). The authors showed that the algorithm exhibits excellent performance in randomly generated job shop test problems with makespan consistently with 5% of the best known value. Similar results have been shown by (Godard D. and Nuijten, 2005) using randomized large neighborhood search. The BRS approach generates a set of random non-delay schedules and picks the best schedule from them. The number of random non-delay schedules generated by BRS approach was 100 for this

study.

The second step was to modify the BRS schedule by inserting slack according to each of the five slack policies. The result was a total of six schedules derived from a common sequencing of jobs on machines. Some of the policies involved random sampling and we did not replicate the sampling for those policies. Davenport et al. (2001) also used random policies without replicating the sampling process to be consistent with the other single pass slack policies like P2, P4 and P5. The sub-optimal best random non-delay schedules with slack tasks were stored in order to preserve the job sequence and slack tasks. Generating the schedules before including disruption events minimizes the confounding effects of disruption events and the BRS schedule generation.

The third step simulates the saved schedule with disruption events based on MTBF, MTTR and machine. The performance of each of these schedules is measured by the change in C_{max}. Like all stochastic simulation packages, random variates are sampled from a stream of random numbers to generate a realization of the system state over time. Thus each of the random disruption patterns were obtained using two different random number seeds, resulting in two replications for each problem case.

A total of 6 schedules for each of the 32 problems, replicated for two disruption patterns required a total of 384 simulation runs of the experiment. The runs were made on a Pentium IV 3.06 GHz dual processor machine with four gigabytes of RAM,

running the Lineox operating system. C++ was used as the programming language with the G++ compiler. The results for each problem class follow.

Tables 4 through 8 summarize the experimental results for each problem class. In each table, the first five *Problem Factors* columns show the levels of the problem factors listed in Table 3. The *Rep* column under the problem factors represents the replication number of the problem instance. The next column titled $BRS\,C_{max}$ shows the baseline value of C_{max} using the BRS algorithm (2005) without disruption or slack. The number of machine disruption events that occurred in the run are listed in the column titled *# of MDs*. The next five columns under *Slack Policies* show the percentage change in C_{max}, $100(\frac{\Delta_{C_{max}}}{BRSC_{max}})$, for each policy for each run. Note that "robustness" can be quantified by the equation $100(1 - \frac{\Delta_{C_{max}}}{BRSC_{max}})$, or a 100% robust schedule has $\Delta C_{max} = 0$. The bottom row of each table summarizes the total number of times the particular slack policy resulted in no change in C_{max} across the problem class.

Table 4 summarizes the results for the 10 job, 10 machine 'abz' problems. Slack policy P2 has the most robust schedules for the 10x10 problems, followed by P5, P4, P3 in the descending order. Slack policy P1, where slack is randomly distributed performs the worst. This result supports the argument of Kandasamy et al. (2004) that distributing slack in the schedule in some intelligent way always performs better than pure random distribution of slack.

Table 4: Machine disruptions C_{max} change for 10x10 'abz' problems

Prob.	Problem Factors						BRS	# of	Slack Policies				
Name	N	M	SR	MTTR	LOD	Rep	C_{max}	MDs	P1	P2	P3	P4	P5
abz5	+	+	-	-	-	1	1251	2	4.5	0	1.2	0	2.1
						2	1251	2	3.7	0	1.9	0.8	1.7
					+	1	1264	1	0	0.3	0	0	0.8
						2	1264	3	8.6	1.5	2.4	0	1.3
				+	-	1	1255	2	6.3	0.5	0	0.9	0
						2	1255	2	0	0	0	1.4	0.5
					+	1	1259	3	5.4	2.1	1.8	2.2	0.3
						2	1259	2	9.6	0.2	0.4	1.7	0.6
			+	-	-	1	1273	3	5.7	1.6	2.2	3.1	2.6
						2	1273	1	1.4	0	0	0.1	0
					+	1	1258	2	1.4	0	0.3	0	0
						2	1258	2	3.9	0	0	0.4	0
				+	-	1	1252	3	1.3	0.1	1.4	0.2	1.2
						2	1252	3	2.4	0.2	0.9	0.3	0.7
					+	1	1281	2	0	0	0	0.2	0
						2	1281	2	0	0	0.1	0	0
abz6	+	+	-	-	-	1	974	2	2.6	0	0.2	0	0.2
						2	974	2	3.4	0.2	0	0.1	0
					+	1	977	2	1.3	0.7	1.3	0.3	0
						2	977	2	2.9	0	1.1	0.2	0.1
				+	-	1	974	2	5.6	0	0.4	0.5	0
						2	974	2	0	0.1	0	0.1	0
					+	1	976	2	8.1	0	2.2	0	0.1
						2	976	3	1.4	0.3	1.6	0.6	0.2
			+	-	-	1	977	2	0.9	0	0.1	0	0
						2	977	2	3.3	0	0.4	0.1	0
					+	1	983	1	2.1	0	0	0	0
						2	983	3	4.5	0	0.2	0.2	0.3
				+	-	1	979	2	6.7	0	0.1	0	0
						2	979	2	0.8	0	0	0	0.1
					+	1	988	3	1.6	0.2	0.2	0.4	0.7
						2	988	2	0.2	0	0	0	0
Total number of robust schedules									5	19	10	12	15

Table 5 summarizes the results for the 20 job, 15 machine 'abz' problems. From the total number of robust schedules under different slack policies, it is quite evident that policy P5 outperforms all other slack policies. The order of performance is P5, P2, P4, P3 and P1. As expected the random slack policy P1 performs the worst. The 20x15 class of problems have more jobs than machines, resulting more machines to fall under the heavily utilized (near-bottleneck) machine category. Since there are more tasks available in this problem, the slack duration per task under policy P2 is reduced more than the slack duration per task under policy P5, which allocates slack only on heavily utilized machines. Hence policy P5 outperforms policy P2, when the number of jobs are more than the number of machines.

Table 5: Machine disruptions C_{max} change for 20x15 'abz' problems

Prob.	Problem Factors						BRS	# of	Slack Policies				
Name	N	M	SR	MTTR	LOD	Rep	C_{max}	MDs	P1	P2	P3	P4	P5
abz8	+	+	-	-	-	1	693	2	3.6	0.4	1.6	0.1	0
						2	693	2	0.7	0.2	1.1	0.8	0.3
					+	1	704	2	2.1	0	0	0	0
						2	704	2	0.4	0.1	0.7	0.4	0.4
				+	-	1	712	3	6.7	1.3	1.9	1.1	0
						2	712	2	0	0	0	0	0
					+	1	697	2	5.9	0.5	0.9	0.7	0.1
						2	697	3	9.3	1.7	2.1	1.5	1.6
			+	-	-	1	699	3	11.2	0	0.4	0	0
						2	699	2	4.7	0	0	0.1	0
					+	1	696	2	6.4	0.4	0.8	0	0
						2	696	2	2.1	0	0	0.2	0
				+	-	1	708	2	0.3	0.2	0	0	0
						2	708	2	0	0.1	0.9	0.6	0.1
					+	1	712	2	2.9	0	1.7	0.8	0
						2	712	2	5.5	0	0.5	0	0
abz9	+	+	-	-	-	1	723	2	4.7	1.5	2.1	1.8	0.7
						2	723	2	2.1	0	0.4	0.2	0.4
					+	1	717	2	5.9	0.2	0.9	0.3	0.2
						2	717	3	9.7	1.9	1.7	1.1	1.1
				+	-	1	719	2	2.4	0	0	0	0
						2	719	2	3.3	0.2	0.6	0.4	0
					+	1	715	2	0.2	0	0.5	0.3	0.3
						2	715	3	1.7	1.6	0	2.4	0.9
			+	-	-	1	731	4	10.6	2.8	3.4	3.1	1.6
						2	731	2	0	0	0.4	0	0
					+	1	724	2	0.2	0.1	0.8	0	0
						2	724	3	6.5	0.7	1.2	0.6	0.2
				+	-	1	719	2	7.8	0	0.7	0.4	0
						2	719	2	0	0	0.3	0	0
					+	1	717	1	1.4	0	0	0	0
						2	717	3	8.6	0.2	0.4	0.7	0.3
Total number of robust schedules									4	14	8	11	17

Table 6 summarizes the results for 'car' class of job shop problems that are well known for their large processing times combined with a smaller number of machines available to process all the jobs. These problems are considered quite hard in the job shop scheduling literature due to the lack of structure and the variability in processing times. Again policy P5 is the winner here, followed by P2, P4, P3 and P1. It is worth noting that the policy P1 was unable to generate a single robust schedule. The difference in the number of robust schedules generated between policies P5 and P2 shows that when there is more variability in the system and more jobs to process on fewer machines, distributing slack on heavily utilized machines provides a superior solution.

