Winter 2006

A Robust Reactive Scheduling System with Application to Parallel Machine Scheduling

Jean-Paul M. Arnaout
Old Dominion University

A ROBUST REACTIVE SCHEDULING SYSTEM WITH APPLICATION TO PARALLEL MACHINE SCHEDULING

by

Jean-Paul M. Arnaout
B.S. July 2002, University of Balamand
M.E. December 2003, Old Dominion University

A Dissertation Submitted to the Faculty of
Old Dominion University in Partial Fulfillment of the
Requirement for the Degree of

DOCTOR OF PHILOSOPHY

ENGINEERING MANAGEMENT AND SYSTEMS ENGINEERING

OLD DOMINION UNIVERSITY
December 2006

Approved by:

Ghaith Rabadi (Director)

Resit Unal (Member)

Shannon Bowling (Member)

Steve Cotter (Member)

ABSTRACT

A ROBUST REACTIVE SCHEDULING SYSTEM WITH APPLICATION TO PARALLEL MACHINE SCHEDULING

Jean-Paul Arnaout
Old Dominion University, 2006
Director: Dr. Ghaith Rabadi

In this turbulent world, scheduling role has become crucial in most manufacturing production, and service systems. It allows the allocation of limited resources to activities with the objective of optimizing one performance measure or more. Resources may be machines in a factory, operating rooms in a hospital, or employees in a company, while activities can be jobs in a manufacturing plant, surgeries in a hospital, or paper work in a company. The goal of each schedule is to optimize some performance measures, which could be the minimization of the schedule makespan, the jobs' completion times, jobs' earliness and tardiness, among others.

Until very recently, research has concentrated on scenarios that assume a predefined schedule that is failure free. Initial schedules produced in advance are being followed hoping no delays will occur, because once they do, the whole schedule may be compromised as it is not designed to adapt to change. Researchers focused on the generation of good schedules in the presence of complex constraints while assuming fixed processing times, known job arrival times, unbreakable machines, and immune employees. However, this is not the case in the real world, where processing times are stochastic, job arrival times could be unknown, machines do break down, and employees get sick. In fact, most environments including manufacturing are dynamic by nature and not static, vulnerable to many unpredictable

events, which leads the initial schedule to become obsolete once it is executed. The reason these deterministic schedules fail is because they do not account for variability, scheduling the activities directly after each other, so when a certain activity is delayed, all its successors will be delayed too.

In this dissertation, new repair and rescheduling algorithms, and robust systems equipped with learning capability are developed for the unrelated parallel machine environment, a known NP-hard problem. The introduced rules and algorithms were subjected to different stochastic rates of breakdowns and delays and were judged based on several performance measures to ensure the optimization of both the schedule quality and stability. Schedule quality is assessed based on the schedule Makespan (time to finish all jobs) and CPU, while schedule stability is based on the number of shifted jobs from one machine to another and the time to match up with the original schedule after the occurrence of a breakdown. The extensive computational tests and analyses show the superiority of the proposed algorithms and systems compared to existing methods in the literature, especially when implemented with the learning capability. Moreover, the rules were ranked based on their performance for different performance measure combinations, allowing the decision maker to easily determine the most appropriate repair/rescheduling rule depending on the performance measure(s) desired.

To Karen George, the optimal solution of my life

ACKNOWLEDGMENTS

First and foremost, I would like to thank my advisor, Dr. Ghaith Rabadi, without whom this dissertation would not have been possible. His valuable feedback contributed greatly to this work. I can only wish for any PhD student to be as fortunate as I was in finding not only a mentor, but also a friend who helped me deliver my best through his constant motivation and reinforcement.

I am also very grateful for having an exceptional doctoral committee and wish to thank Dr. Resit Unal, Dr. Shannon Bowling, and Dr. Steve Cotter for their continual support and commitment. I thank as well the Engineering Management and Systems Engineering Department at Old Dominion University for the financial support during my graduate work.

My appreciation is extended to my parents in Lebanon, Michel and Mayda, for their unconditional love and encouragement from day one. Thank you mom for calling me doctor even when I was still in High School, and thank you dad for being my idol; you gave me so much confidence (and sometimes arrogance) by showing me that nothing is unattainable if I put my mind to it. I can only hope that one day I can provide my children with the same happiness and security you gave me. I also want to thank my beautiful sister Christelle and my brother Georgy for their support and motivation throughout this experience.

Finally, I owe the most to my fiancée Karen, who deserves this PhD as much as I do. She had to commit three years of her life in library imprisonment, accompanied by my frequent bad temper. Thank you so much my love for picking me up every time I was down.

TABLE OF CONTENTS

LIST OF TABLES

Table Page

LIST OF FIGURES

CHAPTER I

INTRODUCTION

Scheduling is one of the most crucial factors in manufacturing and production systems. It allows the allocation of scarce resources to activities with the objective of optimizing one or more performance measures (Leung, 2004). Resources may be machines in a factory, operating rooms in a hospital, or employees in a company, while activities can be jobs in a manufacturing plant, surgeries in a hospital, or paper work in a company. The goal of each schedule is to optimize some performance measures, such as the minimization of makespan, jobs' completion time, jobs' earliness and tardiness, among others. Scheduling is a hard problem both in theory and practice (Dorn *et al.*, 1993). Its difficulty in theory is revealed through the excessive combinatorial complexity due to the search for optimal solutions for NP-hard problems. Scheduling is also difficult in practice due to the high number and variety of the constraints required in the real world. Scheduling dates back to 1950s, when researchers in operations research, industrial engineering, and management were faced with the problem of managing various activities occurring in a workshop (Leung, 2004). Until the 1980s, most of the algorithms developed were exact with a goal of reaching optimal solutions. However, the problems' complexity kept on increasing, which made it infeasible to reach optimal solutions. This is when researchers started investing time in approximation algorithms, heuristics, and meta-heuristics, with the goal of finding good solutions at a reasonable computational cost. Nowadays, the scheduling field has acquired an outstanding body of knowledge.

Until very recently, most of the literature dealing with production scheduling has primarily been oriented towards static deterministic environments where complete knowledge of the problem is available without consideration of any kind of failures. Researchers focused on the generation of good schedules in the presence of complex constraints, while assuming fixed processing times, known jobs' arrival times, unbreakable machines, and immune employees. However, this is not the case in the real world, where processing times are stochastic, jobs' arrival times could be unknown, machines do break down, and employees get sick. As a matter of fact, most manufacturing environments are dynamic by nature and not static. They are subject to many unpredictable disruptions that may cause the predefined deterministic schedule to become obsolete once it hits the shop floor (MacCarthy and Liu, 1993). After a disruption, the predefined schedule can become inappropriate to the new conditions. The reason these deterministic predictive schedules fail is because they do not account for variability, scheduling the activities directly after each other; consequently, when a certain activity is delayed, all its successors will be delayed too.

The purpose of this research is to develop a robust scheduling system, which will be capable of coping with new events through inherent rules and rescheduling in order to reduce the variability in the system and maintain the schedule's quality.

AREA OF RESEARCH

Motivated by the obsolescence of deterministic schedules in practical manufacturing problems, this research was oriented towards dynamic scheduling. The latter's growing popularity is revealed through the increasing number of journal articles and conference papers tackling this topic. Most of this literature defines dynamic scheduling as consisting of three constructs: on-line scheduling, predictive-reactive scheduling, and robust scheduling (Mehta and Uzsoy, 1999; O'Donovan *et al.*, 1999; Ouelhadj, 2003).

On-line Scheduling

To overcome the shortfalls of the deterministic preplanned schedule, many researchers have suggested online scheduling for dynamic scenarios (Feldman *et al.*, 1991; Anderson and Potts, 2004), which is a completely reactive scheduling method where no deterministic schedule is produced in advance, and decisions are made locally in real-time. That is scheduling on the fly following some predefined rules such as priority dispatching rules. While online scheduling could be easily implemented, it is very disadvantageous in practice as it is unable to neither predict system performance nor provide any resource planning for the activities, because no initial schedule exists on which basis a scheduler can allocate resources and predict performance.

Predictive-reactive scheduling

This strategy is considered one of the most common in the literature, where a predictive schedule is generated in advance with an aim of minimizing the objective function without considering any possible perturbations. Once perturbations occur during the schedule's execution, reactive scheduling modifies the predictive schedule in an attempt to improve performance and maintain schedule quality. The importance of a predictive schedule is to enable basic planning for the other activities in the system such as labor allocation and material purchase (Shafaei and Brunn, 1999). A predictive schedule can also identify resource conflicts, control the release of jobs to the shop, and ensure that required raw materials are ordered in time. The disadvantage of predictive-reactive scheduling lies in its instability and high variability; since predictive schedules still do not account for variability and disruptions, the reactive process will have to reschedule the initial schedule whenever new events occur, no matter how small the disruption is, resulting in a high rescheduling frequency and a realized schedule that is far from the pre-planned one. This, of course, may lead to resource conflicts and system instability.

Robust Scheduling

The predictive-reactive scheduling is a good strategy for rescheduling but still does not resolve the main weakness of the pre-schedule (predictive schedule), which lies in its inability to cope with disturbances, because rescheduling is still a must upon the occurrence of any disruption. From here came the need for robust predictable-reactive scheduling, which mainly differs from the original predictive- reactive schedule by its predictable schedule. The predictable schedule is a predictive schedule but with added ability of absorbing the disruptions without affecting planned external activities as well as maintaining high shop performance (Mehta and Uzsoy, 1999). A predictable schedule is generated by inserting idle time between the pre-schedule's activities, enabling the disruptions to be smoothed out through the system in order to maintain the schedule quality. If a disruption occurs during the execution of the predictable schedule, rescheduling will only be necessary if the disruption's duration exceeds the inserted idle time.

Following the description of the three dynamic scheduling constructs, it can be realized that robust predictable-reactive scheduling should be a superior construct for the proposed reactive system as it ensures both system stability as well as schedule's quality. Previous literature also agrees with this realization (Mehta and Uzsoy, 1999; Vieira *et al.*, 2003).

BACKGROUND AND SCOPE OF RESEARCH

This section summarizes the building blocks of the proposed system: system's time response, reactive approach, scheduling techniques, learning capability, and the problem environment.

System's Time Response

There are different policies to determine the appropriate time for rescheduling, i.e. the time when reactive scheduling starts. The literature defines three alternatives: periodic, event-driven, and hybrid (Church and Uzsoy, 1992; Sabuncuoglu and Bayiz, 2000; Vieira *et al.*, 2000; Chong *et al.*, 2003).

In a periodic policy, schedules are generated at regular intervals and the dynamic scheduling problem is decomposed into a series of static problems that can be solved by using classical scheduling algorithms. The schedule will be executed and not revised until the next period interval. Rescheduling occurs regularly with a constant time interval (the rescheduling period) between consecutive rescheduling events and no other events trigger rescheduling (Vieira *et al.*, 2000). This will lead to more stability in the system, but leaves the system totally vulnerable to new events that could occur during execution, which might result in a poor performance or even a total system failure.

In an event-driven policy, rescheduling will only take place if an event that can change the system status occurs. Several studies compared periodic and event-driven policies; the latest showed that in turbulent environments, a periodic policy can increase the system ability to react to new events but will demand a lot of set-ups, and much better results were obtained when event-driven policy was used (Vieira *et al.*, 2000).

In a hybrid policy, rescheduling occurs periodically and also when an exceptional event takes place (Church and Uzsoy, 1992). In this policy, you can define which events not to react to, and by rescheduling the system periodically, it stays up to date so it can easily respond to perturbations. The disadvantage of this policy is the large number of set-ups and computational time.

It is worth reminding here that the proposed system will be equipped with a robust predictable schedule that can overcome by itself some of the disruptions. In an event-driven policy, such disruptions will not trigger the system to react; on the other hand, if the disruption durations are larger than the inserted idle time, rescheduling will take place.

Reactive Approach

The literature shows two main alternatives for the reaction process: schedule repair and complete rescheduling (Vieira *et al.*, 2003; Cowling *et al.*, 2003; Ouelhadj, 2003). Schedule repair refers to a minimum modification of the pre-schedule, leading to more stability in the system, while complete rescheduling refers to rescheduling from scratch, which could result in better solutions but will jeopardize system stability. Moreover, complete rescheduling will lead to system nervousness and could be very costly, as all the pre-arranged plans have to be changed. In practice, most rescheduling has been done using schedule repair, except in some severe situations where complete rescheduling had to be done (Abumaizar and Svestka, 1997). In their experimental tests, Cowling *et al.* (2003) showed that the schedule repair strategies attain better performance levels in terms of both stability and utility measures. They stated that even in environments where significant changes in stability are tolerated and improvements in utility are required, schedule repair strategies remained competitive. However, the results indicated that complete rescheduling becomes a superior strategy when a large number of real time events occur.

As the proposed system in this research will be equipped with a robust predictable schedule, some of the disruptions will be smoothed out through the inserted idle time, and in such a case, schedule repair is suitable (Figure 1). On the other hand, when disruption durations become too large for the inserted idle time to maintain the schedule stability and quality, complete rescheduling becomes necessary.

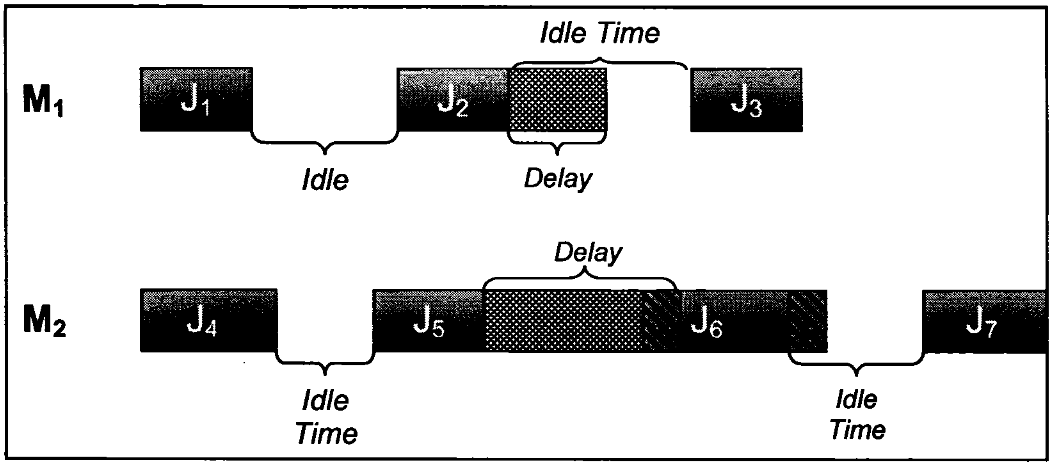

Figure 1. Predictable Schedule subject to Disruptions

Figure 1 shows how disruptions can be smoothed out in a predictable schedule. M_1 and M_2 refer to Machine 1 and Machine 2 respectively, and J_1, J_2,..., J_7 refer respectively to Job 1, Job 2,..., Job 7. As one can see, while being processed on M_1, J_2 encountered a delay, but as this delay's duration was smaller than the inserted idle time, no modification was necessary to the schedule. On the other hand, in M_2, J_5 encountered a delay larger than its allocated idle time; in this case, the start time of J_6 was delayed until the finish time of J_5. Then when J_6 was processed, it was delayed because it had a late start; however, there is enough idle time inserted after J_6, i.e. J_7 could start right on time.

Scheduling Techniques

This section describes the possible dynamic scheduling techniques that can be used in the proposed reactive scheduling system. The literature divides the scheduling techniques into seven main categories: Heuristics based approaches, Dispatching rules and Simulation techniques, Multi agents, Knowledge based scheduling, Constraint based scheduling, Fuzzy logic, and Neural networks (Ouelhadj, 2003; Subramaniam *et al.*, 2005).