Table 6: Machine disruptions C_{max} change for 11x5 & 10x6 'car' problems

Prob.	Problem Factors						BRS	# of	Slack Policies				
Name	N	M	SR	MTTR	LOD	Rep	C_{max}	MDs	P1	P2	P3	P4	P5
car1	-	-	-	-	-	1	7053	2	6.2	0.7	2.1	0.9	0.1
						2	7053	3	9.4	1.2	2.7	2.2	0.6
					+	1	7048	2	5.3	0	1.8	1.1	0
						2	7048	2	4.1	0.3	2.5	0.8	0.1
				+	-	1	7065	3	10.6	1.1	1.6	1.4	0.3
						2	7065	3	9.8	0.9	1.9	1.5	0.2
					+	1	7077	2	2.7	0.2	0.9	0.9	0.1
						2	7077	2	6.9	0.4	1.7	1.1	0.2
			+	-	-	1	7059	3	11.3	0.3	1.7	0.7	0
						2	7059	2	6.5	0	0.2	0	0
					+	1	7054	3	7.2	0.2	0.7	0.3	0
						2	7054	3	8.4	0.6	1.6	1.1	0.2
				+	-	1	7071	2	3.1	0	0	0	0
						2	7071	2	1.6	0	0	0	0
					+	1	7057	3	8.7	0.7	1.9	1.6	0
						2	7057	2	4.3	0	0.3	0.1	0
car5	-	-	-	-	-	1	7735	3	9.8	0.5	2.4	2.1	0.3
						2	7735	2	6.1	0.3	2.1	1.4	0
					+	1	7741	3	8.6	1.4	3.6	1.9	0.7
						2	7741	3	11.1	1.8	3.1	1.5	0.6
				+	-	1	7756	2	4.8	0	0	0	0
						2	7756	4	15.3	1.6	2.4	1.8	0.4
					+	1	7751	3	9.7	0.2	2.9	0.7	0.2
						2	7751	3	8.5	0.7	1.5	1.2	0.6
			+	-	-	1	7745	4	12.4	0.3	1.8	0.9	0.1
						2	7745	3	10.3	0	2.1	0.5	0
					+	1	7736	2	6.6	0	0	0	0
						2	7736	4	15.7	0.3	0.9	0.7	0.2
				+	-	1	7752	2	6.5	0	0.6	0.2	0
						2	7752	4	11.9	1.1	2.1	1.4	0.1
					+	1	7761	3	2.4	0.2	0.9	0.3	0
						2	7761	3	6.3	0.2	1.3	0.7	0
Total number of robust schedules									0	8	4	5	14

Experiments conducted on the 'mt' class of structured job shop problems generated by Muth and Thompson are summarized in Table 7. Once again, the best policy is P5, followed by P2, P4, P3 and P1. It seems that policies P5 and P2 perform very close to each other when there is less variability (or more structure) in the system, as seen in the 'mt06' class of problems. But policy P5 outperforms P2 in the 'mt20' class problems, where there are 20 jobs to be processed on 5 machines. Policies P4 and P3 exhibits mediocre performance across all problem instances.

Results for the 'orb' class of job shop problems generated by Bonn et al. are summarized in Table 8. The 'orb' class is well known for its high ratio of jobs to be processed on each machine. Policy P5 produces more robust schedules for this class

Table 7: Machine disruptions C_{max} change for 6x6 & 20x5 'mt' problems

Prob.	Problem Factors						BRS	# of	Slack Policies				
Name	N	M	SR	MTTR	LOD	Rep	C_{max}	MDs	P1	P2	P3	P4	P5
mt06	-	-	-	-	-	1	55	2	1.7	0	0	0	0
						2	55	1	0	0	0.2	0.1	0.1
					+	1	58	2	2.3	0.3	0.7	0.6	0.3
						2	58	2	4.1	0.8	1.3	1.1	0.9
				+	-	1	56	1	0	0	0.1	0	0
						2	56	2	1.7	0.9	1.4	1.2	0.4
					+	1	62	2	0.8	0.3	0.7	0.6	0.6
						2	62	1	0	0	0.2	0.1	0
			+	-	-	1	55	2	2.7	0	0	0	0
						2	55	2	4.3	0	0	0	0
					+	1	57	3	6.7	0.7	1.1	0.8	0.8
						2	57	2	1.1	0	0.1	0	0
				+	-	1	61	1	0	0	0	0	0
						2	61	2	0	0	0	0	0
					+	1	56	2	2.1	0.1	0.4	0.2	0.2
						2	56	2	1.4	0	0	0	0
mt20	+	-	-	-	-	1	1165	2	2.7	0.6	0.9	0.8	0.1
						2	1165	2	3.9	0.2	0.6	0.5	0
					+	1	1173	3	8.8	1.1	2.3	1.9	0.6
						2	1173	2	1.6	0.8	1.5	0.9	0.4
				+	-	1	1176	2	2.6	0.4	1.1	0.6	0.1
						2	1176	2	4.5	0.5	1.9	0.6	0.2
					+	1	1164	3	6.2	1.7	2.7	2.2	0.6
						2	1164	2	2.1	0.3	0.6	0.4	0
			+	-	-	1	1173	3	5.3	0.9	1.4	0.9	0.2
						2	1173	2	2.9	0	0	0	0
					+	1	1169	2	1.4	0.1	0.6	0.3	0
						2	1169	2	0.4	0	0.2	0.1	0
				+	-	1	1158	2	0	0	0	0	0
						2	1158	2	0.1	0	0	0	0
					+	1	1172	1	0	0	0.3	0	0
						2	1172	2	2.8	0.4	0.6	0.6	0
Total number of robust schedules									7	15	9	12	18

of problem than any other policies. Policy P2 provides the second highest number of 100% robust schedules, followed by P4, P3 and P1. Again it is worth noting that the total number of robust schedules generated by P5 is much higher than that of P2.

It can concluded from the previous discussions that, in general, policy P5 is superior to all others. Table 9 shows the average, minimum and maximum percentage change in makespan due to disruptions across all 384 runs. In robust scheduling, the accuracy for schedule makespan is considered as the most important feature. Hence, a change in makespan more than a percentage is generally not accepted, because the introduction of slack is aimed to absorb and nullify the ripple-down effect of disruptions, preserving the robust schedule C_{max}.

Table 8: Machine disruptions C_{max} change for 11x5 & 13x4 'orb' problems

Prob.	Problem Factors						BRS	# of	Slack Policies				
Name	N	M	SR	MTTR	LOD	Rep	C_{max}	MDs	P1	P2	P3	P4	P5
orb1	-	-	-	-	-	1	1061	2	1.5	0.6	1.7	1.1	0.2
						2	1061	2	2.3	0.4	2.1	1.8	0.1
					+	1	1078	2	1.7	0.2	0.9	0.6	0
						2	1078	3	5.1	1.2	1.5	1.2	0.6
				+	-	1	1067	2	4.3	0.6	2.1	1.1	0.1
						2	1067	2	1.8	0.2	1.4	0.8	0
					+	1	1085	3	6.2	1.1	2.6	1.8	0.6
						2	1085	2	2.2	0.4	0.8	0.5	0.1
			+	-	-	1	1085	2	3.1	0	0.4	0.2	0
						2	1085	2	1.8	0	0	0	0
					+	1	1069	2	1.6	0	0.4	0	0
						2	1069	3	2.9	0.6	0.8	0.6	0.2
				+	-	1	1074	1	0	0	0	0	0
						2	1074	2	2.4	0	0	0	0
					+	1	1077	3	0.6	0.4	0.7	0.6	0
						2	1077	2	1.8	0	0	0	0
orb8	+	-	-	-	-	1	904	2	3.3	0.2	1.4	1.1	0.1
						2	904	1	0	0	0	0	0
					+	1	917	2	0.4	0.1	0.6	0.4	0
						2	917	2	1.7	0.4	1.3	0.9	0.1
				+	-	1	931	2	0.6	0.6	1.8	1.3	0.2
						2	931	3	6.4	1.2	2.7	2.4	0.6
					+	1	921	2	0.7	0.8	2.5	1.6	0.1
						2	921	2	3.1	0.5	1.2	0.9	0.2
			+	-	-	1	928	2	2.5	0.2	0.9	0.6	0
						2	928	2	0.3	0	0.3	0.2	0
					+	1	931	1	0	0	0	0	0
						2	931	2	1.3	0.2	0.6	0.2	0
				+	-	1	915	3	4.7	0.4	1.4	0.9	0
						2	915	2	6.2	0.1	0.6	0.4	0
					+	1	906	2	2.3	0	0	0	0
						2	906	2	3.5	0.2	0.8	0.6	0.1
Total number of robust schedules									3	10	7	8	17

At least once, each slack policy obtained 100% robust schedule for some problems. Also, policies P2 to P5 performed good based on average percentage change in C_{max}. Note that while the average robustness is good for all policies, P2 and P5 exhibits superior worst case performance (maximum percent C_{max} change) when compared to all other policies. Even though policy P2 exhibited superior performance in problem instances with equal number of jobs and machines, its performance deteriorated in

Table 9: Machine-robustness as a % change in makespan

% change	Slack Policies				
in C_{max}	P1	P2	P3	P4	P5
Average	4.17	0.33	1.06	0.67	0.15
Minimum	0	0	0	0	0
Maximum	15.7	1.6	3.6	2.4	0.9

comparison with P5, when there were more jobs to be processed on fewer machines. The worst performance is exhibited by policy P1, the random slack policy. Having gone through the experimental results and analysis, the next section of this document summarizes the conclusions from this study on machine-robust schedules to manage machine disruptions.

Conclusions

The outcomes of the slack-based robust scheduling techniques for job shop problems can be summarized with the following observations.

- Distributing slack on heavily utilized (near-bottleneck) machines to handle machine disruptions provides a better machine-robust schedule.
- The process of building machine-robust schedules help to accurately predict the maximum completion time (C_{max}) with the presence of random machine disruptions.
- By allowing for the uncertain events that could happen during the schedule execution, the slack-based robust schedule provided more accurate performance information. By accurate performance, we imply that the deviations exhibited from planned schedule to executed schedule during uncertain events are within $\pm 1\%$ range.

It is quite safe to conclude at this point that slack-based robust scheduling techniques do provide stable and more accurate schedules to handle machine disruptions.

In real world scheduling, the ability to predict the actual completion time of the jobs with a high degree of certainty is very valuable to successfully maintain customer commitments and stability in shop floor operations.

We consider this work as a foundation for developing better slack-based policies with a slack optimization algorithm for generating robust schedules to handle machine disruptions. Also, extending the study to more general shop scheduling problems like complex job shops would help to obtain valuable insight to the practicability of robust scheduling in different problem domains.

CHAPTER 3

SLACK POLICIES FOR RELEASE-TIME DISRUPTIONS IN JOB SHOPS

Abstract : Production environments have different suppliers providing raw materials for production, to be available at a specified time and different customer demands for the final products. A production schedule is created satisfying the assumptions, result in a specific release time of jobs. Change in job release time creates a disruption in the planned production schedule. Slack policies are used to create robust schedules to pro-actively handle job release-time disruptions. This paper evaluates the effectiveness of slack policies in handling job release-time disruptions for C_{max} job shop problems. Findings about the "big jobs" and heavily utilized machines are summarized.

Keywords : disruption, robust schedule, slack, shop scheduling, slack policies, simulation, arrival times, job release-time disruption, job shops.