Heuristics-Metaheuristics

Crama (2005) proposed the following definition: "A heuristic for an optimization problem *P* is an algorithm which is based on intuitively appealing principles, but which does not guarantee to provide an optimal solution of *P*". A clearer definition was provided by Reeves (1995): "A heuristic is a technique that seeks good solutions at a reasonable computational cost without being able to guarantee either feasibility or optimality, or even in many cases to state how close to optimality a particular feasible solution is". The advantage of heuristics lies in their ease of implementation and reduction of computational time in complex problems; they will be used in the proposed system for schedule repair. Popular repair heuristics include right-shift rule, affected operations, and match-up rescheduling. Many investigations compared these three heuristic types, and the results indicated that match-up scheduling outperformed the other two (Bean *et al.*, 1991; Abumaizar and Svestka, 1997). One main disadvantage in heuristics is that they can easily fall into local optima, but in our proposed work, an improved match-up scheduling method will be used to avoid local optima as much as possible.

Meta-heuristics differ from heuristics by their ability to avoid local optima as they search in different neighborhoods (Reeves, 1995). Three main meta-heuristics are used in dynamic scheduling: tabu search, genetic algorithms, and simulated annealing. The literature shows in many cases that tabu search outperformed the other two under machine scheduling environment (Jozefowska *et al.*, 1998; Youssef *et al.*, 2001; Lee, 2001).

Dispatching rules and Simulation techniques

Despite their ease of implementation, dispatching rules most often lead to poor solutions caused by both their local nature and their large dependency upon the system and job characteristics, i.e. any changes in the system could render the once suitable dispatching rule inappropriate. On the other hand, simulation was used especially under dynamic and stochastic scenarios to compare different rules in order to find which one had the highest effectiveness for a specific scenario, after which the scheduler can choose the most efficient dispatching rule (Arnaout and Rabadi, 2005; Arnaout *et al.*, 2006). "Computer simulation provides a mechanism in which one can capture the essence of a real manufacturing system in the form of a detailed model which can be run, tested, and analyzed in many different ways" (O'kane, 2000). However, the problem with simulation is that it requires a large amount of CPU time, especially in optimization problems, where many runs are needed to obtain gradient information for the decision variables (Kouikoglu and Phillis, 1997). Moreover, it is difficult to find optimal solutions using simulation as the only way to attempt to optimize is to make changes in the variables, rerun the simulation program to check if these changes improved the solution, then repeat this process as long as it takes until reaching the best solution (Beasley, 2006). This of course becomes tedious in complex problems such as the one addressed in this research.

For the reasons just stated, dispatching rules and simulation techniques will not be used in the implementation of the proposed system.

Multi agents

Before multi-agent scheduling, most of the scheduling designs were centralized and hierarchical, which resulted in a very poor reactivity to new events and perturbations. From here came the idea of multi-agent systems, which aims to decentralize the control in the schedules' design, and assumes the presence of many agents with autonomous capabilities. These agents interact and cooperate in order to obtain a global optimal solution. The literature shows two main multi-agent architectures: autonomous and mediator architectures. The autonomous architectures possess high flexibility and robustness, but unfortunately do not always guarantee a global optimum and can become unpredictable in complex systems. On the other hand, mediator architectures can overcome this disadvantage with the help of a mediator that will supervise the agents' coordination to make sure that the schedule is in the direction of global optima. The disadvantage of multi agents lies mainly in the difficulty encountered in the implementation, use, and complexity of coordinating the agents (Subramaniam *et al.*, 2005).

Knowledge-based Scheduling

The main feature of a knowledge-based scheduling system is the identification and application of problem-specific knowledge to solve the addressed problem (Sauer and Bruns, 1997). Some of the branches of knowledge-based systems are constraint-based scheduling, fuzzy logic, and neural networks (Miyashita, 1995; Schmidt, 1994; Garner and Ridley, 1994).

Even though knowledge-based systems can automate human expert reasoning and heuristics to run a production schedule, it is difficult for them to optimize the schedule and upgrade themselves with the needed features to accommodate the new changes. Moreover, they require an extensive database, leading to a large search time (Subramaniam *et al.*, 2005).

Following the above, heuristics will be the scheduling technique used in this research due to its several benefits such as ease of implementation, reduction of computational time, and ability to optimize the schedule and attain good solutions.

Learning Capability

It is important to note the importance of equipping the proposed system with a learning capability. Selfridge (1993) stated: "If an expert system, brilliantly designed, engineered and implemented, cannot learn not to repeat its mistakes, it is not as intelligent as a worm or a sea anemone or a kitten." He then followed: "Find a bug in a program, and fix it, and the program will work today. Show the program how to find and fix a bug, and the program will work forever." Machine learning studies the mechanisms through which intelligent systems improve their performance over time (Shavlik and Dietterich, 1990). Over the past decade, machine learning has evolved from a field of laboratory demonstrations to a field of significant commercial value. Machine-learning algorithms have now learned to detect credit card fraud by mining data on past transactions, learned to steer vehicles driving autonomously on public highways at 70 miles an hour, and learned the reading interests of many individuals to assemble personally customized electronic news (Mitchell, 1997).

As the proposed scheduling system will be subject to dynamic environments, a learning capability becomes crucial in order to stay up to date with the environment. Also the system needs to learn from its mistakes so they would not happen again. For example, suppose that the system assigns 10 minutes of idle time after each job, and after running the schedule for several problem instances, the system detects that this idle time is not sufficient and the jobs are being delayed; in this case, the system should be capable of learning from its past and start assigning larger idle times.

Problem Environment

The rules and policies that are developed for the proposed system will be tested on unrelated parallel machines. The literature defines unrelated parallel machines as machines having different processing times for the same job (Liaw *et al.*, 2003). They are unrelated in the sense that the processing speed depends on the job being executed and not the machine; each job will have different processing times for each of the available machines. Table 1 is an example of jobs' processing times difference over various machines. The processing time is represented by p_{ij}, i.e. processing time of job j on machine i, where $j = 1, \ldots, n$, and $i=1,\ldots,m$. The objective of the problem is to find the optimal combination of jobs to machines that will minimize certain performance measure(s), subject to the following constraints:

- Each job *j* can be processed on any of the machines but needs to be processed by one machine only.
- Each machine *i* is capable of processing one job at a time.
- Job preemption is not allowed.

Table 1. Jobs processing times on unrelated machines

		Jobs			
		1	2	.	n
Machines	1	p_{11}	p_{12}	.	p_{1n}
	2	p_{21}	p_{22}	.	p_{2n}
	.	.	.	.	.
	m	p_{m1}	p_{m2}	.	p_{mn}

The reason for developing the proposed scheduling system for the unrelated parallel machine problem is because the latter is the most general parallel machine scheduling problem. The parallel machine environment includes three main classes: identical machines, uniform machines, and unrelated machines, with the unrelated case being the most difficult (Hoogeveen *et al.*, 2001). Following this, once the proposed rules and policies are developed for the unrelated parallel machine problem, they can be easily transformed with minor modifications to other parallel machine scenarios and environments. Much emphasis in this research is given to the parallel machine problem because most of the findings on reactive scheduling and rescheduling were tested on either a job shop or a flow shop, with very few papers addressing the parallel machine problems, which require a different rescheduling approach. Furthermore, up to our knowledge, no previous literature has discussed the generation of robust predictable or reactive schedules for unrelated parallel machines scenarios, which makes this dissertation innovative and clearly contributing to the body of knowledge.

Many performance measures were defined for the robust reactive scheduling; the most used one is bi-criteria, which minimizes both the makespan and the impact on schedule change (Wu *et al.*, 1991, 1993). Robustness is achieved by reducing the schedule variability from the predictable schedule (schedule change minimization), while ensuring at the same time an output that is close to the best or optimal solution (makespan minimization). This bi-criteria performance measure will be used in this research.

The scope of research that was presented in this chapter is illustrated and summarized in Figure 2. Bolded are the areas of research that are addressed in this dissertation.

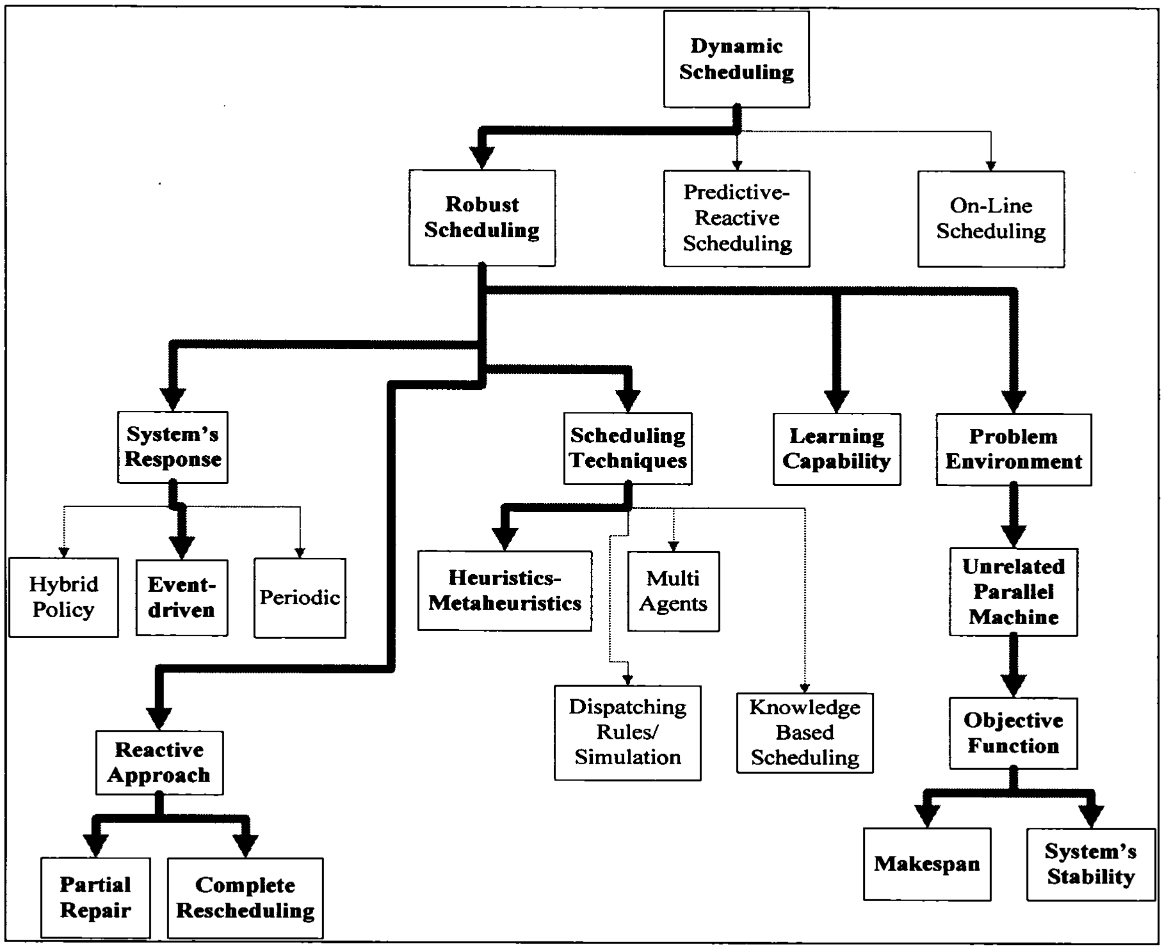

Figure 2. Scope of Research

PURPOSE OF THIS RESEARCH

The purpose of this research is to develop a robust scheduling system, which will be capable of coping with the new events through inherent rules and rescheduling in order to reduce the variability in the system and maintain the schedule's quality.

The mechanisms of the proposed system are described as follows:

- A robust predictable-reactive scheduling construct, which will react according to an event driven policy and attempt to overcome the perturbations using schedule repair as long as possible, otherwise it will use complete rescheduling.
- New and improved heuristics for scheduling repair and rescheduling in unrelated parallel machine environments.
- An objective and cost function that will aim at improving both the schedule's quality and stability.
- A schedule repair / rescheduling approach that can be applicable to different environments and not only the unrelated parallel machine environment.
- Finally, the proposed system will be equipped with a learning capability.

CHAPTER II

LITERATURE REVIEW

In this chapter, a review of the previous literature on the different areas of research (Figure 2) addressed in this dissertation is given.

The literature review is organized as follows. First, the literature on robust scheduling is summarized. Next, the learning research is addressed, followed by the literature on unrelated parallel machines. Finally, an indication of the gap in the literature that will be covered in the proposed research is presented.

ROBUST SCHEDULING

"Even though the need to create robust schedules was recognized over a decade ago by Graves (1981), from literature viewpoint there is no clear research explaining how a robust schedule can be generated in a dynamic environment" (Ouelhadj, 2003). Robustness is considered a concept that is not easy to measure or even define (Pinedo, 2002). A robust predictable-reactive schedule should ensure that the performance of the schedule remains high when subjected to disruptions and variability (Leon *et al.*, 1994). The robust schedule consists of two parts: predictable scheduling, and reactive scheduling (Mehta and Uzsoy, 1999; O'Donovan *et al.*, 1999).

Predictable Scheduling

Mehta and Uzsoy (1998) presented a predictable scheduling (PS) approach for a job shop with random machine breakdowns and an objective of minimizing L_{max}, where L_{max} is the maximum lateness across all jobs in terms of their completion time and their due-date. The authors presented two strategies for idle time insertion and reported that both heuristics did better than the traditional predictive-reactive schedule. O'Donovan *et al.* (1999) presented a PS for a single machine with breakdowns and an objective of minimizing tardiness between the predictable schedule and the realized schedule. Their idle time's insertion rule was similar to OSMH used by Mehta and Uzsoy (1998). Herroelen and Leus (2004) presented different measures for a robust pre-schedule in a project scheduling environment. They proposed a method that can be used in machine scheduling by assuming that 50% of the time each job on its execution will be delayed by 1 period and the other 50% by 2 periods. Hence, their schedule will spread out the disruptions over the schedule horizon, but it might lead to an overestimation of the total schedule completion time. Davenport *et al.* (2001) presented three slack-based techniques for creating the pre-schedule. Their paper considered a job shop with machine breakdowns, and an objective of minimizing the sum of job tardiness. Their techniques were mainly based on Mehta's OSMH rule. Up to our knowledge, no previous literature was found on creating predictable schedules for the unrelated parallel machine problem.