Introduction

Production facilities utilize long term to short term operational plans based on demand over planning horizon and other requirements to create good production schedules to maximize profits. Optimized schedules are known as off-line schedules or off-line optimal schedules (Brucker, 1998). Off-line optimal scheduling is heavily used to optimize operations in many industries having a job shop type production environment.

The deterministic off-line scheduling approach assumes that there is no variability or uncertainty associated with the system. An assumption made when generating optimal schedules for job shops is that all jobs are available at the start of production. Kandasamy et al. (2004) showed that when an off-line optimal schedule is implemented in a dynamic and unpredictable environment, the difference between expected performance and actual performance is high because the off-line schedule fails to accommodate the uncertainties during schedule execution.

In real-world systems where there are both internal and external factors associated with, the assumption of job arrival at the scheduled time is difficult to satisfy. Tang (2000) identified the four major sources of demand disruption in a production environment:

1. uncertainty in external demand
2. uncertainty in supply conditions
3. effect of rolling planning horizon
4. system effect caused by the combination of above three.

In this study, for scheduling purposes all jobs are assumed to be available any time after the beginning of the planning horizon. Based on this assumption, an optimal or near-optimal schedule is generated, which in other words is a short term production plan. Thus, each job in this schedule has a release time associated with it, which is fixed based on the initial schedule.

The schedule is then subjected to disruptions of one or more jobs, resulting in a release time disruption greater than zero. For example, raw material supply from the

supplier may be delayed due to various reasons or the job delivery date may be delayed by the customer effecting the schedule job release time to the system. This disruption event unavoidably brings changes to the static environment of operation assumed under optimal or near-optimal scheduling. These conditions create a constraint on the early start time of the job, which was not known when the schedule was generated.

To handle this disruption, either the schedule must change or it must be able to absorb the disruption. It should be noted that only those changes that delay a job's release time beyond the scheduled release time will cause schedule problems. The term *release-time disruption* is used in this paper to define the type of disruption resulting from a delay in the scheduled job release time. Disruptions are unpredictable events that occur due to the variability in the system (Hopp and Spearman, 2000). Job release-time disruptions in a production facility known to result in market loss and missed delivery deadlines. (Gallego, 1990).

The goal of disruption management is to minimize the effects of unforseen events on production. If disruptive events are planned for in the schedule before their occurrence, it is known as *pro-active* disruption management. When the disruption event is managed after its occurrence during the schedule execution, it is known as *reactive* disruption management.

Since many manufacturing facilities can be modeled as a job shop (Sule, 1997), this study is focused on release-time disruptions in the job shop problems. Robust schedules are explored in this study to pro-actively handle release-time disruptions.

The approach used in this study is based on the use of slack policies to generate robust schedules. The effectiveness of a slack policy in handling the job release-time disruptions is measured by effect of right shifting tasks when necessary on the makespan. Since simulation is used as the tool for evaluating policy effectiveness, the terms job release-times and job arrival times are used synonymously.

The general approach used in this research can be summarized in four simple steps.

1. Utilize the adapted SIMPL simulation library developed by Mooney (2005) to simulate release-time disruptions.

2. Modify the classical job shop problems by Applegate and Cook (1991) to model release-time disruption.

3. Develop robust schedules using various slack policies to handle release-time disruptions.

4. Analyze the effectiveness of slack policies with simulation experiments.

The rest of this document is organized as follows: The next section reviews the literature relevant to the field of release-time disruption management using robust schedules. The slack policies used to create the robust schedules are described in the next section, followed by the test problems used in this study. The results of the experiments are summarized and analyzed in the next section followed by the conclusions of this research.

Relevant Literature

Reviewing the current literature on release-time disruption management showed that some of the studies focused on supply chain related problems (Yang J. and Yu, 2005). In supply chain problems, the study is classified under demand disruption management, where the requirement for an order is changed by the customer, disrupting the scheduled delivery process. Some authors have investigated the applications of proactive as well as reactive methods to handle such demand disruptions (Hall N. G. and Potts, 2007).

The concept of schedule deviation cost by Bean et al. (1991) is considered as the pioneering work in disruption management. The authors proposed the idea of developing a new schedule with finite recovery time after the disruption event to manage the disruption. This method was later known as the reactive method of disruption management. The airline industry and ship handling industry are perhaps the largest implementors of reactive disruption management techniques to recover from demand disruptions.

Kohl et al. (1999) studied the expensive tools developed for generating optimal schedules for various industries. They documented the difficulty of implementing such optimal schedules on a daily basis. Their study was focused on the airline industry, where airlines attempted to develop an optimal flight schedule with very little slack left to accommodate for any disruption. Dorndorf et al. (2007) also studied the phenomena of aircraft arrival disruption at a flight gate and proposed

reactive methods to recover from the delay.

Asperen et al. (2003) studied the arrival process of container ships at a port facility. The authors researched the effects on cargo handling schedules due to a change in ship arrival time at the port. Clauser et al. (2001) studied reactive disruption management in project scheduling problems from the ship building industry.

Yang and Yu (2005) studied disruptions due to various internal and external factors like machine breakdowns, raw materials availability, raw material quality, shipment delays, unanticipated employee absences, stochastic demand fluctuations, power outages, environmental factors, and weather in production environment. The authors observed that when a heavily optimized plan was used, the proper and timely management of disruptions that affected the planned job release-times were almost impossible without rescheduling the entire shop.

Proactive disruption management methods try to anticipate uncertain disruption events and develop schedules to manage such events. One of the oldest approaches in proactive methods is the use of probabilistic models to deal with disruptions. Porteus (1990) and Putterman (1990) proposed Markov decision processes to obtain contingency plans to accommodate unplanned disruptions. Putterman (1990) also showed that the contingency plans derived by Markov decision process are not so effective in dealing with completely random disruptions.

Barahona et al. (2005) studied proactive methods to handle demand disruptions in semiconductor manufacturing. The authors proposed a stochastic programming

approach to produce a tool set that is robust with respect to demand uncertainty. A similar approach has been developed by Swaminathan (2002) in relation to tool procurement planning for wafer fabrication facilities. Both studies draw parallels from the chemical plant capacity expansion studies conducted by Sahinidis and Grossman (1992). Aytug et al. (2004) reviewed various factors that cause uncertainties during schedule execution in a production facility and suggested possible approaches in dealing with the uncertainties. Hopp and Spearman (2000) showed that a CONWIP (Constant Work In Process) system is more robust to job arrival time fluctuations than a pure push system.

Stewart and Golden (1983) proposed different models to quantify fluctuations in demand for the stochastic vehicle routing problem. Bertsimas and Simchi-Levi (1996) proposed the *apriori routing* approach to handle demand fluctuations in the stochastic vehicle routing problem.

It is quite evident that the application of robust schedules to handle job release-time disruptions in job shops is not extensively studied. Slack policies are used to generate job shop schedules that are robust with respect to job release-time disruptions. Such schedules as referred hereafter as *release-time-robust schedules*. The next section of this document summarizes the slack policies used to obtain the release-time-robust schedules.

Slack Policies

The aim of this research is to identify practical slack policies that could be used to obtain release-time-robust schedules to manage schedule job release-time disruptions in job shop. According to Kouvelis and Yu (1997) the best robust schedule is the schedule with the best worst case performance. A schedule incorporating a lot of extra slack need not be very robust when managing a disruption (Kouvelis and Yu, 1997). Dorndorf et al. (2007) argued that the best robust schedules are sub-optimal schedules where no rescheduling is necessary when a disruption occurs.

The idea of *release-time robustness* is quite similar to the machine robustness proposed in the previous chapter of this document. A release-time-robust schedule is obtained by distributing slack in the schedule, specified by a certain *slack policy*. The slack policies are defined by three *slack distribution parameters*:

1. Frequency - number of slack tasks inserted in the schedule
2. Duration - length of individual slack tasks
3. Location - distribution of slack tasks in the schedule by machine.

Slack policies based on previous studies conducted by Davenport et al. (2001), Kandasamy et al. (2004), Mehta and Uzsoy (1998) and Aytug et al. (2004) are used in this study. From the arguments of Aytug et al. (2004) the frequency of the slack tasks depends on the number of late arriving jobs at the facility. The authors also proposed that the duration of the slack tasks should depend on the mean delay in job

arrival times. The location of the slack is also an important parameter that has not been significantly studied.

Slack frequency, duration and location are determined indirectly by specifying total slack and selecting one or more machines (along with tasks on them) for slack insertion. The total slack is set equal to the expected total delay for job release-time disruptions, i.e. $E[\text{Total Slack}] = E[\text{jobs delayed}] * E[\text{time to delay}]$. This total slack time is then distributed equally to the selected subset of machines (location) and then to their tasks by placing dummy slack tasks on the selected machines. Table 10 summarizes the subset of machines chosen (location) for each slack policy to obtain release-time-robust schedules.

Table 10: Location values for slack policies for job release-time disruptions

Policy	Machine selection for slack distribution
P1	One random machine from the set of machines
P2	On all machines
P3	The most heavily utilized machine
P4	On one of the heavily utilized (near-bottleneck) machines
P5	On all heavily utilized (near-bottleneck) machines

Policy P1 selects a machine at random from the set of machines, whereas policy P4 selects one machine randomly from the set of heavily utilized machines. Policy P2 distributes slack on all machines, whereas policy P5 distributes slack to all heavily utilized machines. Policy P3 distributes slack only to the most heavily utilized machine. In general, the more machines chosen, the higher the slack frequency.

Sufficient slack at the time of a disruption can cushion the schedule from the disruption, minimizing the ripple-down or knock-on effects. The right shift approach by Kohl et al. (1999) was used to evaluate the effectiveness of the slack policies. Experiments were conducted to verify slack policies for classic job shop problems to evaluate the importance of policy parameters in managing release-time disruptions.

Test Problems

Experiments conducted for this study applied the slack policies discussed in the previous section to a standard set of job shop problems. These classical job shop problems used by Applegate and Cook (1991) were modified to include job release-time disruptions. These well known problems are from different sources:

1. Adams et. al - 'abz' problems
2. Carlier and Pinson - 'car' problems
3. Muth and Thompson - 'mt' problems
4. Bonn - 'orb' problems
5. Lawrence et al. - 'la' problems.