Reactive Scheduling

Reactive scheduling is a procedure to modify the created schedule during processing to adapt to changes in production environment (Sun and Xue, 2001). Abumaizar and Svetska (1997) proposed the affected operations algorithm (AOR) for the job shop problem with the objective of minimizing the makespan as well as the jobs' deviations. The authors reported that their repair heuristic (AOR) performed better than the right shift rescheduling strategy and complete rescheduling in almost all of the scheduling scenarios. Nof and Grant (1991) compared three types of recovery procedures, including rerouting, splitting orders and rescheduling, when disruptions occur. Their experiments showed that rescheduling is better than the others when there are machine breakdowns. Guo and Nonaka (1999) addressed rescheduling in a flow shop of three machines under machine failures scenarios and an objective of minimizing the completion time. They assumed that only one failure occurs at a time, and proposed a trigger value that once the disturbance time exceeds it, rescheduling would start. Akturk and Gorgulu (1999) proposed a match-up point to reschedule the pre-schedule in the case of machine breakdowns in a modified flow shop (MFS), with the objective of minimizing both tardiness and match-up point. The authors defined the match-up point as the schedule's point following a disruption, where the state reached by the revised schedule is the same as that reached by the initial schedule, and the preschedule can be followed again. It is advantageous to minimize the match-up point, i.e. the period of time where a new schedule is used instead of the preschedule, in order to ensure schedule's stability as the resources' planning was done according to the preschedule. After a machine breakdown, a match-up point for each machine is determined and a part of the initial schedule that covers the time interval between the disruption and the match-up point is rescheduled. The match-up point is created for all the affected machines by this disruption. After the match-up point, the pre-schedule is used again. The authors used Branch & Bound to solve their problem; they minimized two objectives: the tardiness of the jobs as well as the match-up point. The results obtained were satisfactory as the revisited schedule had less deviation, smaller match-up point and reduced computational time. Bean *et al.* (1991) proposed a "match-up" heuristic method for scheduling problems with disruptions. They showed that assuming enough idle time is present in the original schedule and disruptions are sufficiently spaced over time, the optimal rescheduling strategy is to match-up with the pre-schedule at some time in the future. Their algorithms were tested on a set of problems from an automobile manufacturer using tardiness as a performance measure. Alagoz and Azizoglu (2003) and Azizoglu and Alagoz (2005) addressed the rescheduling problem for identical parallel machines under machine eligibility restrictions subject to machines' breakdowns. Their objective was to reduce the total flow time of all jobs in the system and their stability measure was to reduce the number of jobs processed on different machines in the initial and revised schedules. They assumed that the times of the disruption as well as its duration are known. They proposed an LP model for the rescheduling problem of minimizing total flow time, and after they reduced the total flow time, they implemented a branch and bound for the problem of minimizing the number of disrupted jobs subject to the constraint that total flow time is kept to a minimum. The authors reported good results. No previous literature on rescheduling in unrelated parallel machines was found.

LEARNING CAPABILITY

The literature suggests that machine learning dates back to the mid-1950s when it was considered as part of the artificial intelligence (Langley, 1996; Michalski *et al.*, 1983). However, it did not become a distinct field until around 1980, when the first workshop on the topic occurred (Langley, 1996). Langley and Carbonell (1984, 1987), Dietterich (1989), and Michalski *et al.* (1983) presented the contributions of researchers in the machine learning field. Moreover, numerous conferences and workshops tackled this topic (European conference on Machine Learning, International Conference on Machine Learning). A machine learns whenever it changes its structure, program, or data (based on its inputs or in response to external information) in such a manner that its expected future performance improves (Nilsson, 1996).

UNRELATED PARALLEL MACHINES

The literature defines unrelated parallel machines as machines having different processing times for the same job (Liaw *et al.*, 2003). They are unrelated in the sense that the processing speed depends on the job being executed and not the machine; each job will have different processing times for each of the available machines (see Table 1). Previous research showed that even the identical parallel machine problem with only two machines is NP-hard when the objective function is the minimization of makespan (Garey and Johnson, 1979). Ghirardi and Potts (2005) considered the problem of scheduling jobs on unrelated parallel machines to minimize the makespan. The heuristic they used was an application of the recovering beam search. Weng *et al.* (2001) addressed the problem of scheduling a set of independent jobs on unrelated parallel machines with sequence dependent setup times so as to minimize the weighted mean completion time. They presented in their paper seven heuristic algorithms and tested them. In their algorithms, they either assigned a job to the machine with the least cost contribution, or to the machine on which the job has the shortest processing time. They also introduced an algorithm where they first assigned the job with the smallest ratio of processing time plus setup time to weight; this strategy outperformed the rest significantly. The authors claimed that their algorithms are extremely fast and can find solutions for up to 120 jobs and 12 machines in a fraction of a second. Low (2005) solved a multi-stage flow shop scheduling problem with unrelated parallel machines and an objective of minimizing total flow time in the system. A simulated annealing (SA)-based heuristic was proposed to solve the problem in a reasonable running time. Mosheiov and Sidney (2003) addressed the case of job-dependent learning curves and applied it to the problem of unrelated parallel machines with the objective of minimizing total flow time. Rabadi et al. (2006) addressed the same problem with sequence dependent setup times to minimize the makespan, where they introduced a new heuristic (Meta-RaPS) for the deterministic problem and compared it to an existing heuristic called the Partitioning Heuristic, which was introduced by Al-Salem (2004). The new heuristic outperformed the existing Partitioning Heuristic in almost all cases.

RESEARCH GAP

The previous literature clearly indicates the need for more robust predictable-reactive scheduling research and solutions, as no prior research describes a clear approach for the generation of robust scheduling systems in dynamic environments.

Moreover, the parallel machine environment lacks the appropriate recovery rules and strategies that currently exist in other environments. Most of the knowledge in this field has been limited to static deterministic scenarios, which have a great value in theory but can not be safely applied in practice due to its lack of consideration of the dynamic characteristics that are present in practical environments. In this research, we take the problem a step closer to practical applications.

Up to our knowledge, no published work was found on the generation of predictable schedules in parallel machine environments. Furthermore, most of the literature that addressed schedule repair and rescheduling strategies were designed for either a flow shop or a job shop, which require different recovery rules than the ones necessary for a parallel machine environment.

In addition, the research gap extends to an absence of publications tackling learning methods for predictable schedules, schedule repair, and rescheduling strategies in unrelated parallel machine environment.

Finally, no previous literature was found on designing a robust scheduling system that combines schedule repair, rescheduling, system's response, and learning in a parallel scheduling environment.

This dissertation addresses these research gaps and develops new and improved recovery rules and rescheduling policies for the dynamic parallel machine environment.

PROBLEM NOTATIONS

The key notations that will be used throughout the dissertation are shown below.

Notation	Definition
B	Breakdown occurrence time
C_i	Completion time of all the jobs scheduled on machine i, i.e. $C_i = \sum_{j=1}^{n} p_{ij} * x_{ij}$.
$Cmax_I$	Initial makespan without idle time
$Cmax_P$	Predictable makespan before schedule execution
$Cmax_R$	Realized makespan after schedule execution
D	Indicates the down machine
D_J	Indicates the job that needs to be rescheduled/fitted
e_{ij}	Efficiency of job j on machine i
ES_i	The earliest start of a job after the occurrence of a breakdown on a machine i
F_{ij}	Planned finish of job j on machine i
i	Machines index, i = 1, …, m
$idle_{ij}$	Idle time assigned to job j that will be processed on machine i
j	Jobs index, j = 1, …, n
J_i	The position of job that needs to be scheduled after the breakdown on up machine i.
J_D	The position of the job that needs to be scheduled after the breakdown on machine *D*, i.e. the position of the interrupted job
JP_{ik}	Indicates which job is in position k on machine i
k_i	Position index, i.e. indicates a job position on a machine i, k = 1, …, n
L_i	Location of ***B*** on machine i
LF_i	Latest finish of the rescheduled jobs on machine i.
m	Number of machines
MIncrease	Integer indicating the amount of jobs per machine to add to ResJobs in the *PR* rule
n	Number of jobs
N_i	Number of jobs assigned to machine i
$Path_i$	The new location on machine i if it processes D_J
p_{ij}	Processing time of job j on machine i
Q	Objective function $Q = Cmax_R - Cmax_P$
RC	The receiver machine of the down job
RE	Repair time required by a breakdown ***B***
$residle_i$	Idle time residue once D_J is fitted between two jobs on a machine i
ResJobs	The number of jobs that need to be rescheduled
RF_i	Repair finish on machine i
SD	The sender machine, i.e. the down machine
S_{ik}	Planned start of the k^{th} job on machine i
$Span_i$	Span of machine i, i.e. time to reschedule the jobs within.
Tidle	Total idle time in the system = $\sum_{i=1}^{m}\sum_{j=1}^{n} idle_{ij}$
x_{ij}	Binary decision variables = 1, if job j is assigned to machine i, 0 otherwise
LJ_i	Latest position of the job that will start after the breakdown on machine i

CHAPTER III

OPTIMAL SOLUTIONS FOR THE UNRELATED PARALLEL MACHINE PROBLEM USING INTEGER PROGRAMMING

The development of an Integer Program (IP) model for the unrelated parallel machine problem (R) with the objective of minimizing the makespan Cmax addressed in this research(R||Cmax) is crucial as the IP will be used to generate optimal initial schedules, and also when total rescheduling is necessary.

Several researchers formulated linear/integer program models for the unrelated parallel machine problem in order to obtain optimal solutions. Guinet (1991) and Rabadi *et al.* (2006) formulated mixed integer programs for this problem, but with added machine-dependent and job sequence-dependent setup times. Martello *et al.* (1997) presented in their paper a mixed integer program for the problem at hand, while Lawler and Labetoulle (1978) provided a linear program in order to attain near optimal solutions in a much faster computational time than the one required by an IP. Below we will explain both programs.

INTEGER PROGRAM

Let us first describe the integer program, recalling that our objective is to minimize the makespan Cmax, where Cmax = max $\{C_i\}$ (for i = 1,...,m), m is the number of machines, and C_i is the completion time of all jobs scheduled on machine i, i.e. $C_i = \sum_{j=1}^{n} p_{ij} * x_{ij}$.

Objective: Minimize Z

Subject to $\sum_{i=1}^{m} x_{ij} = 1$, for j = 1, ..., n, (C1)

$\sum_{j=1}^{n} p_{ij} * x_{ij} \leq z$, for i = 1, ..., m, (C2)

$x_{ij} \in \{0,1\}$, (i = 1,...,m; j = 1,...,n), (C3)

where,

Z: makespan C_{max}

p_{ij}: processing time of job j on machine i.

x_{ij}: binary decision variable = 1, if job j is assigned to machine i; 0 otherwise.

The objective is to minimize the makespan Z, which is also defined as a decision variable. Constraints (1) ensure that all the jobs will be assigned and each job will be assigned to only one machine. Constraints (2) guarantee that the completion time of jobs on each machine does not exceed the makespan.

In the case of the problem at hand, the above mixed integer program (referred to as MIP [1]) guarantees optimal solutions as long as the problem size is computationally feasible. Of course, the computation time increases dramatically as the problem size increases. Note that

MIP [1] assumes no disruptions, i.e. used in the static case of the problem. Nevertheless, it is indispensable in order to generate optimal solutions for the initial schedule, as well as for the total rescheduling scenarios.

Upper Bound

A function f is said to have an upper bound *UB* if $f(x) \leq UB$ for all x in its domain (Rowland and Weisstein, 2006a). The closer the *UB* is to the optimal solution of the problem, the better it is. It is very advantageous to use an *UB* for the MIP because it reduces the search space as the nodes that result in a solution worse than the *UB* will be eleminated. The new constraint that is added to MIP [1] in order to use the *UB* is as follows:

$$Z \leq UB \qquad \text{(C4)}$$

The value of the *UB* will be the feasible solution of the problem obtained using the algorithm provided by Davis and Jaffe (1981). Their algorithm is discussed below:

<u>Step 1</u>: {Sort the jobs in the non increasing order of their efficiency}

- Find for each job j its minimum processing time over all machines (ρ_j):

  $\rho_j = \min_{1 \leq i \leq m} p_{ij}$, for $j = 1, \ldots, n$.

- Find for each job j its efficiency on each machine i $(e_{(i,j)})$:

  $e_{(i,j)} = \rho_j / p_{(i,j)}$, for $i = 1, \ldots, m; j = 1, \ldots, n$.

- For each machine i, create a list of jobs j =1, …, n, sorted in the non increasing order of e(i, j).

Step 2: Assign the jobs to the machines such as sum_i is minimal, where sum_i is the sum of the processing times of jobs already assigned to machine i.

If any machine had no more jobs in its list or had a job with an efficiency $e_{(i,j)} < 1/\sqrt{m}$, this machine is marked as inactive (inefficient) and no more jobs will be assigned to it.

Step 3: The algorithm terminates when all the jobs are assigned.

The authors reported that their algorithm requires $n + m$ iterations, with a total running time of O($mn \log n$).

As an example, let us consider 8 jobs to be assigned on 3 unrelated parallel machines with the objective of minimizing the makespan. The jobs' processing times are shown in Table 2.

Table 2. Jobs' Processing Times

		Jobs							
		1	2	3	4	5	6	7	8
Machines	1	82	22	24	62	38	93	51	33
	2	2	93	56	59	60	48	31	49
	3	17	92	94	48	14	94	58	49

The first step in the algorithm is to sort the jobs in the non increasing order of their efficiency. To do this, ρ_j and $e_{(i,j)}$ are calculated and presented in Table 3, and the jobs are sorted in Table 4. Note that ties are broken arbitrarily.

Table 3. ρ_j and $e_{(i,j)}$ values

	Jobs							
	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**
ρ_j	2	22	24	48	14	48	31	33
$e_{(1,j)}$	0.02	1	1	0.77	0.368	0.516	0.608	1
$e_{(2,j)}$	1	0.237	0.429	0.81	0.233	1	1	0.6735
$e_{(3,j)}$	0.12	0.239	0.255	1	1	0.511	0.534	0.6735

Table 4. Sorted Jobs in the Decreasing Order of $e_{(i,j)}$

Machines	**Jobs Number**							
1	2	3	8	4	7	6	5	1
2	1	6	7	4	8	3	2	5
3	4	5	8	7	6	3	2	1

Next, the jobs are assigned to the machines such as sum_i is minimal. For example, the first jobs checked for assignment are job 2 on machine 1, job 1 on machine 2, and job 4 on machine 3; the algorithm will assign job 1 on machine 2 as it will result in the smallest sum_i over the machines (job 1 will be removed from the other machines' lists). Next, jobs 2, 6, and 4 are checked to be assigned respectively on machines 1, 2, and 3; the selected assignment is job 2 on machine 1 as again it will cause the minimal sum_i. The algorithm will continue in the same manner until all jobs have been assigned. During this, if any machine had no more jobs in its list or had a job with an efficiency $e_{(i,j)} < 1/\sqrt{m}$, this machine is marked as inactive (inefficient) and no more jobs will be assigned to it. Following this, the jobs assignment to the machines is presented in Table 5, where 1 indicates that a job is assigned to a machine, 0 otherwise.

Table 5. *Upper Bound* Jobs Assignment

		Jobs							
		1	2	3	4	5	6	7	8
Machines	1	0	1	1	0	0	0	0	1
	2	1	0	0	0	0	1	1	0
	3	0	0	0	1	1	0	0	0

The makespan obtained by the *Upper Bound* algorithm is 81; the optimal makespan is 79. As can be seen, the algorithm obtained a solution that is very close to the optimal (in this case, 2.53% from the optimal).

Mokotoff and Chrétienne (2002) also used the above algorithm as an *UB* when solving the same problem addressed in this research, i.e. $R||Cmax$.

LOWER BOUND

As was stated earlier, the unrelated parallel machine problem ($R||Cmax$) is NP-hard, meaning that the search for optimal solutions grows exponentially and in many cases may not be attainable in a feasible time. Therefore, a comparison with lower bounds (LBs) may be necessary as a way to evaluate the performance of the proposed rules. Moreover, the *LB* will be also used as a constraint in the MIP in an attempt to attain optimal solutions faster. Different LBs definitions are provided in the literature and they mainly state the following: "a lower bound is a function or growth rate below which solving a problem is impossible" (Algorithms and Theory of Computation Handbook, 1999). So if a function f is said to have a lower bound c, then $c \leq f(x)$ for all x's in its domain (Rowland and Weisstein, 2006b). As previously mentioned, the proposed scheduling system mainly consists of three stages: an initial schedule, a predictive schedule, and a reactive schedule. Our approach in developing lower bounds for the proposed system in the case of unrelated machines is as follows. Start by generating separate lower bounds for each stage of the system, then try to cluster these bounds together in order to serve the global system.