A problem case considered in this study is specified by its class (author) and the disruption pattern added to it. The number of jobs (N), number of machines (M), individual job processing times ($p'_{ij}s$) and individual job routes are all kept the same as that of the original problem. Disruption parameters for generating release-time

disruption events were added to the problem parameters. The disruption parameters are the number of disrupted jobs (NDJ), slack ratio (SR), and mean job delay (MJD). The goal of this study is to find the significant interactions between the important problem and policy factors in handling job release time disruptions.

There are parameters that are applicable to a job release-time disruption in the job shop problem instance. All those parameters were not considered as factors in this study. Some parameters were eliminated and some parameters were kept constant as described in the following paragraphs.

The mean processing time ($\bar{P}$) is a parameter that defines the grand mean of all the individual job processing times in the classical job shop problems. Based on previous studies by Davenport et al. (2001), Barahona et al. (2005) and Asperen et al. (2003) the mean processing time parameter was eliminated. It is generally considered that the arrival time of a job in a dynamic environment is independent of the mean processing time of the job (Barahona F. and Hood, 2005). Also, the processing times for the test problems were deterministic values that did not vary during schedule execution. Hence, parameter $\bar{P}$ was not considered to influence the job release time disruption.

Similarly another parameter named the LOD defines the location of the disruption event with respect to a machine in the shop. In this case of job release time disruptions, the disruption occurs at the beginning of the schedule (or at time equals zero), due to the change in job arrival time, which is not machine specific. The delayed jobs

are processed on multiple machines, decided by the individual job's predetermined route and hence the location of disruption factor assumes no significance in a job release-time disruption study (Sahinidis and Grossman, 1992).

The parameter *NDJ* determines the number of jobs with a release time disruption for a given schedule, which is a random variable in this study. Aytug et al. suggests that the number of jobs arriving late from its scheduled arrival time to the production facility is usually between 10 - 25% of the total number of jobs to be processed. Also, Asperen et al. (2003) used a uniform distribution to sample the number of ships that arrived late at the port's cargo handling facility from the scheduled arrival time. Hence, for this research, sampled values for NDJ, rounded to the next highest integer, were obtained from $UNIF(0.10 * N, 0.25 * N)$.

The remaining parameters that were considered as factors for this study are listed in Table 11. Factors one through four were considered important by different authors in previous studies. Preliminary analysis showed that these four factors can influence the robustness of a particular release time-robust schedule. Hence, the levels for all the four factors were defined for a 2^k factorial experiments to study the importance of these factors in managing job arrival time disruptions.

Table 11: Job release-time disruption study factor list

Parameter	Parameter Name	Description
1	N	Number of jobs
2	M	Number of machines
3	SR	Ratio of total slack to total processing time
4	MJD	Mean Job Delay

Factors N, and M, determine the job shop problem size. Since N and M had non-unform discrete values with some possible outliers, it is advisable to use the Median Average Technique (MAT) proposed by Montgomery (2001). The median average is considered less sensitive to noise created by outlying values in the data when compared with the conventional mean average method (Devore, 2004), (Schillings, 2008).

The set of values for N are $= \{6, 7, 10, 11, 13, 15, 20\}$ across all job shop problem classes. Using MAT, the averaged median was found to be 11.7 and the closest value to the median was chosen as the splitting point. Since there are odd number of values in the set, the high and low levels for factor N were $N \leq 11(-)$ and $N \geq 12(+)$.

For the number of machines, the values were $M = \{4, 5, 6, 7, 10, 15\}$, across all job shop problem classes. By using MAT, the averaged median was found to be 6.5, splitting the even number of data points equally on both sides of the averaged median. Hence, the high and low levels for factor M were $M \leq 6(-)$ and $M \geq 7(+)$.

The factor SR determines the total duration of slack that is strategically introduced into the schedule. Studies by Davenport et al. (2001), Kandasamy et al. (2004), Dorndorf et al. (2007) and Asperen et al. (2003) have identified the ratio of available slack in the schedule to the schedule makespan to influence the robustness of the given schedule. Also, the SR factor normalizes the introduced slack for comparison between instances. Hence, the levels of SR factor was set to $SR \approx 25\%(-)$ and $SR \approx 50\%(+)$.

The final factor, MJD, is used to sample the delay in the arrival time of the delayed jobs to the facility. Studies conducted by Asperen et al. (2003) on container ship arrivals at port facility have used an exponential distribution to sample the delay in arrival time of the delayed ships. Aytug et al. (2004) argued that the average delay in arrival times range within 5% to 30% of mean processing time. Based on the rationale above, an exponential distribution was chosen to sample the job arrival time delays. Thus MJD becomes the expected value (or mean) of the exponential distribution. The levels for the factor MJD were set at $MJD = 0.05 * \bar{P}(-)$ and $MJD = 0.30 * \bar{P}(+)$.

Table 12 summarizes the levels for all problem factors used in the release time disruption study. When the high and low levels for each experimental factors are set, the next step is to conduct the 2^k factorial experiments and analyze the results to identify the significant factors with respect to release-time-robust schedules in handling job arrival time disruptions. The following section of this document summarizes the experimental results and analysis.

Table 12: Factors and levels for job shop release-time disruption study

Factor	Factor Name	Low (-)	High (+)
1	N	$N \leq 11$	$N \geq 12$
2	M	$M \leq 6$	$M \geq 7$
3	SR	$SR \approx 25\%$	$SR \approx 50\%$
4	MJD	$MJD = 0.05 * \bar{P}$	$MJD = 0.30 * \bar{P}$

Experimental Results And Analysis

As the slack policies and factors for the experiment are defined in the previous section, this section of this document summarizes the experimental methods and results. The introduction section of this document mentioned that the study uses a general purpose simulation to perform functions like robust schedule generation, statistics collection and performance evaluation. This study utilizes robust schedules from various slack policies to manage job release-time disruptions, which is a pro-active method of disruption management. Hence, there is no application of a rescheduling algorithm to manage the job arrival time disruptions in the schedule.

The study approach can be summarized in three steps as shown in Figure 4. First,

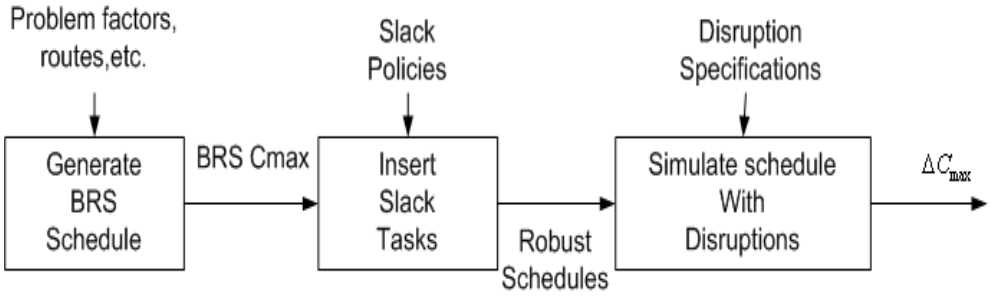

Figure 4: Experimentation process for job release-time disruptions

the modified simulation was used to obtain the baseline random non-delay schedule without the disruptions. All problems were scheduled initially without slack using the Best Random non-delay Schedule (BRS) approach proposed by Mooney and Philip (2006). The authors showed that the algorithm exhibits excellent performance in randomly generated job shop test problems with makespan consistently with 5% of

the best known value. Similar results were shown by (Godard D. and Nuijten, 2005) using the randomized large neighborhood search, built on the same principle. The BRS approach generates a set of random non-delay schedules (or known as starting points) and picks the best random non-delay schedule from the generated set. The number of random starting points generated by BRS was limited to 100 for this study.

Once the best random non-delay schedule was obtained without disruption, the second step modified the BRS schedule by inserting slack in the schedule according to the slack policies. Some of the policies (P1 and P3) involved random sampling and the sampling process for these policies were not replicated. Davenport et al. (2001) have used random policies without replicating the sampling process to be truthful to the other single pass slack policies like P2, P4 and P5. Once the sub-optimal best random non-delay schedules with the slack tasks were obtained, the schedules were stored in order to preserve the job sequence. Hence, at the end of the second step, there are six schedules (five robust schedules and one BRS schedule) to evaluate for each disruption pattern. Thus, by separating the schedule generation before the disruptions, the confounding effects of disruptions and the robust schedules can be avoided.

Third, we introduce the disruptions based on NDJ and MJD to the evaluate the performance of the sub-optimal BRS robust schedules. The performance of each of these release-time-robust schedules are measured using the change in C_{max}. Like all stochastic simulation packages, random variates were sampled from a stream of

random numbers to generate a realization of the system state over time. Thus, each of the random job release-time disruption patterns were obtained using two different random number seeds, resulting in two replications of each problem case.

The experiments were conducted on a Pentium IV 3.06 GHz dual processor machine with four gigabytes of RAM, running Lineox operating system. C++ was used as the programming language with the G++ compiler.

Tables 13 through 17 have similar format and summarize the experimental results for each problem class. In each table, the first four *Problem Factors* columns show the levels of the problem factors listed in Table 12. The *Rep* column under the problem factors represents the replication number of the problem instance. The next column titled *BRS* C_{max} shows the baseline value of C_{max} using the BRS algorithm (2005) without disruption or slack. The number of release-time disruptions that occurred in the run are listed in the column titled *# of RTDs*. The next five columns under *Slack Policies* show the percentage change in C_{max}, $100(\frac{\Delta_{C_{max}}}{BRSC_{max}})$, for each policy for each run. Note that "robustness" can be quantified by the equation $100(1 - \frac{\Delta_{C_{max}}}{BRSC_{max}})$, or a 100% robust schedule has $\Delta C_{max} = 0$. The bottom row of each table summarizes the total number of times the particular slack policy resulted in no change in C_{max} across the problem class.