Initial Schedule Lower Bound

The initial schedule is essentially the deterministic schedule without the disruptions. Therefore, in this section we develop a *LB* for the unrelated parallel machines scheduling problem with the objective of minimizing the makespan.

Many researchers worked on this problem and some of them achieved good LBs. Grigoriev *et al.* (2005) generated a *LB* for the same problem but with an added resource constraint *K*, meaning that the jobs were not only machine dependent, but also resource dependent. In their paper, they suggested that a good *LB* could be the feasible solution of the relaxation of the problem's mixed integer program to a linear program. Many researchers used the same approach for generating LBs (Martello et al., 1997; Vredeveld and Hurkens, 2002). The mixed integer program for the problem on hand was presented in the beginning of this chapter (MIP [1]). A similar formulation was used by Martello *et al.* (1997).

MIP [1] can be transformed to a linear program (LP) by relaxing constraints (3), i.e. $x_{ij} \in \{0,1\}$ is replaced by $x_{ij} \geq 0$. Of course, x_{ij} will not be greater than 1 due to the restriction of constraints (1). In other words, constraints (3) will become:

$$0 \leq x_{ij} \leq 1 \qquad \text{(C3')}$$

Therefore, $\mathbf{Z_{LP}}$ (makespan when using linear programming) would be the lower bound $\boldsymbol{L_1}$. The reason for giving so much attention to using LP relaxations of IP models is that LP representations, unlike IP's, are generally easier to solve (Williams and Brailsford, 1996). In other words, optimal solutions for large problems, or even medium (unrelated machines) cannot be obtained using IP, as this process would be computationally infeasible (Sundararaghavan et al., 1997). The LP model does not always generate feasible solutions, because the decision variables are not binary when they actually should be, as they represent the assignment of jobs to machines, which should be either 1 or 0 (yes or no). However, the

LP model produces a solution that could be close to the optimal one generated by the IP (and sometimes the optimal) in a much shorter computational time. For example, when MIP [1] was solved using Lingo Solver for an instance of 2 machines and 90 jobs, it took around 16 seconds to reach the optimal solution (Cmax = 626 minutes); on the other hand, the LP was able to reach a very close makespan (Cmax = 622.74 minutes) in less than a second.

Vredeveld and Hurkens (2002) compared an LP relaxation similar to $\boldsymbol{L_1}$ to a modified LP relaxation and a convex quadratic program relaxation. The authors proved through computational tests the superiority of $\boldsymbol{L_1}$.

Another way for generating LBs was proposed by Costa et al. (2002). However, it was for the identical parallel machine problem, where the jobs' processing times are job dependent as opposed to being machine dependent. The authors suggested that a good *LB* would be the problem solution with the preemption constraint, i.e. allowing the jobs to be split on different machines, and the *LB* will be equal to the sum of all jobs processing times divided by the number of machines. Unfortunately, this *LB* cannot be applied to the unrelated parallel machine problem, because the jobs' processing times are dependent on their machines' assignments. However, one way to work around this is to actually determine for each job j = 1,…,n, its minimum processing time, ρ_j, over the machines i = 1,…,m, and since the total processing time cannot be less than $\sum_{j=1}^{n} \rho_j$, a valid lower bound would be $\boldsymbol{L_2} = \left\lceil \frac{1}{m} \sum_{j=1}^{n} \rho_j \right\rceil$, where $\lceil X \rceil$ is the ceiling of X (for example, $\lceil 10.1 \rceil = \lceil 10.9 \rceil = 11$). L_2 was also suggested by Martello et al. (1997) for the same problem at hand.

Martello *et al.* (1997) also pointed out that, as each job must be scheduled, a second obvious bound to this problem would be the maximum ρ_j (max{ρ_j}, for j = 1,…, n). So a valid lower bound is L_3 = **max (L_2, max {ρ_j}).**

In $L_2 = \left\lceil \frac{1}{m}\sum_{j=1}^{n}\rho_j \right\rceil$, the sum of the minimum jobs' processing times was divided over the number of machines, without actually assigning each minimum to its appropriate machine, i.e. where this minimum occurred. The reason behind this large problem relaxation is that assigning each job to its machine could lead to an unbalanced schedule. We could have problem instances where a machine is assigned more than one minimum processing time, resulting in empty machines on one hand, and a large makespan on the other. A better *LB* than L_2 could be the preemptive relaxation solution of the problem. Lawler and Labetoulle (1978) proved that the unrelated parallel machine problem with preemptive relaxation could be solved using the following LP:

Objective: Minimize Z

Subject to $\sum_{i=1}^{m} x_{ij} = 1, \ (j = 1,\ldots,n),$ (C1)

$\sum_{j=1}^{n} p_{ij} \times x_{ij} \le z, \ (i = 1,\ldots,m),$ (C2)

$0 \le x_{ij} \le 1$ (C3')

$\sum_{i=1}^{m} p_{ij} \times x_{ij} \le z, \ (j = 1,\ldots,n)$ (C4)

Constraints (1), (2), and (3') are as discussed above. Constraints (4) ensure that no split job is processed in parallel. We will refer to the *LB* obtained in this case as L_4.

Ghirardi and Potts (2005) suggested that a good *LB* for the unrelated parallel machine problem is the lagrangian relaxation of constraints (2) in the mixed integer model (presented above). However, Martello et al. (1997) proved in their paper that this lagrangian relaxation would lead to the same *LB* realized with the LP relaxation, i.e. L_1.

In summary, we described different LBs for the deterministic schedule in the unrelated parallel machine problem with the objective of minimizing the makespan. L_1, which was generated through the linear relaxation of the IP model of the problem, was reported to be a good lower bound. $L_2 = \left\lceil \frac{1}{m}\sum_{j=1}^{n} \rho_j \right\rceil$ was generated following the rationale that Costa et al. (2002) used in their identical parallel machine problem; however, this *LB* fails to account for job-machine assignments. L_4 was generated through the linear relaxation of the MIP of the problem subject to the preemption relaxation. Therefore, L_4 clearly is a better bound for our problem than L_2 (as it is always larger). Furthermore, L_4 will also attain better *LB*s than L_1 as it uses the same *LP* but with the extra constraint (4). Finally, a good *LB* to be used in our problem is L_4.

Predictable Schedule Lower Bound

As it was mentioned above, a predictable schedule is in fact a deterministic predictive schedule but with added idle time between the activities (jobs). The idle time will be inserted following the *Critical First Job Idle Time (CFJI)* rule (Equation (2) in Chapters 4 and 5) that was developed by Arnaout and Rabadi (2005). The authors reported that their rule significantly outperformed the popular Mehta et al. (1998) rule, *OSMH*, when the rules were compared in the unrelated parallel machine environment. *CFJI* inserts for each job the following idle time:

$$idle_{ij} = R_i * \delta_i * p_{ij} * (1 - \frac{k[i]}{Ji}) \quad (2)$$

where R_i is the mean rate of repair duration on machine *i*, δ_i is the average number of breakdowns on machine *i* per minute, k[i] is the job's position on machine *i*, and J_i is the total number of jobs that are scheduled on M_i. As can be seen from Equation (2), the idle time is directly related to the jobs' processing time. Furthermore, the predictable schedule will be generated right after the deterministic schedule; i.e. we know exactly where each job is located and how much its processing time is. This is why a *LB* for the predictable schedule is not needed, as we are only adding idle time to the jobs.

Reactive Schedule Lower Bound

We recall that the reactive schedule is the new schedule generated by the robust system while executing the predictable schedule if disruptions occur. The perfect scenario arises when absolutely no disturbances hit the system, and in this case, the solution would be the predictable schedule. This means, the predictable schedule is in fact a *LB* for the reactive schedule, because the latter solution could never be better than the predictable solution; it could either be the same (in case no disruptions occur) or bigger (disruptions leading to delays in the schedule).

MODELS VALIDATION

Williams (1999) suggests that a good approach to validate an integer/linear model is to convert this model into the format necessary to be tested on computer software. Therefore, in order to demonstrate the validity of the proposed approach, the above MIP was converted to the format required to be tested in Lingo Solver. Lingo is a tool provided by Lindo Systems, Inc. to solve linear, nonlinear, and integer optimization models (Schrage, 2001). Williams (1999) states that there are three possible outcomes in an IP's validation: a. the model is infeasible; b. the model is unbounded; c. the model is solvable.

Lingo Solver would indicate if a model is infeasible after checking it. The feasibility of the IP at hand was ensured when Lingo solver reported that it found the optimal solution.

Some models could be unbounded, i.e. the objective function can be optimized without a limit. However, we confirm that this is not the case in the tested model as again Lingo reported that an optimal solution was reached (instead of the message "Unbounded Solution").

If a model is neither infeasible nor unbounded, then a good solution is reached. Moreover, the optimality of the solution is confirmed by Lingo, which identifies the optimal solution if the model tested generates one. In fact, when Lingo reaches a global optimal solution, it reports the following: "Global Optimal Solution Found". On the other hand, if

Lingo finds a solution but does not guarantee that it is the global optimal one, it will then report: "Local Optimal Solution Found".

COMPUTATIONAL TESTS

MIP [1] presented above will be used to generate optimal initial schedules and will also be used when complete rescheduling is needed. After generating LB and UB in the previous sections, MIP [1] needs to be tested to determine when it will achieve the best performance: without an UB or LB, with an UB, or with both UB and LB. Following this, the different instances of MIP [1] were tested using Lingo with different number of machines (2, 4, 6, 8) and jobs (20, 40, 60, 80, 100) to ensure the validity of our decision. Each problem setting was run for 50 replications (*total = 20 × 50 = 1000 replications*). Moreover, the performance of MIP [1] was judged by the CPU time as well as the number of iterations required to reach the optimal solution.

The processing times of the jobs on different machines were generated randomly following the uniform distribution U[10,100] (Martello *et al.*, 1997). The reason a uniform distribution was used is due to its high variance, ensuring that the presented model is being tested under unfavorable conditions (Weng *et al.*, 2001). The tested problem instances are summarized in Table 6, where **CPU** refers to the average CPU time in seconds required to reach optimal solutions for all replicates, ***Iter*** refers to the average number of iterations needed for all replicates, and ***M*** and ***J*** refer respectively to Machines and Jobs.

Table 6. Computational Tests for MIP [1]

M	J	W/O UB, W/O LB		With UB, W/O LB		With UB & LB	
		CPU (sec)	Iter	CPU (sec)	Iter	CPU (sec)	Iter
2	20	0.17	41	0.18	41	0.24	61
	40	0.17	107	0.94	120	0.95	178
	60	0.245	162	0.92	195	0.99	303
	80	0.2	219	0.78	277	1.04	531
	100	0.23	241	0.26	261	0.77	521
4	20	0.59	1194	1.35	1156	1.61	1876
	40	1.6	5186	1.69	4656	2.64	6076
	60	2.94	8557	3.08	9630	3.6	13797
	80	3.58	15973	3.47	14367	6.9	20193
	100	8.49	28915	5.63	23490	16.72	53291
6	20	0.63	2583	0.68	2832	1.19	4331
	40	10.05	30848	12.87	38839	15.66	53839
	60	32.5	98285	22.71	78447	121.77	233218
	80	103.98	204816	87.5	206355	348.85	564586
	100	209.86	370011	99.33	283101	529.18	596226
8	20	1.2	3039	0.82	2070	1.31	3685
	40	21.81	51676	12.05	46732	39.91	113918
	60	270.31	553584	243.78	536784	458.81	955791
	80	823.47	1443105	445.3	1341065	1040.36	1690672
	100	1970.32	3484094	1587.63	2634369	4282.44	6468239

It is clear from Table 6 that the MIP with no LB or UB performed the best in terms of both CPU time and Iterations in small problems (of size up to 4 machines and 60 jobs), while the MIP with the UB performed better in the larger problems. Furthermore, the MIP with both UB and LB performed the worst in all problem settings. This behavior can be attributed to the fact that the solver now needs to iterate more in order to determine the feasible search region (between the UB and the LB). In addition, as MIP [1] with UB performed good but with UB and LB performed the worst, MIP [1] with LB will not be tested as obviously it will perform worse than MIP [1] with UB.

Figures 3-6 show how both the CPU and Iterations are the smallest when not using an UB in small problems, and with an UB for larger problems.