Table 13 summarizes the results for the 10 job, 10 machine 'abz' problems. The last row titled 'total number of robust schedules' represent the number of times a particular slack policy resulted in a job release-time-robust schedule. It can be noted

that slack policy P2 resulted in the most number of robust schedules for the 10x10 problem class, followed by P5, P4, P3 in descending order. Also, the schedule deviations for P2 and P5 for non-robust schedules are less than a percent. It should be noted that there is not much of a deviation in processing time across all jobs in this problem class and hence the equal slack to all tasks policy performs better. The slack policy P1, which distributes slack on a randomly selected machine performed worst. This result supports the findings of Kandasamy et al. (2004) that distributing slack in the schedule in some intelligent way is always performs better than pure random distribution of slack.

Table 13: Release-time disruptions C_{max} change for 10x10 'abz' problems

Prob.	Problem Factors					BRS	# of	Slack Policies				
Name	N	M	SR	MJD	Rep	C_{max}	RTDs	P1	P2	P3	P4	P5
abz5	+	+	-	-	1	1251	2	1.9	0	1.6	1.1	0.3
					2	1251	2	2.1	0	1.1	0.2	0
				+	1	1264	2	6.4	0.2	1.7	1.2	0.5
					2	1264	2	2.6	0.5	0.9	0.8	0.7
			+	-	1	1255	1	0	0	0	0	0
					2	1255	2	0.3	0	0.2	0	0
				+	1	1259	2	3.7	0	0.6	0.3	0.1
					2	1259	1	0	0	0	0	0
abz6	+	+	-	-	1	974	2	1.4	0.2	0.4	0.4	0.2
					2	974	2	3.5	0.7	1.2	0.9	0.9
				+	1	977	2	4.1	0.6	0.9	0.9	0.7
					2	977	1	0	0	0	0	0
			+	-	1	983	2	1.8	0	0.6	0.5	0
					2	983	2	0.2	0	0	0	0
				+	1	976	2	2.3	0	0.8	0.3	0.3
					2	976	1	0.6	0	0	0	0
Total number of robust schedules								3	11	5	6	8

Table 14 summarizes the results for the 20 job, 15 machine 'abz' problems. From the total number of robust schedules under different slack policies, it is quite evident that both policies P2 and P5 outperforms all other slack policies. The order of performance is P5, P2, P4, P3 and P1, even though the difference between P5 and

P2 is quite negligible. Similarly, the difference between P4 and P3 is also negligible. As expected the random slack policy P1 performs the worst, without returning a single release-time-robust schedule. This class of problems has a lot more jobs than machines, resulting more machines to fall under the heavily utilized machine category. Since there are more tasks available in this problem, the slack duration per task under policy P2 is reduced more than the slack duration per task under policy P5. Hence, policy P5 marginally outperforms policy P2, when the number of jobs are more than the number of machines along with more disruptions.

Table 14: Release-time disruptions C_{max} change for 20x15 'abz' problems

Prob.	Problem Factors					BRS	# of	Slack Policies				
Name	N	M	SR	MJD	Rep	C_{max}	RTDs	P1	P2	P3	P4	P5
abz8	+	+	-	-	1	693	3	6.9	0.4	2.6	1.3	0
					2	698	2	2.1	0	0.3	0.1	0
				+	1	691	3	4.8	0.7	1.7	1.1	0.5
					2	704	3	7.4	0.9	1.6	1.2	0.7
			+	-	1	712	2	3.6	0	0	0	0
					2	707	4	11.2	1.7	2.3	1.9	1.1
				+	1	697	2	4.9	0	0	0	0
					2	693	3	7.1	0.4	1.7	1.5	0.3
abz9	+	+	-	-	1	715	2	5.5	0	0.6	0.5	0
					2	723	4	12.3	0.9	3.4	2.9	0.6
				+	1	717	3	3.7	0.2	1.9	1.2	0.1
					2	729	2	0.4	0	0	0	0
			+	-	1	724	2	2.1	0	0.6	0	0
					2	719	4	6.8	1.7	1.8	1.8	1.2
				+	1	717	3	9.7	0.4	1.9	1.1	0.3
					2	715	4	11.7	0.9	3.2	2.9	0.7
Total number of robust schedules								0	6	3	4	7

Table 15 summarizes the results for 'car' class of job shop problems that are well known for their large processing times combined with smaller number of machines available to process all the jobs. These problems are considered quite hard due to the lack of structure and the variability in processing times, resulting in the presence of some "big" jobs. Also, these problems have multiple heavily utilized machines,

making it a harder problem. It is interesting to note that policy P5 is the winner here, followed by P2, P4, P3 and P1. Even though the difference between number of release-time-robust schedules generated by policies P5 and P2 is more, the difference between policies P4 and P3 is quite negligible. It is worth noting that the policy P1 was able to generate only one release-time-robust schedule. Even in the problems with only one job release-time disruption event, policies except P5 and P2 struggled because they are not set up to handle the big jobs and heavily utilized machines. The behavior of policies P2 and P5 indicates that when there is more variability in the system from big jobs and multiple heavily utilized machines, it is quite important to focus on the big jobs on the heavily utilized machines to obtain better robustness.

Table 15: Release-time disruptions C_{max} change for 11x5 & 10x6 'car' problems

Prob.	Problem Factors					BRS	# of	Slack Policies				
Name	N	M	SR	MJD	Rep	C_{max}	RTDs	P1	P2	P3	P4	P5
car1	-	-	-	-	1	7043	2	3.7	0.2	1.2	0	0.1
					2	7053	2	2.4	0.3	0.9	0.7	0.1
				+	1	7048	1	1.6	0	0	0	0
					2	7053	1	0.7	0.2	0.4	0.3	0
			+	-	1	7065	2	1.6	0.2	0.5	0.3	0.1
					2	7047	2	3.1	0.7	0.9	0.8	0.3
				+	1	7077	2	5.8	0.5	1.1	0.7	0
					2	7053	2	2.3	0.1	0.6	0.5	0
car5	-	-	-	-	1	7735	2	1.5	0.5	1.2	0.8	0.3
					2	7762	1	0.9	0	0	0	0
				+	1	7741	2	1.4	0.3	1.7	1.1	0.4
					2	7756	2	3.6	0.2	0.9	0.7	0.1
			+	-	1	7748	1	0	0	0	0	0
					2	7743	2	6.4	0.2	0.9	0.4	0.2
				+	1	7751	1	2.7	0	0.8	0.7	0
					2	7759	2	1.9	0	0.5	0.4	0.2
Total number of robust schedules								1	5	3	4	7

The next set of experimentation was conducted on the 'mt' class of problems generated by Muth and Thompson and the results are summarized in Table 16. The 'mt' class of job shop problems are generally considered as structured job shop problems.

The policy that performs best in this problem class is P5, immediately followed by policy P2. Policies P4, P3 and P1 all perform equally to be ranked third. It seems that P2 perform better than P5 when there is less variability in the system, as seen in the 'mt06' class of problems. But, policy P5 outperforms P2 in the 'mt20' class problems, where there are 20 jobs to be processed on five machines and there are multiple release-time disruptions. Also, the 'mt20' problem is also known for its "big jobs" and heavily utilized machines. The results of the 'mt20' problem class supports our argument that policy P5 performs better when there is more variability in the system as the policy focuses on the jobs on the heavily utilized machines. All other policies exhibit mediocre performance across the problem instances in this class.

Table 16: Release-time disruptions C_{max} change for 6x6 & 20x5 'mt' problems

Prob.	Problem Factors					BRS	# of	Slack Policies				
Name	N	M	SR	MJD	Rep	C_{max}	RTDs	P1	P2	P3	P4	P5
mt06	-	-	-	-	1	55	1	0	0	0	0	0
					2	59	2	2.7	0	0.8	0.4	0.1
				+	1	58	1	0	0	0	0	0
					2	55	2	3.4	0.2	1.9	1.3	0.5
			+	-	1	56	1	0	0	0	0	0
					2	59	2	1.2	0	0.9	0.6	0.2
				+	1	62	2	0.9	0.5	1.3	1.1	0.7
					2	61	1	0	0	0	0	0
mt20	+	-	-	-	1	1165	2	3.9	0.8	1.3	1.1	0
					2	1171	3	5.7	1.4	2.3	1.9	0.6
				+	1	1169	3	6.1	1.6	2.9	2.6	0.5
					2	1173	4	9.8	2.5	3.4	3.1	1.3
			+	-	1	1176	3	7.3	1.1	2.3	1.8	0
					2	1167	4	11.1	1.6	3.5	2.9	0.9
				+	1	1164	2	3.1	0	0.8	0.6	0
					2	1165	3	5.7	0.8	1.4	0.9	0
Total number of robust schedules								4	7	4	4	8

The next set of experimentation was conducted on the 'orb' class of job shop problems generated by Bonn et al. (1991) and the results are summarized in Table 17. The 'orb' class is well known for its high ratio of jobs to be processed on each machine

with some "big jobs" present in the system. Hence, as expected the policy P5 produces more robust schedules for this class of job shop problems than all other policies. Policy P2 provides the second highest number of robust schedules, followed by P4, P3 and P1. The problems in 'orb' class have more jobs to process on fewer machines and also have more release-time disruptions, introducing more variability in the system. When the delayed job is a "big job," it adds to the already existing variability in the system. Hence, policy P5 performs better, as there is more slack available per task on the heavily utilized machines to absorb the knock-on effects of the disruption. Again, it is worth noting that the total number of robust schedules generated by the policy P5 is much higher than that of P2.

Table 17: Release-time disruptions C_{max} change for 11x5 & 13x4 'orb' problems

Prob.	Problem Factors					BRS	# of	Slack Policies				
Name	N	M	SR	MJD	Rep	C_{max}	RTDs	P1	P2	P3	P4	P5
orb1	-	-	-	-	1	1061	2	3.7	1.1	1.6	1.4	0
					2	1063	1	0	0	0	0	0
				+	1	1078	2	1.4	0.9	1.4	1.2	0.2
					2	1081	1	0	0	0	0	0
			+	-	1	1067	2	2.1	0.2	0.8	0.5	0
					2	1069	2	5.3	0.1	1.3	0.7	0
				+	1	1077	1	0	0	0	0	0
					2	1085	2	1.4	0	0.6	0	0
orb8	+	-	-	-	1	904	1	0	0	0	0	0
					2	923	2	3.1	0.9	1.6	1.3	0
				+	1	917	3	7.4	1.3	2.1	1.8	0.3
					2	919	3	9.7	0.7	1.9	1.2	0.4
			+	-	1	931	2	2.9	0.4	0.8	0.6	0
					2	909	2	5.4	0.4	1.2	0.9	0
				+	1	915	2	6.1	0	0.7	0.6	0.2
					2	921	2	1.3	0	0.9	0.4	0
Total number of robust schedules								3	7	4	5	12

Based on the total number of robust schedules generated by each policy for particular problem class we can conclude that policies P5 and P2 are superior to all others, though it is hard to pick a clear winner between the two policies. Table 18 summarizes

the average, minimum and maximum percentage change in makespan due to release-time-disruptions across all problem classes. In robust scheduling, the accuracy for schedule makespan is considered as the most important feature. Hence, a change in makespan more than a percentage is generally not accepted, because the introduction of slack is aimed to absorb and nullify the ripple-down effect of disruptions, preserving the robust schedule C_{max}.

Table 18: Job release-time robustness as a % change in makespan

% change in C_{max}	Slack Policies				
	P1	P2	P3	P4	P5
Average	3.24	0.43	1.14	0.79	0.21
Minimum	0	0	0	0	0
Maximum	11.7	1.8	3.5	2.8	1.1

At least once, each slack policy obtained 100% robust schedule for some problems. Also, policies P2 to P5 performed good based on average percentage change in C_{max}. Note that while the average robustness is good for all policies, P2 and P5 exhibits superior worst case performance (maximum percent C_{max} change) when compared to all other policies. Policy P2 exhibited superior performance in problem instances with equal number of jobs and machines and lesser release-time disruptions, its performance deteriorated in comparison with P5, when there were more jobs to be processed on fewer machines along with more release-time disruptions. Also, for problem instances with "big jobs" (jobs that have disproportionately large processing times compared to other jobs), policy P5 performed best. This observation leads us to a valuable conclusion that the job release-time disruptions can cause more ripple-

down effect than machine disruptions. Having gone through the experimental results and analysis, the next section of this document summarizes the conclusions from this study on release-time-robust schedules to manage job release-time disruptions.

Conclusions

The outcomes of the slack based robust scheduling techniques for job shop problems with job release-time disruptions can be summarized in the following observations:

- Distributing slack on heavily utilized (near-bottleneck) machines to handle job release-time disruptions provide a better release-time-robust schedule.
- If the variability in the system is less, then distributing slack equally to all tasks on all machines will also result in good performance.
- If there are more jobs to be processed on lesser machines, with some jobs can be classified as "big jobs," then emphasis should be provided for these big jobs on heavily utilized machines.
- In the process of building release-time-robust schedules, the overall schedule quality when compared with the non-robust C_{max} may be diminished, but the accuracy in predicting completion time during random job arrival time disruptions did increase.
- By accounting for the arrival time disruptions, the robust schedules not only exhibited more accurate performance, but also a better overall schedule per-

formance. By better performance, we imply that the deviations from planned schedule to executed schedule during uncertain events are within ±5% range.

It is quite safe to conclude at this point that slack based robust scheduling technique do provide stable and more accurate schedules to handle job release-time disruptions. In real world scheduling, the ability to predict the actual completion time of the jobs with a high degree of certainty is necessary to successfully maintain customer commitments. We consider this work as a foundation for developing better slack-based policies with a slack optimization algorithm for generating robust schedules to handle job release-time disruptions. Also, extending the study to a more general shop scheduling problem like complex job shops would help to obtain valuable insight to the practicability of robust scheduling in different problem domains.

CHAPTER 4

SOME GENERAL RULES FOR DISRUPTION MANAGEMENT IN JOB SHOPS

Abstract : This study combines and generalizes the results of research evaluating slack policies to schedule for unanticipated disruptions in job shops. The disruptions studied in are of two types - machine failure disruptions and job release-time disruptions. Different slack policies based on frequency, duration and location of slacks in the schedule were used to obtain robust schedules. The study was conducted using modified classical job shop problems with the maximum completion time (C_{max}) objective. We conclude that by focusing the slack distribution on the heavily utilized machines can result in good robust schedules.

Keywords : disruption, robust schedule, slack, shop scheduling, slack policies, simulation, arrival times, job release-time disruption, job shops.

Introduction

All production facilities utilize schedules to plan their activities over time. The definition of scheduling by Baker (1974) emphasizes the allocation of resources over time to perform a collection of tasks. Hence scheduling involves both operation sequencing and timing on resources (Baker and Scudder, 1990). Scheduling aims to get good values for one or more performance measures. Typical performance measures include makespan (C_{max}), lateness (L_{max}), Work In Process (WIP), and the number of tardy jobs. The method of obtaining schedules that result in better values for

performance measures is popularly known as schedule optimization (Yang J. and Yu, 2005).

When schedules are optimized, deterministic assumptions, which ignore uncertainty or variability in the system, are often made. Some typical assumptions are:

- All jobs are available at the beginning of the schedule.
- No machine breakdowns will occur during the schedule execution.
- All internal and external shop factors remain static throughout the schedule execution.

This type of scheduling approach is normally called as off-line optimal scheduling (Brucker, 1998).

In the real world, systems have uncertainty and variation associated with them and disruptions occur due to the inherent variability in the system (Hopp and Spearman, 2000). This study focused on job shops, often used to study manufacturing systems with high variability in routes and processing times (Sule, 1997). Based on the extent of study conducted on job shops, McKay et al. (1989) suggested that "the [static job shop] problem definition is so far removed from job shop reality that perhaps a different name for the research should be considered." The authors found that modern scheduling technology developed for job shops failed to address scheduling in uncertain, dynamic environments. Researchers also showed that disruptions adversely affect the expected performance of off-line optimal schedules resulting in revenue losses, loss of customer good will, and in many cases potential future customers

(N. Kandasamy and Karsai, 2004).

To date, the two major areas of disruption management research are airline scheduling and single machine scheduling (Yang J. and Yu, 2005). Disruptions are typically managed either proactively or reactively (Vieira G. and Lin, 2003). In proactive management, disruptions are planned for in the schedule where as reactive management aims at rescheduling after the occurrence of the disruption. The match-up scheduling work of Bean et al. (1991) was the pioneering study in reactive disruption management.

Many authors have studied both proactive and reactive disruption management methods to handle different disruption types. Except for a few studies, almost all limited the focus to a particular disruption type. However, Kohl et al. (1999) showed that the possibility of multiple disruption types makes planning much harder. The authors investigated the disruptions that occur in the airline industry due to multiple factors and the *ripple-down effect* on a well-optimized schedule and found that a combination of both proactive and reactive methods provide good solutions. According to Yu and Qi (2004), planning for multiple disruption types are as important as multiple disruptions of the same type. Dorndorf et al. (2007) studied multiple disruption types in the flight gate scheduling problem and found a combination of proactive and reactive methodology to manage the disruptions.

Chapters 2 and 3 studied machine-robust and release-time robust schedules generated with slack policies were studied independently. However, previous studies have

shown that schedules designed to specifically handle one disruption type often fail to accommodate multiple disruption types (Kohl N. and Tiourine, 1999). In this study, we analyze the performance of the slack policies developed in chapters 2 and 3 in handling both disruption types in job shop like production environments to verify the general applicability of the approach.

The rest of this chapter is organized as follows. The next section summarizes the slack policies used in this research, followed by an overview of the research method. Conclusions and recommendations are summarized in the final section.

Slack Policies

The studies in chapters 2 and 3 on machine and job release-time disruption considered five slack polices. Slack policies specify the number, length and placement of dummy, or slack, tasks in the schedule with three slack policy parameters: frequency, duration and location. Slack frequency, duration and location are determined indirectly by specifying total slack and selecting one or more machines (along with tasks on them) for slack insertion.

The total slack is set equal to the sum of the expected total downtime for machine disruptions and the expected total release-time delay for release-time disruptions. This total slack time is then distributed equally to the selected subset of machines and then to their tasks. Table 19 summarizes the subset of machines chosen (location) for each slack policy.

Table 19: Location values for slack policies for multiple disruption types

Policy	Machine selection for slack distribution
P1	One random machine from the set of machines
P2	On all machines
P3	The most heavily utilized machine
P4	On one of the heavily utilized (near-bottleneck) machines
P5	On all heavily utilized (near-bottleneck) machines

Policy P1 selects a machine at random from the set of machines, whereas policy P4 selects one machine randomly from the set of heavily utilized machines. Policy P2 distributes slack on all machines, whereas policy P5 distributes slack to all heavily utilized machines. Policy P3 distributes slack only to the most heavily utilized machine. In general, the more machines chosen, the higher the slack frequency.

From the study on slack policies for multiple machine disruptions, summarized in chapter 2, the following observations were made:

- Policy P5 generated the most robust schedules across all problem classes, followed by policy P2.
- For problems where not many jobs are to be processed on the available machines, and there were no "big jobs" with long processing times in the schedule, policy P5 and P2 performed equally good.
- For problems with more jobs to be processed on the available machines, and have "big jobs" present in the schedule, policy P5 performed best.
- The availability of slack on the disrupted machine (whether heavily utilized machine or not) at the time of disruption absorbed the disruption effects, thus

reducing the knock-on effect on the schedule.

Similarly, from the results of robust scheduling for the job release-time-disruptions using slack policies in chapter 3, the following conclusions were made:

- In general, policy P5 generated the most number of robust schedules across all problem classes.
- The number of robust schedules generated by policy P5 was much higher compared to other policies on problem instances with more variability (i.e. more number of jobs to be processed on available machines, presence of "big jobs").
- When "big jobs" (i.e. jobs with longer processing times) are present in the schedule, adequate slack on heavily utilized machines was able to minimize the ripple-down effect of job release-time disruptions.
- Generally, more slack was needed in the schedule to manage job-release-time disruptions than machine disruptions.