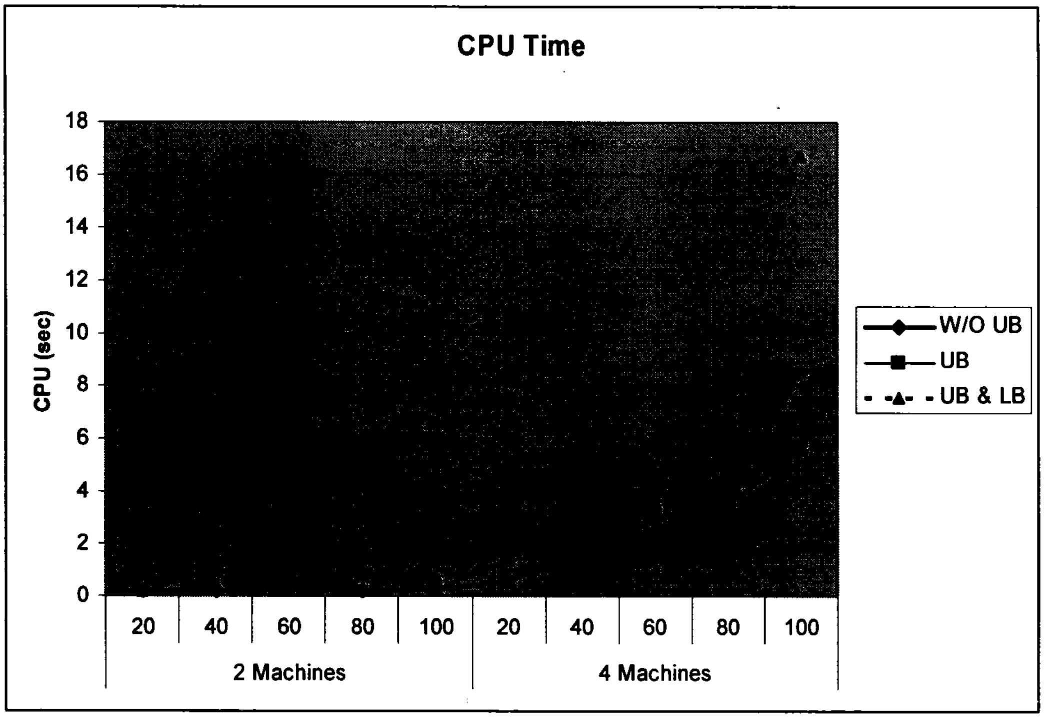

Figure 3. Average CPU Time comparison for 2 and 4 machines

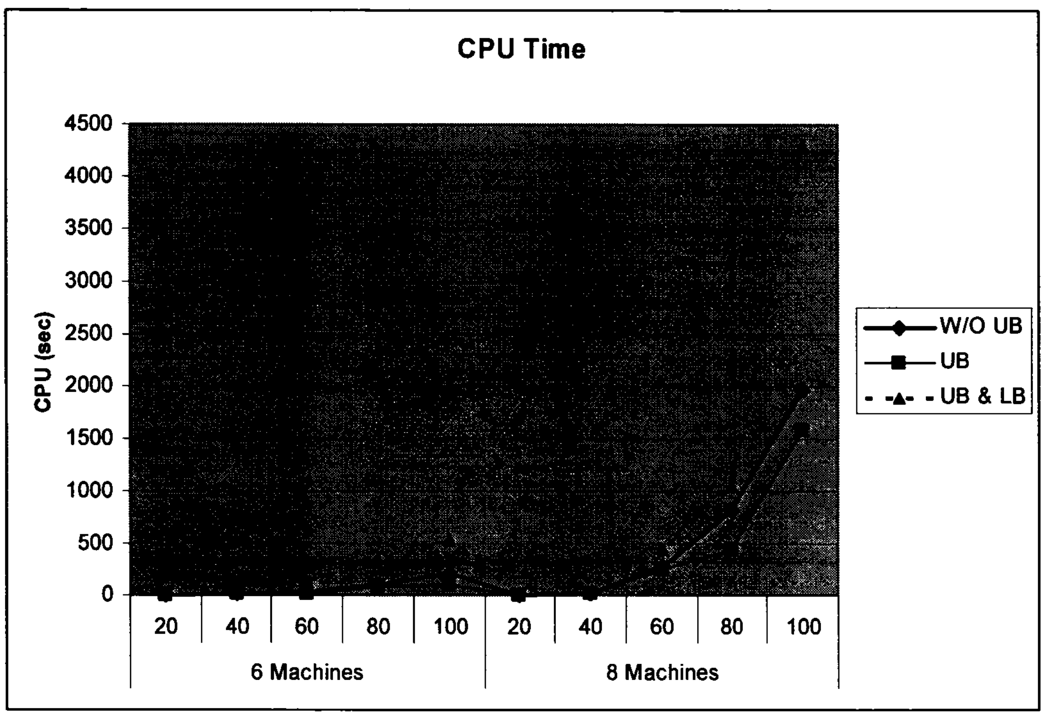

Figure 4. Average CPU Time comparison for 6 and 8 machines

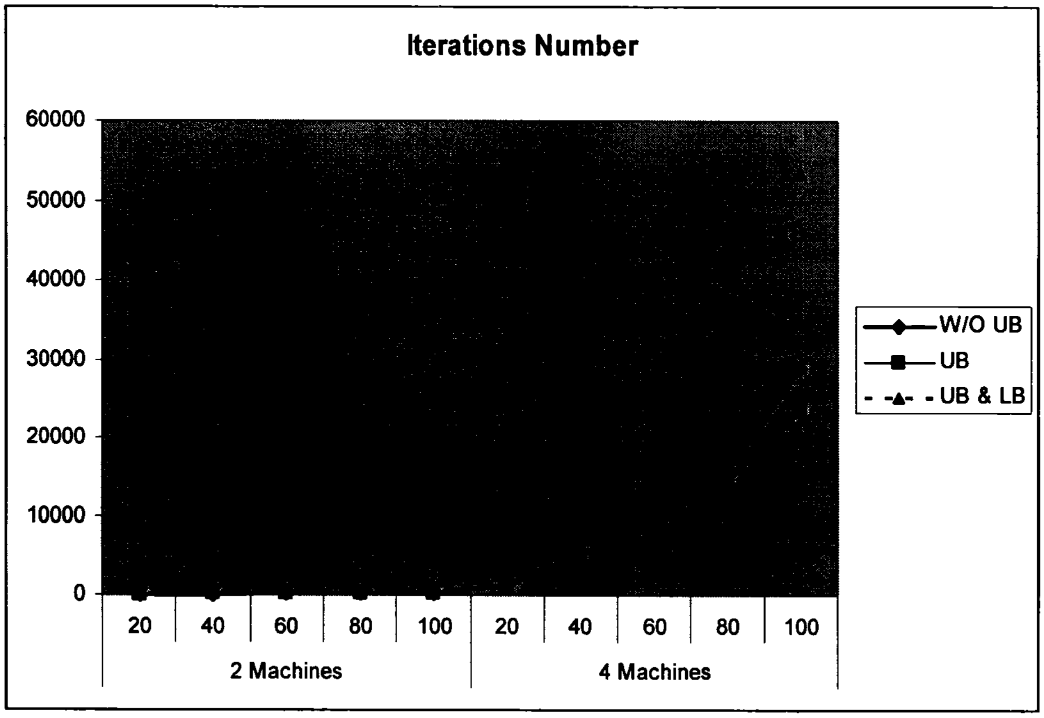

Figure 5. Average Iterations' Comparison for 2 and 4 machines

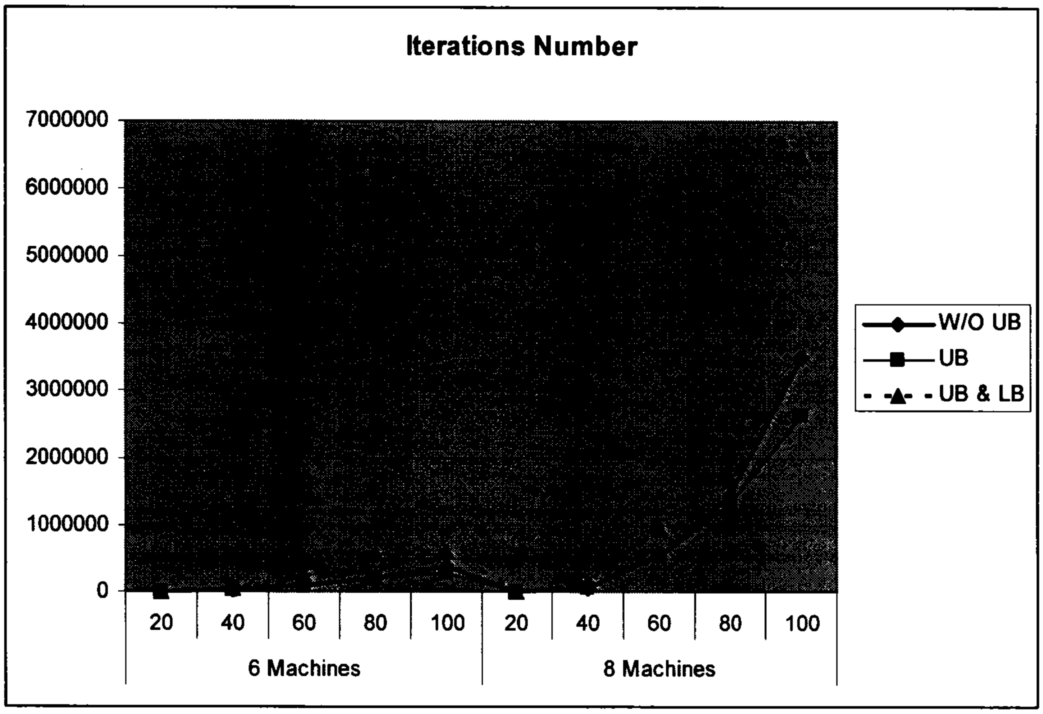

Figure 6. Average Iterations' Comparison for 6 and 8 machines

SUMMARY

In this chapter, MIP [1] that will be used to generate optimal initial schedules for the problem at hand was discussed. Furthermore, a LB and an UB were generated and tested with MIP[1] to determine in which case it will achieve the best performance: without an UB or LB, with an UB, or with both UB and LB.

From the computational tests, it can be concluded that including both *UB* and *LB* in the MIP will deteriorate its performance instead of improving it. Moreover, the MIP without an *UB* will be used to obtain optimal solutions for small size problems, while the MIP with an *UB* will be used in large size problems.

CHAPTER IV

PREDICTABLE SCHEDULING

Much research has been done in the field of scheduling, concentrating on deterministic scenarios and assuming a predefined schedule that is failure free. Unfortunately, most manufacturing and service environments are dynamic in nature, vulnerable to many unpredictable events, such as machine breakdowns, which leads the predefined schedule to become obsolete once it hits the shop floor (MacCarthy and Liu, 1993). Deterministic schedules produced in advance are followed hoping no delays will occur, because once they do, the whole schedule may be compromised, as it is not designed to incorporate change. The reason these deterministic schedules fail is because they do not account for variability by scheduling the activities directly after each other, so when a certain activity is delayed, all its successors will be delayed too. To overcome this shortfall, many researchers have suggested online scheduling, which is a completely reactive scheduling where no deterministic schedule is produced in advance, and decisions are made locally in real-time. One of the popular approaches in online scheduling are priority-dispatching rules, where whenever a machine becomes free, the available job with the highest priority is selected for processing. Dispatching rules are quick in general but inefficient and inaccurate because they typically do not use global information, and cannot guarantee that the system will operate at a good performance level (Ouelhadj, 2003). Furthermore, on-line scheduling is unable to provide any plans for other activities, and it is difficult to predict system performance because no initial schedule exists on which basis a scheduler can allocate resources and forecast performance. From here commenced the awareness of the importance of an initial schedule that will allow for preplanning and prediction.

Very few research papers dealt with generating robust pre-schedules, also called predictable schedules. Predictable scheduling is the process of making the predictive (deterministic) schedule robust enough to account as much as possible for unpredictable events. This is done through the insertion of idle time according to some rule between the scheduled jobs, so the disruptions can be smoothed out throughout the schedule.

In this chapter, a new rule for constructing robust schedules for the unrelated parallel machine problem is introduced and the computational results showing its dominance are reported.

PROBLEM FORMULATION AND ANALYSIS

This section describes respectively the problem at hand, the objective function that needs to be minimized, and the proposed rules.

Problem statement

The scheduling problem considered in this chapter is to schedule *n* jobs on *m* unrelated parallel machines. The problem constitutes of two parts: generating an initial schedule and making the schedule robust. The first part will be achieved using MIP [1] that was described in Chapter 3 of this dissertation, recalling that this problem is NP-hard as explained earlier. After generating the initial schedule, the second part of this problem consists of making this schedule robust enough to be able to absorb the disruptions. This is done through the insertion of idle time according to some rule between the scheduled jobs, so the disruptions can be smoothed out throughout the schedule.

The jobs' processing times are dependent on the machine they are assigned to; i.e. job *j* has a processing time p_{ij} when it is assigned to machine *i*. Our objective is to minimize the variability between the predictable and realized schedule makespans. This is represented as follows:

$$\text{Minimize } Z' = \frac{C\max_P - C\max_R}{C\max_R} \times 100\% \qquad (1)$$

where $Cmax_P$ is the makespan obtained from the predictable schedule, and $Cmax_R$ is the actual makespan from the realized schedule (i.e. the executed schedule with machine breakdowns).

Initial schedule (*Si*)

The problem objective is to compare different rules for idle time insertion within the *initial (Deterministic) schedule* so it becomes robust (*Predictable*) where MIP [1] described in Chapter 3 will be used to obtain optimal *initial schedules*. Once the *initial schedule (Si)* is generated, it will be compared to the predictable schedule, which is the same schedule but with added idle time. Recall that MIP [1] is described as follows:

Objective: Minimize Z

Subject to $\sum_{i=1}^{m} x_{ij} = 1$, for j = 1, …, n, (C1)

$\sum_{j=1}^{n} p_{ij} * x_{ij} \leq z$, for i = 1, …, m, (C2)

$x_{ij} \in \{0,1\}$, (i = 1,…,m; j = 1,…,n), (C3)

$Z \leq UB$ (C4)

where,

Z: the makespan ($Cmax_{Si}$) for schedule *Si*.

p_{ij}: processing time of job j on machine i.

x_{ij}: binary decision variables = 1, if job j is assigned to machine i; 0 otherwise.

UB: is an upper bound for the problem discussed in chapter 3.

In other words, MIP [1] will deliver the initial deterministic schedule S_i along with its makespan $Cmax_{Si}$.

Mehta's predictive rule

Mehta *et al.* (1998) presented a predictable scheduling (PS) approach for a job shop with random machine breakdowns and objective of minimizing L_{max}. One of the rules they proposed for inserting the idle time became quite popular in the robust scheduling domain and this is why it will be presented in this chapter and compared to the proposed rules. Their rule is called *OSMH* and it works as follows:

Step 1: generate a schedule without breakdown consideration (*Si*)

Step 2(OSMH): add to each operation of *Si* the associated idle time A_j as follows:

$A_j = E\ [DL_{ij}] = (p_{ij} * R_i)/\lambda_i$ where p_{ij} is the processing time of job *j* on machine *i*, R_i is the mean rate of repair duration on machine *i*, λ_i is the mean rate of breakdowns on machine *i*, and $E\ [DL_{ij}]$ is the expected delay of job *j* on machine *i*.

After updating *Si* with the idle time and generating the predictable schedule, its makespan will be calculated and referred to as $Cmax_{OSMH}$.

CFJI Insertion Rule

Kizilisik (1999) introduced a measure for idle time defined by $M1_j$ (PS) as the number of jobs critical to job *j* (succeeding job *j*), so the larger $M1_j$ (PS), the larger should the idle time inserted be. This is very logical because if the first job fails, it will lead to a delay in all its successors. On the other hand, if the last job fails, no successive impact will occur. Following this concept, we propose the *Critical First Job Idle Time (CFJI)* rule for inserting idle time, which will be similar to *OSMH* but with an addition of job position effect k[i]. The idle time $idle_{ij}$ for a job *j* on a machine *i* is calculated as follows:

$$idle_{ij} = R_i * \delta_i * p_{ij} * (1 - \frac{k[i]}{Ji}), \text{ for } J_i \geq 1 \tag{2}$$

where R_i is the mean rate of repair duration on machine *i*, δ_i is the average number of breakdowns on *i* per minute, k[i] is the job's position on machine *i*, and J_i is the total number of jobs that are scheduled on machine *i* (*note that δ_i in CFJI rule is different from λ_i in Mehta's rule as the latter was defined to be the mean rate of breakdowns on machine i; the calculation of δ_i is described in the next section).*

The associated makespan in this rule is referred to as $Cmax_{CFJI}$.

COMPUTATIONAL TESTS

The above rules have been implemented and compared in Microsoft Visual C++ 6.0 running on Windows XP with a Pentium 4 processor. The processing times of the jobs on different machines were generated randomly following the uniform distribution U[10,100]. Uniform distributions were used due to their high variances, ensuring that the rules are being tested under adverse conditions.

Once the predictable schedule is implemented, it will be subjected to machine breakdown events. Each machine will have its own breakdown rate, where the time between breakdowns (TBB_i) will follow an exponential distribution with mean $E[M_i]$ (Mehta *et al.*, 1998), where $E[M_i]$ is the expected processing time of a job on machine *i*. The average number of breakdowns per minute on machine *i* will be calculated as follows:

First we determine the number of breakdowns on machine *i*: (Total processing time on *i*) / TBB_i

$$\text{\# of breakdowns on } i = \frac{\sum_{j=1}^{Ni} p_{ij}}{TBB_i} \tag{3}$$

where N_i is the number of jobs assigned to machine *i*.

As δ_i is the average number of breakdowns on machine *i* per minute and is equal to:

$$\delta_i = \left(\frac{\text{\# of breakdowns on machine i}}{\text{Total processing time on machine i}} \right) \tag{4}$$

Substituting (3) in (4) $\Rightarrow$ $$\delta_i = \frac{\sum_{j=1}^{Ni} p_{ij} \Big/ TBB_i}{\sum_{j=1}^{Ni} p_{ij}} = \frac{1}{TBB_i}$$

$$\Rightarrow \quad \delta_i = \frac{1}{TBB_i}$$

The breakdowns' repair time on a specific machine follows a uniform distribution between $\beta_1 E[M_i]$ and $\beta_2 E[M_i]$, where we considered (β_1, β_2) to be (0.1,0.2) as in Mehta *et al.* (1998). The rules above have been tested under 2, 4, 6, and 8 unrelated parallel machines, and respectively 20, 40, 60, 80, and 100 jobs. The results obtained are shown in Table 7, where $Cmax_{Si}$, $Cmax_{OSMH}$, $Cmax_{CFJI}$, and $Cmax_R$ refer respectively to the predicted makespan of the initial schedule (S_i), OSMH rule, CFJI rule, and the realized makespan obtained after the occurrences of machines' breakdowns. The closer the predicted makespan to $Cmax_R$, the more robust the rule is.