The next section summarizes the research method used in this study.

Research Methods

Experiments for this study applied the policies discussed earlier to a subset of the harder classical job shop test problems used by Applegate and Cook (1991) and identified in chapters 2 and 3, modified to include multiple disruption event types. These well-known problems are from several sources:

1. Adams et al. (1998) - 'abz' problems,

2. Carlier and Pinson (1989) - ‘car’ problems,

3. Muth and Thompson (1991) - ‘mt’ problems,

4. Bonn (1991)- ‘orb’ problems,

5. Lawrence et al. (1991) - ‘la’ problems.

Each problem instance has a specified number of jobs (N), number of machines (M), processing time durations (p_{ij}'s) and routes, which were kept the same as the original problem. Disruption parameters were added to obtain different disruption types (machine breakdown and job release-time delays).

This study was limited to one disruption of each type. By limiting each scenario to one disruption of each type, most factors in the previous studies were fixed. The mean time to repair factor for machine breakdown was fixed at 30% of the mean processing time, whereas the mean job delay for the release-time disruption was fixed at 20% of the mean processing time. The machine to breakdown was fixed as the most heavily utilized machine. Based on the results from chapters 2 and 3, the study was limited to the 10 job, 10 machine problems and to the 11 job, 5 machine problems.

As shown Figure 5, the study approach can be summarized as a three step process. First, the Best Random non-delay Schedule (BRS) approach proposed by Mooney and Philip (2006) was used to obtain the baseline schedule without slack or disruptions. The BRS approach exhibited good performance in the randomly generated job shop test problems. Godard and Nuijten (2005) arrived at the same conclusion using their

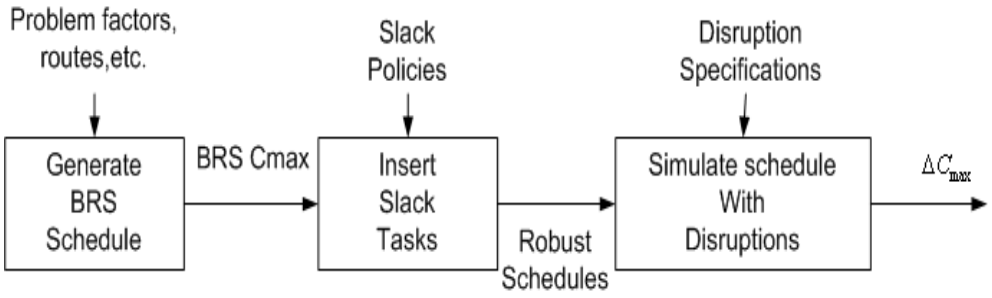

Figure 5: Experimentation process approach for multiple disruption types

large neighborhood search. The number of random schedules generated by BRS to pick the best schedule was limited to 100 for this study.

The second step modified the BRS schedule by inserting slack according to each of the five slack policies. This resulted in a total of six schedules obtained from a common job sequence, which were stored to preserve the job sequence and slack distribution. Generating these schedules before simulating the disruptions minimized the confounding effect of BRS schedule generation and disruption events.

The third step subjected the saved schedules to different disruption types simulated based on the disruption parameters. The right shift approach by Kohl et al. (1999) was used to evaluate the effectiveness of the slack policies by quantifying the amount of disruption absorbed by the strategically distributed slacks in the schedule. If the slack policy results in a robust schedule then the net ripple-down effect as measured by the change in makespan is equal to zero. Any increase in the schedule's C_{max} implies the failure of the slack policy to fully accommodate the disruption, thus quantifying the performance of each schedule under different disruption

types. Unlike other studies where the effect of a single disruption event type was studied, this study simulated and studied multiple machine disruptions and multiple job-release-time disruptions.

The runs were made on a Pentium IV 3.06 GHz dual processor machine with four gigabytes of RAM, running the Lineox operating system. C++ was used as the programming language with the G++ compiler. The results for each problem class follow.

Conclusions And Recommendations

Tables 20 and 21 summarize the experimental results for each problem class. In each table, the first three *Problem Factors* columns show the levels of the problem factors as mentioned in previous section. The *Rep* column under the problem factors represents the replication number of the problem instance. The next column titled *BRS* C_{max} shows the baseline value of C_{max} using the BRS algorithm (2005) without disruption or slack. The next five columns under *Slack Policies* show the percentage change in C_{max}, $100(\frac{\Delta_{C_{max}}}{BRSC_{max}})$, for each policy for each run. Note that "robustness" can be quantified by the equation $100(1 - \frac{\Delta_{C_{max}}}{BRSC_{max}})$, or a 100% robust schedule has $\Delta C_{max} = 0$. The bottom row of each table summarizes the total number of times the particular slack policy resulted in no change in C_{max} across the problem class.

Table 20 summarizes the results for the 10 job, 10 machine 'abz' problems. Slack policies P2 and P5 have the most robust schedules for the 10x10 problems, followed

by P4, P3 and P1 in the descending order. It should be noted that policies P3 and P1 were unable to generate robust schedule for any instances. This result supports the argument of Kandasamy et al. (2004) that distributing slack in the schedule in some intelligent way always performs better than pure random distribution of slack.

Table 20: Multiple disruption types C_{max} change for 10x10 'abz' problems

Prob.	Problem Parameters				BRS	Slack Policies				
Name	N	M	SR	Rep	C_{max}	P1	P2	P3	P4	P5
abz5	10	10	25%	1	1251	12.3	1.4	2.4	1.8	1.2
				2	1251	15.1	0.9	2.1	1.9	0.9
			50%	1	1264	4.7	0	0.9	0	0
				2	1264	2.1	0	0.8	0.2	1
abz6	10	10	25%	1	974	9.6	0	2.4	1.2	0.4
				2	974	11.8	1.1	1.9	1.6	0
			50%	1	977	6.5	0	0.6	0.2	0
				2	977	5.2	0	0.9	0	0
Total number of robust schedules						0	5	0	2	5

Table 21 summarizes the results for 'car' class of job shop problems that are well known for their large processing times combined with a smaller number of machines relative to the number of jobs. These problems are considered quite hard in the job shop scheduling literature due to the lack of structure and the variability in processing times. Policy P5 is the winner here, followed by P2, P4, P3 and P1. It is worth noting that the policies P3 and P1 were once again unable to generate a single robust schedule. The difference in the number of robust schedules generated between policies P5 and P2 shows that when there is more variability in the system and more jobs to process on fewer machines, distributing slack on heavily utilized (near-bottleneck) machines provides a superior solution.

Table 21: Multiple disruption types C_{max} change for 11x5 & 10x6 'car' problems

Prob.	Problem Factors				BRS	Slack Policies				
Name	N	M	SR	Rep	C_{max}	P1	P2	P3	P4	P5
car1	11	5	25%	1	7053	15.3	1.6	3.4	1.1	0.4
				2	7053	14.9	2.2	3.4	2.8	0.5
			50%	1	7048	9.7	0	0.6	0.4	0
				2	7048	6.1	0.9	1.1	0.9	0
car5	10	6	25%	1	7735	12.7	1.2	2.6	2.1	0.3
				2	7735	16.4	1.6	1.8	1.8	0.5
			50%	1	7741	8.6	0.6	0.9	0.8	0
				2	7741	9.8	0	0.8	0	0
Total number of robust schedules						0	2	0	1	4

In the disruption management field a C_{max} deviation of more than a percentage is often considered significant. By that measure, policy P5 performs better when both machine and job-release-time disruptions are present. Hence, based on the results of the pilot study, the following rules are proposed for proactively managing machine and/or job-release-time disruptions:

- If the problem has small variability (i.e. not many jobs to process on available machines and no large deviations in individual job processing times), distributing slack equally to all tasks on all machines (P2) or equally distributing slack to all tasks on heavily utilized machines (P5) will result in good robust schedules.
- If there is more variability in the system (i.e. more jobs to be processed on the available machines and large deviations in individual job processing times), distributing slack equally to all tasks on heavily utilized machines is necessary to obtain good robust schedules.

Figure 6 summarizes the rules. In real world scheduling, the ability to predict the actual completion time of a set of jobs with a high degree of certainty is very valuable.

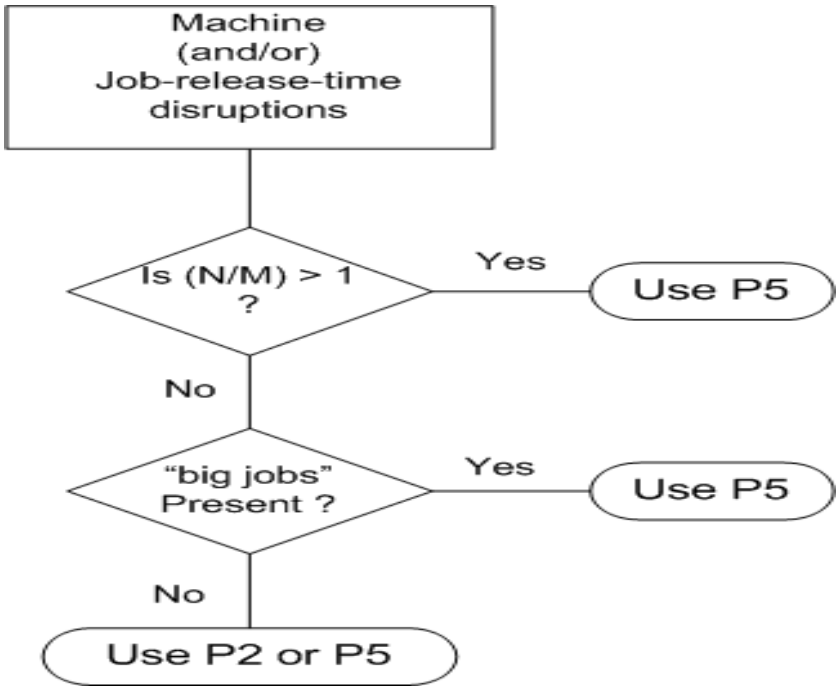

Figure 6: Disruption management framework for job shops

Slack policies can be effectively used to obtain robust schedules for job shops that handle multiple disruption types. Policies P5 and P2 performed consistently better compared to all other policies across all problem instances.

It should be straightforward to extend the this approach to incorporate a feedback loop to search for optimal robust schedules with respect to makespan for multiple disruption types. Also if there is any relationship between a BRS job sequence and the slack introduced in the schedule, that interaction can also be studied. Extending the study to include due date objectives and to more generalized systems could provide a generalized framework for robust shop scheduling process.

CHAPTER 5

CONCLUSIONS

This study explored disruption management in job shops using robust schedules. It was found that robust schedules obtained by strategically distributing slack in a schedule can handle different disruption events. Machine disruptions and job release-time disruptions were first studied individually. The policies from the two studies were tested further with problem instances allowing both disruption types (machine breakdowns and job release-time delays). Guidelines for developing robust job shop schedules followed from these results.

The studies showed that slack policies that distributed slack equally to heavily utilized machines' tasks (P5) or that distributed slack equally to tasks on all machines (P2) performed best. For problems with fewer jobs to be processed on a set of machines and no jobs with extremely long processing times ("big jobs"), P2 and P5 performed equally well. But, for problems with more jobs to be processed on a set of machines or jobs with longer processing times ("big jobs"), policy P5 performed best. Also, it was observed that job release-time disruptions can result in a larger ripple-down effect in a schedule than machine disruptions.

It is quite safe to conclude at this point that slack-based robust scheduling provides stable schedules and more accurate makespan prediction in the presence of disruptions. In real world scheduling, the ability to predict the actual completion time of a set of jobs with a high degree of certainty is very valuable. Robust schedules provide

that degree of certainty.

In this research, no attempt was made to find optimal sequences with slack included nor to optimize slack distribution among the selected tasks. It should be straightforward to extend this approach to incorporate a feedback loop to search for robust schedules that are optimal with respect to makespan. Enhancing the approach to allow for unequal slack distribution would also be possible, but a search for the best distribution would take significant work. Future work might also consider due-date related objectives as well as more general shop scheduling models. We suspect slack policies of the type explored here will also prove useful in these environments.

REFERENCES CITED

Adams J., E. Balas, D. Zawack. 1998. The shifting bottleneck procedure for job shop scheduling. *Management Science* **34** 391–401.

Akturk, M. S., E. Gorgulu. 1999. Match-up scheduling under a machine breakdown. *European Journal of Operational Research* **112** 81–97.

Applegate, D., W. Cook. 1991. A computational study of the jobshop scheduling problem. *ORSA Journal of Computing* **3** 149–156.

Ariri I., E. Frostig, J. Bruno, A. H. G. R. Kan. 1989. Single machine flow-time scheduling with a single breakdown. *Acta Informatica* **26** 679–696.

Asperen E. V., M. Polman, E. Dekker, H. de Swaan Arons. 2003. Arrival process in port modeling: insights from a case study. *Proceedings of 2003 Winter Simulation Conference*, vol. 2. 1737–1744.

Aytug H., K. McKay S. Mohan R. Uzsoy, M. A. Lawley. 2004. Executing production schedules in the face of uncertainities: A review and some future directions. *European Journal of Operations Research* **161** 86–110.

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

Baker, K. R., G. D. Scudder. 1990. Sequencing with earliness and lateness penalties: A review. *Operations Research* **38** 22–36.

Barahona F., O. Gunluk, S. Bermon, S. Hood. 2005. Robust capacity planning in semiconductor manufacturing. *Naval Research Logistics* **52** 459–468.

Bean J. C., J. Mittenthal, J. R. Birge, C. E. Noon. 1991. Match-up scheduling with multiple resources, release dates and disruptions. *Operations Research* **39** 470–483.

Bertsimas, D. J., D. Simchi-Levi. 1996. A new generation of vehicle routing research: Robust algorithms addressing uncertainity. *Operations Research* **44** 286–304.

Birge, J. R., F. Louveaux. 1997. *Introduction to Stochastic Programming*. Springer-Verlag, New York.

Brucker, P. 1998. *Scheduling Algorithms (2nd Edition)*. Springer Verlag, Berlin.

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

Clauser J., J. Larsen, J. Hansen, L. Larsen. 2001. Disruption management. *OR/MS Today* **28** 40–43.

Davenport A. J., C. Gefflot, J. C. Beck. 2001. Slack based techniques for robust schedules. *Proceedings of 6th European conference on planning*.

Devore, J. L. 2004. *Probability and Statistics for Engineers and Scientists*. Thomson Brooks/Cole, Belmont.

Dorndrof U., C. Lin H. Ma, F. Jaehn, E. Pesch. 2007. Disruption management in flight gate scheduling. *Statistica Neerlandica* **61** 92–114.

Eden C., F. Ackerman, T. Williams, S. Howick. 2000. The role of feedback dynamics in disruption and delay on the nature of disruption and delay in major projects. *Journal of Operations Research Society* **51** 291–300.

Gallego, G. 1990. Scheduling the production of several items with random demands in a single facility. *Management Science* **36** 1579–1592.

Gao, H. 1995. Building robust schedules using temporal protection - an empirical study of constraint-based scheduling under machine failure uncertainity. Masters thesis, University of Toronto.

Godard D., P. Laborie, W. Nuijten. 2005. Randomized large neighborhood search for cumulative scheduling. *ICAPS-05, Monterey, California*.

Hall N. G., X. Liu, C. N. Potts. 2007. Rescheduling for multiple orders. *INFORMS Journal on Computing* **19** 633–645.

Hopp, W. J., M. L. Spearman. 2000. *Factory Physics (Second Edition)*. Irwin-McGraw Hill.

Kohl N., J. Larsen A. Ross, A. Larsen, S. Tiourine. 1999. Airline disruption management - perspective, experiences and outlook. Project report, CEC Contract (IST 1999-14049).

Kouvelis, P., G. Yu. 1997. *Robust Discrete Optimization and its Applications*. Kluwer Academic Publishers, Dordrecht, Netherlands.

Lambert V., G. Laporte, F. V. Louveaux. 1993. Designing collection routes through bank branches. *Computers and Operations Research* **20** 783–791.

Mason S. J., S. Jin, C. M. Wessels. 2004. Rescheduling strategies for minimizing total weighted tardiness in complex job shops. *International Journal of Production Research* **42** 613–628.

McKay K. N., J. A. Buzacott, F. R. Safayeni. 1989. *The scheduler's knowldege of uncertainity: The missing link (Book Chapter)*. Knowledge Based Production Management Systems (Elsevier Science Publications, North-Holland).

Mehtha, S. V., R. M. Uzsoy. 1998. Predictable scheduling for a job shop subject to breakdowns. *IEEE Transactions on Robotics and Automation* **14** 365–378.

Mehtha, S. V., R. M. Uzsoy. 1999. Predictable scheduling of a single machine subject to breakdowns. *IEEE Transactions on Robotics and Automation* **12** 15–38.

Montgomery, D. C. 2001. *Design and Analysis of Experiments*. John Wiley and Sons, New York.

Mooney, E. 2005. Simpl library. Technical report, Mechanical and Industrial Engineering Department, Montana State University Bozeman.

Mooney, E., D. Philip. 2006. Scheduling dynamic reentrant flexible job shops with sequence dependent setups. *INFORMS 2006 Conference, Pittsburgh*.

Mooney, E. L. 2007. Private communication.

N. Kandasamy, C. V. Buskirk H. Neema, D. Hanak, G. Karsai. 2004. Synthesis of robust task schedules for minimum disruption repair. *IEEE International Conference on Systems, Man and Cybernetics*, vol. 6. 5056–5061.

Natale, M. D., J. A. Stankovic. 1994. Dynamic end-to-end guarantees in distributed real time systems. *Proceedings of real time systems symposium*. 216–227.

Porteus, E. L. 1990. *Stochastic Inventory Theory*. Handbooks in Operations Research and Management Science, Vol.2: Stochastic Models, Elsevier Science Publishers B.V., North Holland, pp 605-652.

Putterman, M. L. 1990. *Markov Decision Process*. Handbooks in Operations Research and Management Science, Vol.2: Stochastic Models, Elsevier Science Publishers B.V., North Holland, pp 331-434.

Sahinidis, N. V., I. E. Grossman. 1992. Reformulation of the multiperiod milp model for capacity expansion of chemical processes. *Operations Research* **40** 127–144.

Schillings, P. 2008. Private communication.

Smith, S. 1994. *OPIS: A methodology and architecture for reactive scheduling*. Intelligent Scheduling by Morgan Kaufmann, San Mateo, CA.

Sorensen, K., M. Sevaux. 2003. Robust and flexible vehicle routing in practical situations. Working paper, University of Antwerp.

Stewart, W. R., B. L. Golden. 1983. Stochastic vehicle routing: A comprehensive approach. *European Journal of Operational Research* **14** 371–385.

Sule, D. R. 1997. *Industrial Scheduling*. PWS Publishing Company, Boston.

Swaminathan, J. M. 2002. Tool procurement planning for wafer fabrication facilities: A scenario based approach. *IIE Transactions* **34** 145–155.

Tang, O. 2000. Planning and replanning within the material requirements planning environment - a transform approach. *Production-Economic research in Linkoping, Sweden*, vol. 16.

Teodorovic, D., S. Guberinic. 1984. Optimal dispatching strategy on an airline network after a schedule perturbation. *European Journal of Operational Research* **15** 178–182.

Vieira G., J. Herrmann, E. Lin. 2003. Rescheduling manufacturing systems: A framework of strategies, policies and methods. *Journal of Scheduling* **6** 39–62.

Yang J., X. Qi, G. Yu. 2005. Disruption management in production planning. *Naval Research Logistics* **52** 420–442.

Yu, G., Qi X. 2004. *Disruption management: framework, models, solutions and applications*. World Scientific Publishers: Singapore.

Zhu G., J. F. Bard, G. Yu. 2005. Disruption management for resource constraint project scheduling. *Journal of Operations Research Society* **56** 365–381.