Moreover, the 95% Confidence Interval (*CI*) attained from running $\cong$ 100 iterations of each rule was also included in Table 7. This *CI* was determined using Equation 4.5 that was described by Law and Kelton (2000) using the *t* distribution:

$$\overline{X} \pm t_{n-1,1-\alpha/2}\sqrt{\frac{S^2}{n}} \tag{5}$$

where $t_{n-1,1-\alpha/2}$ is the upper 1 - α/2 critical point for the *t* distribution with n-1 df, $\overline{X}$ is the mean, S is the standard deviation, and n is the sample size.

Table 7. Computational Tests for the Predictable Schedules

Machines	**Jobs**	**$Cmax_{SI}$ Mean**	**$Cmax_{SI}$ 95% *CI***	**$Cmax_{OSMH}$ Mean**	**$Cmax_{OSMH}$ 95% *CI***	**$Cmax_{CFJI}$ Mean**	**$Cmax_{CFJI}$ 95% *CI***	**$Cmax_{R}$ Mean**	**$Cmax_{R}$ 95% *CI***
2	20	409.12	[402.8-415.4]	566.98	[558.2-575.7]	481.99	[474.5-489.4]	492.45	[482.6-502.3]
	40	796.93	[784-809.9]	1104.47	[1086.5-1122.4]	946.84	[931.4-962.3]	957.71	[938.9-976.5]
	60	1205.49	[1186.3-1224.7]	1670.69	[1644-1697.3]	1437.56	[1414.4-1460.7]	1446.73	[1417.8-1475.6]
	80	1592.27	[1569.7-1614.9]	2206.73	[2175.4-2238.1]	1899.8	[1872-1927.6]	1882.43	[1850.8-1914.1]
	100	1971.88	[1940.8-2002.9]	2732.83	[2689.8-2775.8]	2355.17	[2317.9-2392.4]	2337.2	[2290.7-2383.6]
4	20	150.95	[148.7-153.2]	209.19	[206.04-212.36]	176.13	[173.4-178.9]	165.17	[162.2-168.1]
	40	284.42	[280.1-288.9]	394.31	[388.1-400.5]	337.05	[331.7-342.4]	310.54	[304.3-316.7]
	60	426.16	[418.9-433.4]	590.61	[580.6-600.6]	507.3	[498.6-515.9]	462.22	[453.8-470.6]
	80	554.71	[547.4-562]	768.78	[758.6-778.9]	661.87	[652.8-670.9]	599	[590.7-607.3]
	100	691.09	[679.8-703.3]	957.78	[940.8-974.8]	826.77	[812.7-840.8]	744.69	[730.5-758.9]
6	20	87.63	[86.4-88.9]	121.44	[119.7-123.2]	100.69	[99.2-102.1]	93.91	[92.2-95.6]
	40	158.58	[156.3-160.8]	219.77	[216.6-222.9]	186.94	[184.1-189.8]	167.14	[164.4-169.8]
	60	232.22	[228.1-236.3]	321.83	[316.1-327.5]	275.96	[271.1-280.9]	246.19	[241.3-251.1]
	80	307.88	[302.9-312.8]	426.69	[419.8-433.6]	366.54	[360.6-372.5]	328.08	[321.9-334.3]
	100	383.79	[377.2-390.3]	531.89	[522.8-540.9]	459.02	[450.9-467]	406.58	[399.5-413.6]
8	20	60.89	[60.2-61.6]	84.39	[83.4-85.4]	69	[68.1-69.9]	64.05	[63.1-64.9]
	40	107.67	[106.1-109.2]	149.23	[147-151.4]	126.57	[124.6-128.5]	113.01	[111.1-114.9]
	60	155.74	[153.3-158.2]	215.84	[212.4-219.2]	184.55	[181.6-187.5]	163.04	[160.1-165.9]
	80	205.19	[201.7-208.7]	284.37	[279.5-289.2]	243.99	[239.7-248.2]	216.25	[212-220.5]
	100	253.14	[248.8-257.5]	350.82	[344.8-356.9]	301.88	[296.6-307.1]	264.69	[259.9-269.5]

As 101 iterations were run for each problem setting (i.e. n = 101), then the confidence intervals will be:

$$\overline{X} \pm t_{100,0.975}\sqrt{\frac{S^2}{101}} \Rightarrow \overline{X} \pm 1.984\sqrt{\frac{S^2}{101}}$$

We recall that our objective is to minimize the variability between the predictable and realized schedules' makespans. Table 8 show the values of the objective function Z' (Equation 1):

$$Z' = \frac{C\max_P - C\max_R}{C\max_R} \times 100\% \qquad (1)$$

It is important to note that as Z' approaches zero, the more robust the rule is; a zero indicates that the predictive schedule has lead to a makespan equal to the realized schedule. Figures 7-11 show the percentage of variability from $C\max_R$ for S_i, *OSMH*, and *CFJI* for the 2, 4, 6, and 8 machines.

Table 8. Rules' Relative Deviation percent from $Cmax_R$

Machine	Job	$Cmax_{Si}$	$Cmax_{OSMH}$	$Cmax_{CFJI}$
2	20	-16.9215	15.134531	-2.12407
	40	-16.788	15.324054	-1.135
	60	-16.6748	15.480428	-0.63384
	80	-15.4141	17.227732	0.922743
	100	-15.6307	16.92752	0.768869
4	20	-8.60931	26.651329	6.635588
	40	-8.41115	26.975591	8.536742
	60	-7.80148	27.776816	9.752932
	80	-7.39399	28.343907	10.49583
	100	-7.19763	28.614591	11.02204
6	20	-6.68725	29.315302	7.219678
	40	-5.12146	31.488572	11.84636
	60	-5.67448	30.724237	12.09229
	80	-6.15703	30.056693	11.72275
	100	-5.60529	30.820503	12.89783
8	20	-4.93365	31.75644	7.728337
	40	-4.72525	32.050261	11.99894
	60	-4.47743	32.384691	13.19308
	80	-5.11445	31.500578	12.82775
	100	-4.3636	32.539952	14.0504

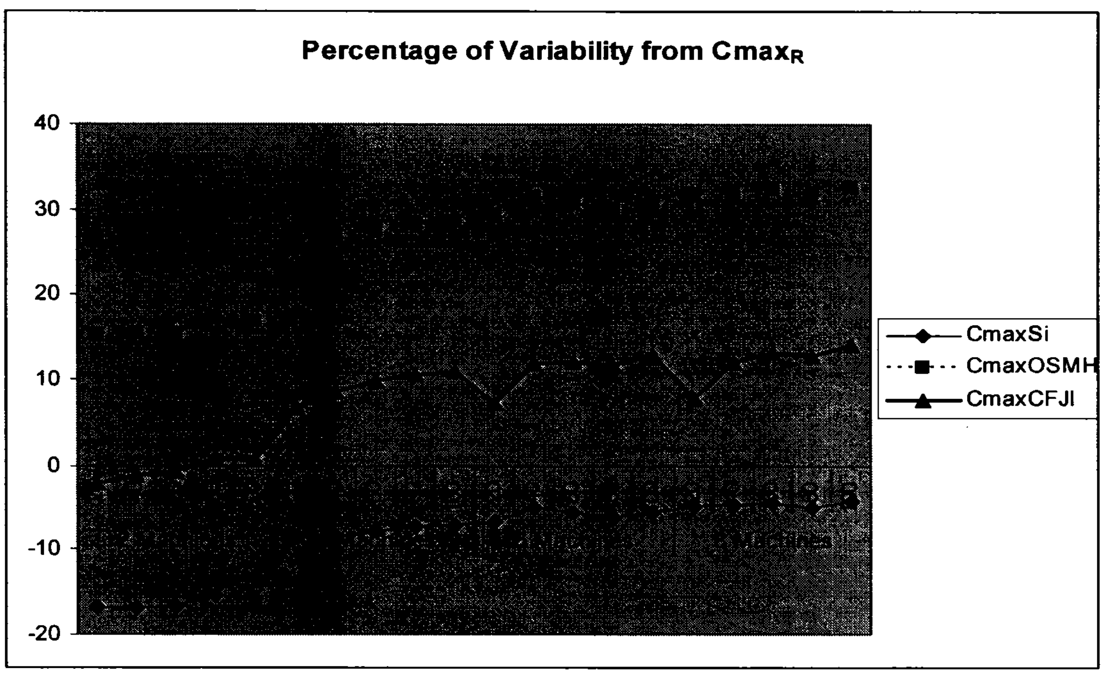

Figure 7. Relative Deviation percent from $Cmax_R$ for all machines

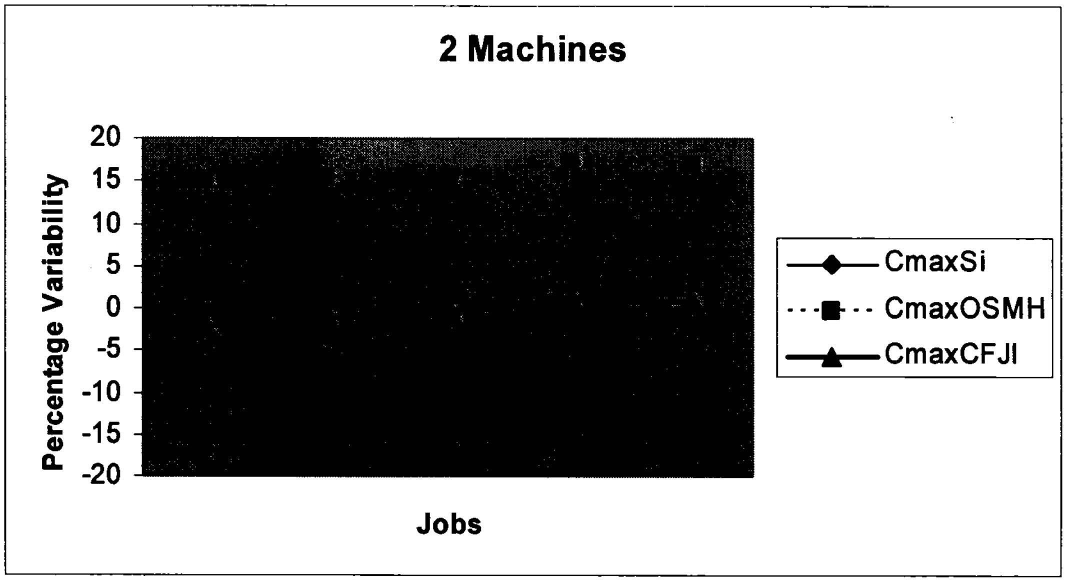

Figure 8. Relative Deviation percent from $Cmax_R$ for 2 machines

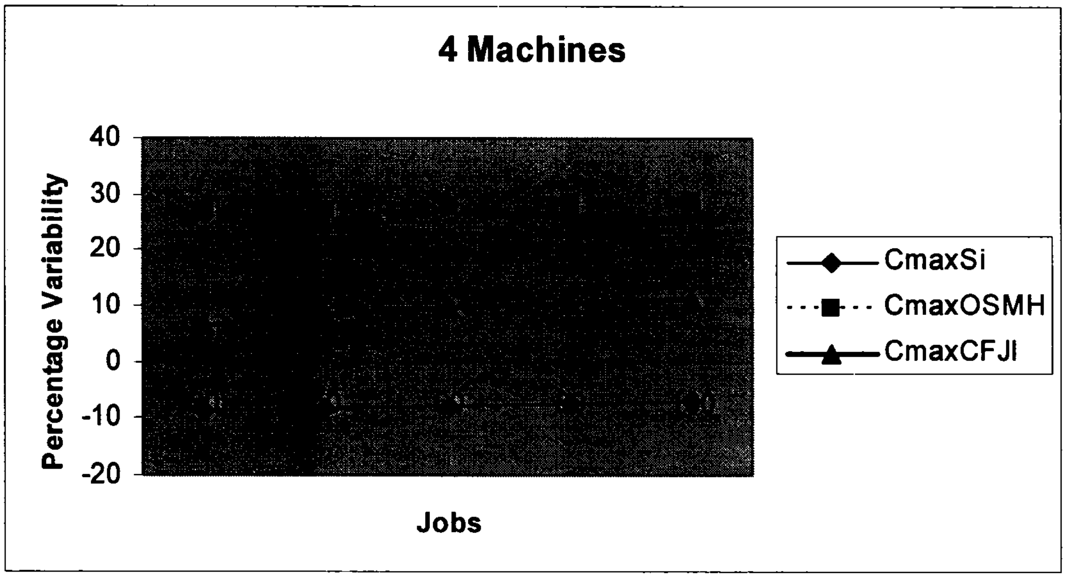

Figure 9. Relative Deviation percent from $Cmax_R$ for 4 machines

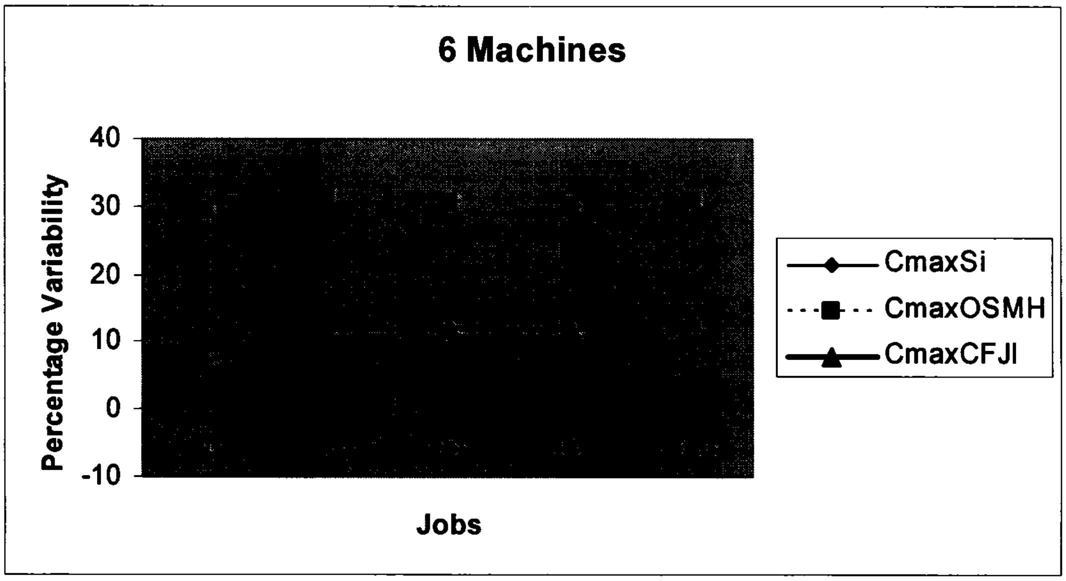

Figure 10. Relative Deviation percent from $Cmax_R$ for 6 machines

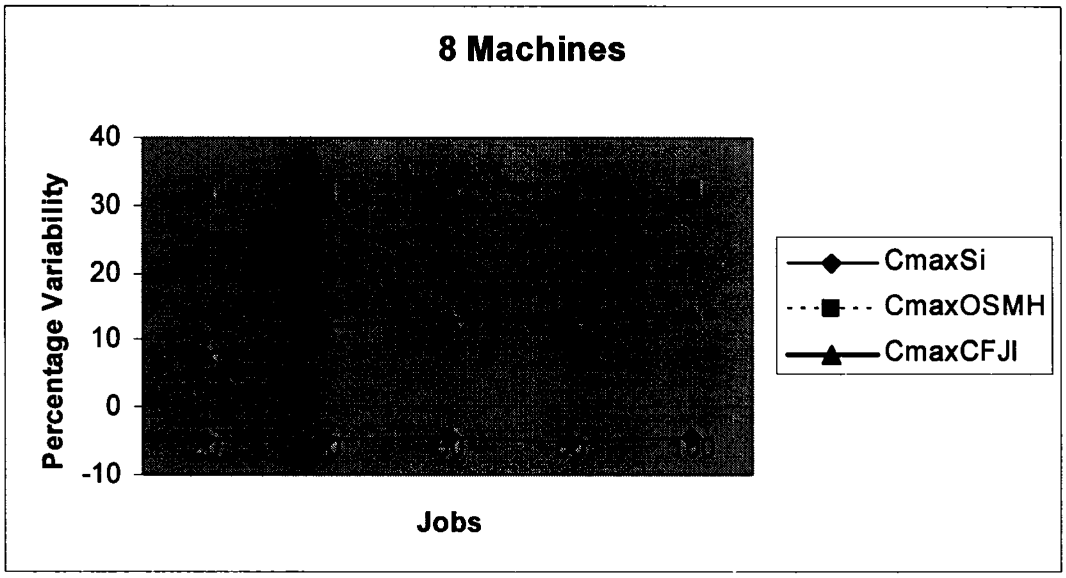

Figure 11. Relative Deviation percent from $Cmax_R$ for 8 machines

From Figures 7-11 and Tables 7 and 8, it can be concluded that the proposed rule *CFJI* outperformed *OSMH* in all problem combinations as it was always closer to the realized makespan. As it was expected, the *initial schedule* S_i has a makespan that is always smaller than the realized makespan as it does not account for machine breakdowns. *OSMH* performed better than the initial schedule in a sense that it was never below the realized makespan, but in many instances it overestimated the idle time needed to smooth out the breakdowns, leading to makespans that are far from the actual realized makespan ($Cmax_R$), resulting in an unstable pre-schedule. *CFJI* reflected high robustness and a good degree of schedule prediction.

For the 2 machines (Figure 8), *CFJI* almost overlapped with the realized schedule, when S_i predicted a much smaller makespan and *OSMH* a much higher one. For the 4, 6, and 8 machines (Figures 9-11), S_i had the closest prediction to $Cmax_R$; however, it was always smaller, i.e. such schedules will not be able to meet the set fourth deadlines. On the other hand, and even though *CFJI* was farthest than S_i from the realized makespan, it was higher than $Cmax_R$, meaning that the schedule execution finished before the deadline and not after. Furthermore, *CFJI* was still much closer to $Cmax_R$ than was *OSMH*.

SUMMARY

In this chapter, we have introduced a new idle time insertion rule, *CFJI*, for the generation of robust predictable schedules on unrelated parallel machines. *CFJI* was compared to the traditional initial schedule where no idle time is built-in, and to Mehta's rule *OSMH* from Mehta *et al.* (1998). All three rules were implemented in Microsoft Visual C++ 6.0 running on Windows XP with a Pentium 4 processor, and the conclusions were drawn using a large number of experiments and data instances. Computational tests showed that the introduced rule outperformed the other rules; however, as the problem size increased, *CFJI* overestimated the idle time needed for insertion. Following this, a learning parameter that will be added to *CFJI* is introduced in the next chapter in order to deliver superior predictable schedules.

CHAPTER V

LEARNING PARAMETER FOR THE PREDICTABLE SCHEDULE

In Chapter 4, a new idle time insertion rule (*CFJI*) was developed and compared to existing rules. Even though *CFJI* outperformed the other rules, it was clear that it overestimated the idle time needed especially as the problem size increased. Therefore, the system should learn to adjust its behavior, and thus, a learning parameter is developed in this chapter to aid *CFJI* reach more robust predictable schedules; i.e. schedules that are closer to the realized schedule.

Selfridge (1993) stated: "If an expert system, brilliantly designed, engineered and implemented, cannot learn not to repeat its mistakes, it is not as intelligent as a worm or a sea anemone or a kitten." He then followed: "Find a bug in a program, and fix it, and the program will work today. Show the program how to find and fix a bug, and the program will work forever." Machine learning studies the mechanisms through which intelligent systems improve their performance over time (Shavlik and Dietterich, 1990). Over the past decade, machine learning has evolved from a field of laboratory demonstrations to a field of significant commercial value. Machine-learning algorithms have now learned to detect credit card fraud by mining data on past transactions, learned to steer vehicles driving autonomously on public highways at 70 miles an hour, and learned the reading interests of many individuals to assemble personally customized electronic news (Mitchell, 1997).

As the proposed robust scheduling system will be dealing with a dynamic environment and to aid *CFJI* reach superior predictable schedules, a learning capability will be developed to ensure that the proposed rules stay up to date with the environment. Moreover, the system needs to learn from its mistakes so they would not occur again. Learning is essential because most of the machines' designs do not perform as intended when used in different environments. Even if a machine is used in its associated environment, the latter is subject to changes and consequently, the machine could perform poorly if no learning capability is incorporated.

MACHINE LEARNING FOUNDATIONS

Different subjects have contributed to the field of machine learning; below we describe some of the disciplines listed by Nilsson (1996):

- *Statistics*: estimation of the value of an unknown function at a new point given the value of this function at sample points. Statistical solutions of such estimations are considered a subset of the machine learning as the algorithms are learning the values of new points from previous samples in the same settings. More information on such methods can be found in Anderson (1958).
- *Brain Models*: different researchers (Gluck and Rumelhart, 1989; Sejnowski *et al.*, 1988) suggested modeling brains and networks based on nonlinear elements (neural networks).
- *Adaptive control theory*: used to estimate the changing parameters of a process during its operation. Bollinger and Duffie (1988) provide an introduction to this theory.
- *Artificial Intelligence*: AI has been concerned with machine learning since the 1950s (Langley, 1996). Researchers studied how future decisions can be based on previous ideal instances (Nilsson, 1996), and recent work has been aimed at generating rules for expert systems using decision tree methods and inductive logic programming.
- *Evolutionary Models*: Genetic algorithms and programming are considered a part of machine learning as they incorporate evolution through crossover and mutation in order to attain better performance levels.

LEARNING APPLICATIONS

Some of the machine learning' achievements that were summarized by Mitchell (1997) are listed as follows.

There are new programs that can effectively learn to recognize spoken words (Lee, 1989), detect fraudulent use of credit cards (Pomerleau, 1989), and play world-class backgammon (Tesauro, 1995). New research is founded on initial models of human and animal learning, as well as their relationship to learning algorithms developed for computers (Anderson, 1991; Ahn and Brewer, 1993).

Aytug *et al.* (1994) stated that a system should be able to correct its misconceptions and improve its performance based on experience; this is learning. Other researchers also acknowledged the necessity for learning in scheduling systems (Ow *et al.*, 1988; Fox and Smith, 1984).

Shaw *et al.* (1990) implemented a machine learning approach in order to perform intelligent scheduling and determine the most effective dispatching rule based on simulation runs. Simulation models have been frequently used as learning tools (Yih and Thesen, 1991; Adachi *et al.*, 1989; Davis and Smith, 1983).

PROPOSED LEARNING METHODOLOGY

Learning is needed in the proposed system for the idle time insertion when creating predictable schedules. The system should be capable of estimating the appropriate idle time to be inserted using results from previous problem iterations. For example, if prior iterations indicated an overestimation of idle time, then the system should readjust and insert less idle time.

A predictable schedule is generated by inserting idle time between the pre-schedule activities, enabling the disruptions to be smoothed out through the system in order to maintain the final output. The idle time will be inserted following the *Critical First Job Idle Time (CFJI)* rule that was discussed in chapter 4 of this dissertation. *CFJI* inserts for each job the following idle time:

$$idle_{ij} = R_i * \delta_i * p_{ij} * (1- \frac{k[i]}{J_i}), \text{ for } J_i \geq 1 \tag{2}$$

where R_i is the mean rate of repair duration on machine *i*, δ_i is the average number of breakdowns on *i* per minute, k[i] is the job's position on *i*, and J_i is the total number of jobs scheduled on machine *i*.

The need to make the system capable of learning and determining how much idle time should be inserted is crucial, and statistics will be used to achieve this aim.

The proposed system has the following objective function:

$$\text{Min } Q = \text{Cmax}_R - \text{Cmax}_P,$$

where,

Cmax_R is the realized makespan obtained after the occurrences of machine breakdowns.

Cmax_P is the predictable makespan generated using *CFJI* rule.

The closer the predictable makespan to Cmax_R, the more robust it is. In the case where Cmax_P is far from Cmax_R, the system will implement rescheduling techniques and schedule repairs in order to ensure a minimal Q. The learning purpose is to use rescheduling until the knowledge of the environment is robust enough to provide predictable schedules that almost overlap with the realized schedules, i.e. rescheduling would only be necessary in infrequent and severe situations. Through the learning parameter, the system will adjust the inserted idle time in order to minimize the deviation between the actual and predictable schedules.

The Learning Capability

The learning component will be incorporated by including in Equation (2) a parameter $\boldsymbol{\alpha}$ that will be adjusted during iterations to decide on the appropriate amount of idle time. For example,

$$\begin{cases} \text{if } Q > 0 \rightarrow Cmax_R > Cmax_P \rightarrow \text{increase } \boldsymbol{\alpha} \\ \text{if } Q < 0 \rightarrow Cmax_R < Cmax_P \rightarrow \text{decrease } \boldsymbol{\alpha} \end{cases}$$

If the predictable makespan was smaller (or larger) than the realized makespan, i.e. should be adjusted by Q, then the total idle time in the predictable schedule should be adjusted by m×Q, where m is the number of machines. Due to different jobs having machine-dependent processing times, it is not easy to predict which machine will result in the largest completion time, and thus the Cmax. Therefore, the rationale behind m×Q is to simplify the problem by adjusting the idle time inserted using one parameter only, $\boldsymbol{\alpha}$, for all the machines. Following this, we assume that the load is balanced over the m machines; i.e. the completion times of all the jobs scheduled on machine 1 through m are equal and the makespan is equal to the completion time of all jobs on any machine. This way the idle time would be equally divided among the machines, and as the idle time on one machine (that determined the makespan) should be adjusted by Q, the rest of the machines' idle time should also be adjusted by Q, i.e. the total idle time in the system should be adjusted by m×Q. This assumption is valid as our objective for the parallel machine problem is to minimize Cmax. The best solution that can be attained for such an objective is when all the machines finish at the same time (this is the best case scenario), i.e. the load is balanced over the machines.

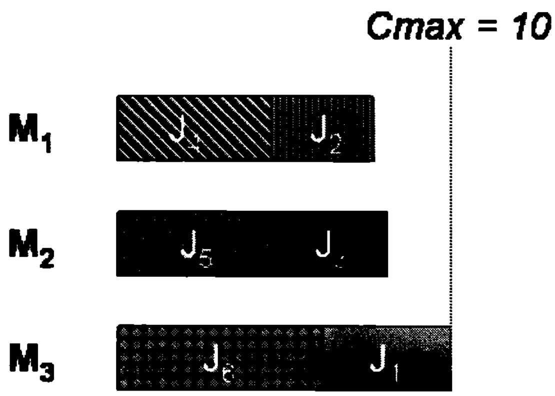

Figure 12.a. Cmax with Unbalanced Load

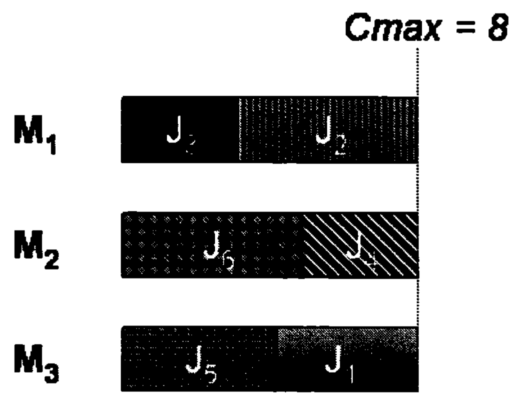

Figure 12.b. Cmax with balanced Load

In Figure 12, we illustrate how a balanced load over the machines leads to the smallest makespan possible. When the jobs are not balanced over the machines (Figure 12.a), Cmax is equal to 10. However, by balancing the jobs over the machines (Figure 12.b), all the machines will finish at the same time, leading to the smallest possible makespan (Cmax = 8). Following this, it is acceptable to assume that the MIP used to obtain initial schedules will attempt to balance the load over the machines as this will lead to the optimal solution.

The purpose of learning here is to estimate **α** such that **α** × existing idle time would be very close to the existing idle time + adjustment needed; this is given in Equations 6 and 6'.

$$\alpha \times \text{Tidle} = \text{Tidle} + (m \times Q) \tag{6}$$

$$\boldsymbol{\alpha} = 1 + \frac{m \times Q}{Tidle} \tag{6'}$$

where, Tidle is the total idle time in the schedule = $\sum_{i=1}^{m}\sum_{j=1}^{n} idle_{ij}$.

As $\alpha * Tidle = \alpha * \sum_{i=1}^{m}\sum_{j=1}^{n} idle_{ij} = \sum_{i=1}^{m}\sum_{j=1}^{n} \alpha * idle_{ij}$, Equation (2) becomes:

$$idle_{ij} = \boldsymbol{\alpha} * R_i * \delta_i * p_{ij} * (1 - \frac{k[i]}{J_i}), \text{ for } J_i \geq 1 \tag{2'}$$

and $\boldsymbol{\alpha}$ will be calculated using Equation 6'.

Determining the number of iterations for the learning parameter ($\boldsymbol{\alpha}$)

Changing $\boldsymbol{\alpha}$ for every iteration is unfavorable because it will result in big fluctuations in the system as can be illustrated in the following example:

The R||Cmax problem with 2 machines and 100 jobs was tested in Microsoft Visual C++ 6.0 running on Windows XP with a Pentium 4 processor. The code was designed in such a way that the program will keep on iterating while adjusting $\boldsymbol{\alpha}$ until Q, the predictable makespan deviation from the realized schedule, is less than 4 minutes. In other words, the program changes $\boldsymbol{\alpha}$ in every iteration in an attempt to find its finest value that minimizes Q. In the first problem instance, the program computed $Cmax_P$ (831 min), $Cmax_R$ (847 min), and Q (15.7 min), indicating that the system underestimated the realized schedule by 15.7 minutes.

Following this, via Equation 6', the program calculated the appropriate value of α that will increase the predictable schedule's makespan to a similar rate of the realized one (increase by 15.7 minutes). In the second problem instance, α successfully brought up $Cmax_P$ to 848 minutes (almost equal to the previous $Cmax_R = 847$ min); however, $Cmax_R$ for this instance was 821 minutes, resulting in a predictable overestimation of 27 minutes. The reason behind this is that the breakdowns follow an exponential distribution (vs. a constant one), i.e. the realized schedule is always fluctuating according to some distribution that needs to be determined.

In other words, in order to give a good estimate of α, we need to determine the realized schedule's distribution, then the required number of iterations k after which α can be updated.

Realized Schedule's Distribution

If we examine the realized schedule, the cause behind its fluctuations is due to the repair time. Every time a random breakdown occurs, a repair time (that follows a uniform distribution) is added to the realized schedule.

In order to understand the realized schedule's distribution, let us examine how the realized schedule is formed. At first, we start with an initial deterministic schedule S_i (can be assumed to have a constant duration), then idle time is inserted to S_i so it becomes a predictable schedule (assumed also to have a constant duration). Next, the predictable schedule is executed in a dynamic environment under machine breakdowns; this schedule will incur several delays (Repair time) of durations following the uniform distribution repair time. Finally, upon the completion of the execution of the predictable schedule, the latter plus the delays will constitute the realized schedule. In other words, it can be assumed that the realized schedule is nothing but the predictable schedule plus the repair delays:

$$\text{Realized Schedule} = \text{Predictable Schedule} + \text{Repair Delays}$$

$$\text{Realized Schedule} = C + U[\beta_1 E[M_i], \beta_2 E[M_i]]_1 + \ldots + U[\beta_1 E[M_i], \beta_2 E[M_i]]_t$$

where t is the number of breakdowns and C is a constant.

As the probability distribution of the sum of a sequence of uniformly distributed random values rapidly approaches that of a *normal distribution* as the number of values summed increases (Derbyshire, 2004), it can be concluded that the realized schedule makespan ($Cmax_R$) follows a normal distribution. Derbyshire (2004) also provided a graphical illustration of how the sum of uniform distributions will lead to a normal one (Figure 13), where the first graph represents the uniform distribution, and the subsequent graphs correspond to the cumulative addition of this distribution to itself up to n distributions. As can be seen, the sum of the uniform distributions approaches a normal distribution.

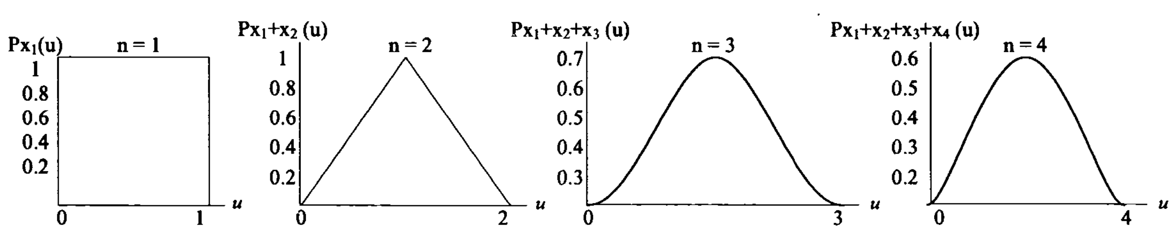

Figure 13. Graphical Illustration of the Sum of Uniform Distributions

In fact, even if the repair duration follows a different distribution (other than the uniform one), $Cmax_R$ will still follow a normal distribution because *The Central Limit Theorem* states that under very general conditions when *n* random variables (regardless of their distributions) are added together, the distribution of the sum tends towards the normal as *n* increases (Brignell, 2006); where *n* refers in this case to the number of breakdowns with durations equal to the repair time.

To further affirm that the $Cmax_R$ distribution is a normal one, we ran 1000 instances of the same input for the problem of 2 machines and 100 jobs, and then constructed a histogram from the data outputted by the program as shown in Figure 14.

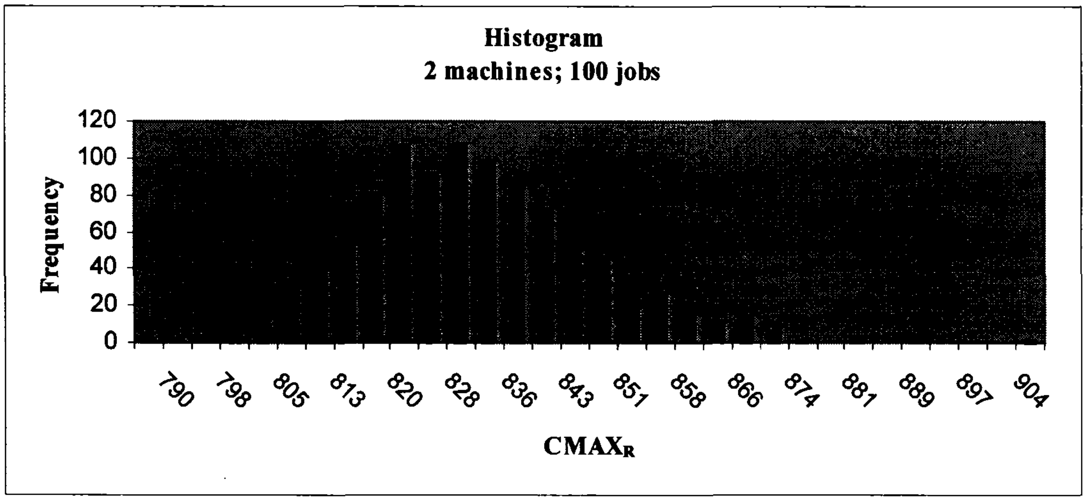

Figure 14. The $Cmax_R$ Distribution

As can be seen from Figure 14, $Cmax_R$ follows almost a normal distribution, indicating that the mean of *k* iterations can be used to calculate **α**.

Following the above, Equation 6' becomes:

$$\alpha = \frac{\sum_{u=1}^{K}\left(1+\frac{m*Q}{\text{Tidle}}\right)}{K} \tag{6''}$$

where, *u* refer to the problem iterations' index from 1 to *k* iterations.

Next, we run iterations until the half width of *CFJI* and Tidle is within 2% of the mean at most (at a 95% confidence interval). In other words, for each problem iteration, we calculate the mean and 95% confidence intervals, then we check, if the confidence intervals are far from the mean at 2% at the most, we stop and calculate α; otherwise, we run more problem iterations. Following this, the number of iterations *k* needed was on average 115, with a min of 34 and max of 315.

COMPUTATIONAL TESTS

In order to test the proposed learning parameter, the same experiments undergone in Chapter 4 were rerun with the addition of the α parameter to *CFJI* which we will refer to as *MCFJI (*Modified *CFJI). S_i, OSMH*, *MCFJI*, and *Realized* have been implemented and compared using Microsoft Visual C++ 6.0 running on Windows XP with a Pentium 4 processor. The processing times of the jobs on different machines were generated randomly following the uniform distribution U[10,100].

The results are shown in Table 9 along with the 95% Confidence Intervals.

We recall that our objective is to minimize the variability between the predictable and realized schedule makespans. Table 10 show the values of the objective function Z':

$$Z' = \frac{C\max_P - C\max_R}{C\max_R} \times 100\% \qquad (1)$$

Table 9. Computational Tests with a Learning Parameter

Machine	Job	$Cmax_{SI}$ Mean	$Cmax_{SI}$ 95% *CI*	$Cmax_{OSMH}$ Mean	$Cmax_{OSMH}$ 95% *CI*	$Cmax_{MCFJI}$ Mean	$Cmax_{MCFJI}$ 95% *CI*	$Cmax_{R}$ Mean	$Cmax_{R}$ 95% *CI*
	20	409.11	[402.8-415.4]	590.54	[581.4-599.7]	492.99	[485.3-500.6]	492.45	[482.6-502.3]
	40	796.93	[784-809.9]	1127.43	[1109.1-1145.8]	958.11	[942.5-973.7]	957.71	[938.9-976.5]
2	60	1205.49	[1186.3-1224.7]	1689.64	[1662.7-1716.6]	1447.06	[1423.8-1470.3]	1446.73	[1417.8-1475.6]
	80	1592.27	[1569.7-1614.9]	2171.1	[2140.3-2201.9]	1881.93	[1854.4-1909.4]	1882.43	[1850.8-1914.1]
	100	1971.88	[1940.8-2002.9]	2696.38	[2653.9-2738.8]	2299.87	[2299.9-2373.7]	2337.2	[2290.7-2383.6]
	20	150.95	[148.7-153.2]	180.53	[177.8-183.2]	163.47	[160.9-165.9]	165.17	[162.2-168.1]
	40	284.52	[280.1-288.9]	334.96	[329.7-340.2]	308.19	[303.4-313]	310.54	[304.3-316.7]
4	60	426.16	[418.9-433.4]	493.45	[485.1-501.8]	458.92	[451.1-466.7]	462.22	[453.8-470.6]
	80	554.71	[547.4-562]	635.53	[627.1-643.9]	594.67	[586.7-602.6]	599	[590.7-607.3]
	100	691.09	[678.8-703.3]	786.82	[772.8-800.8]	739.32	[726.5-752.1]	744.69	[730.5-758.9]
	20	87.63	[86.4-88.9]	121.44	[119.7-123.2]	92.12	[90.8-93.4]	93.91	[92.2-95.6]
	40	158.58	[156.3-160.8]	219.77	[216.6-222.9]	164.41	[162.1-166.8]	167.14	[164.4-169.8]
6	60	232.22	[228.1-236.3]	321.83	[316.1-327.5]	242.9	[238.6-247.2]	246.19	[241.3-251.1]
	80	307.88	[302.9-312.8]	426.69	[419.8-433.6]	324.45	[319.2-329.7]	328.08	[321.9-334.3]
	100	383.78	[377.2-390.3]	531.89	[522.8-540.9]	401.76	[394.9-408.6]	406.58	[399.5-413.6]
	20	60.88	[60.1-61.6]	84.38	[83.3-85.4]	62	[61.3-62.8]	64.16	[63.2-65.1]
	40	107.67	[106.1-109.2]	149.23	[147-151.4]	110.49	[108.8-112.1]	113.01	[111.1-114.9]
8	60	155.74	[153.3-158.2]	215.84	[212.4-219.2]	160.14	[157.6-162.6]	163.12	[160.2-166.1]
	80	204.16	[201.1-207.2]	282.95	[278.7-287.2]	212.12	[208.8-215.4]	214.58	[211-218.1]
	100	249.83	[245.5-254.2]	346.24	[340.2-352.2]	257.59	[253.1-262.1]	261.02	[256.2-265.8]

Table 10. Percentage of Variability of each rule from the Realized Schedule

Machine	Job	$Cmax_{Si}$	$Cmax_{OSMH}$	$Cmax_{MCFJI}$
2	20	-16.9235	19.91877348	0.109655803
	40	-16.788	17.72143968	0.041766297
	60	-16.6748	16.79027877	0.022810061
	80	-15.4141	15.33496597	-0.026561413
	100	-15.6307	15.36796166	-1.597210337
4	20	-8.60931	9.299509596	-1.029242599
	40	-8.37895	7.86372126	-0.756746313
	60	-7.80148	6.756522868	-0.71394574
	80	-7.39399	6.098497496	-0.722871452
	100	-7.19763	5.657387638	-0.721105426
6	20	-6.68725	29.31530188	-1.90608029
	40	-5.12146	31.48857245	-1.633361254
	60	-5.67448	30.72423738	-1.336366221
	80	-6.15703	30.05669349	-1.106437454
	100	-5.60775	30.82050273	-1.185498549
8	20	-5.11222	31.51496259	-3.366583541
	40	-4.72525	32.05026104	-2.22989116
	60	-4.52428	32.31976459	-1.82687592
	80	-4.856	31.86224252	-1.146425576
	100	-4.28703	32.64883917	-1.31407555

Figures 15-18 show the percentage of deviation of S_i, *OSMH*, and *MCFJI* from $Cmax_R$ for the 2, 4, 6, and 8 machines.

From Figures 15-18 and Tables 9 and 10, it can be concluded that the proposed rule *MCFJI* outperformed the traditional scheduling strategy (*initial schedule*) and also *OSMH*. As it was expected, the *initial schedule* had the worst robustness as it does not account for machine breakdowns. *OSMH* performed better than the initial schedule, but in many instances it overestimated the idle time needed to smooth out the breakdowns, resulting in makespans that are far from the actual realized makespan. *MCFJI* reflected high robustness and a good degree of schedule prediction.

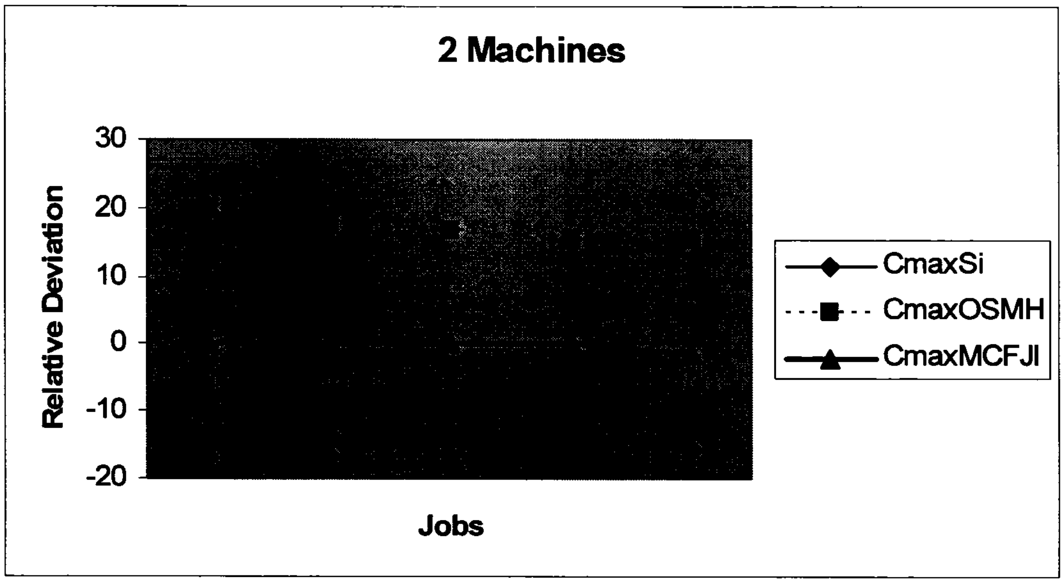

Figure 15. Relative Deviation percent from $Cmax_R$ (*0 on the Y-axis*) for 2 machines

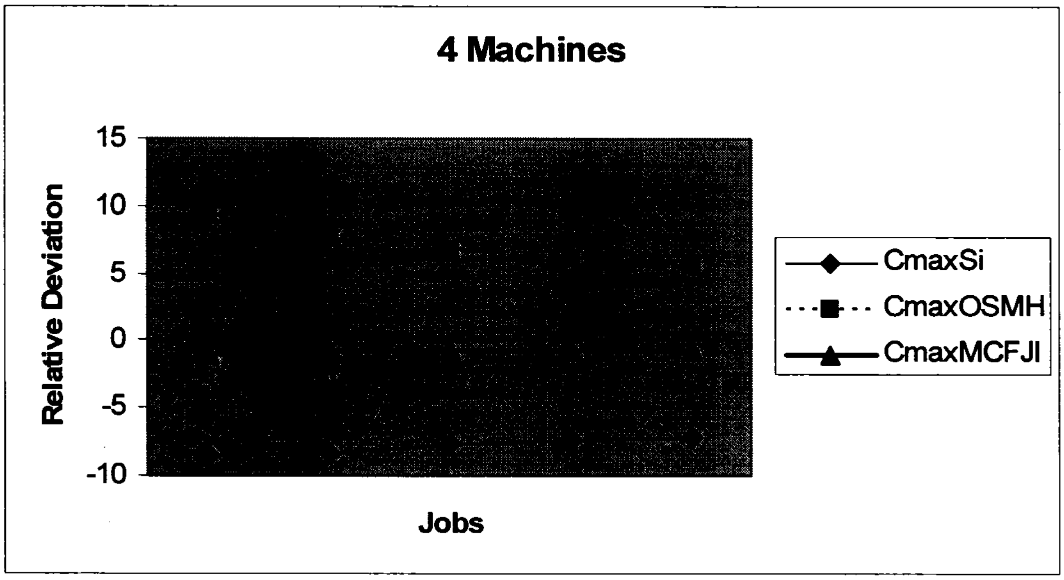

Figure 16. Relative Deviation percent from $Cmax_R$ (*0 on the Y-axis*) for 4 machines

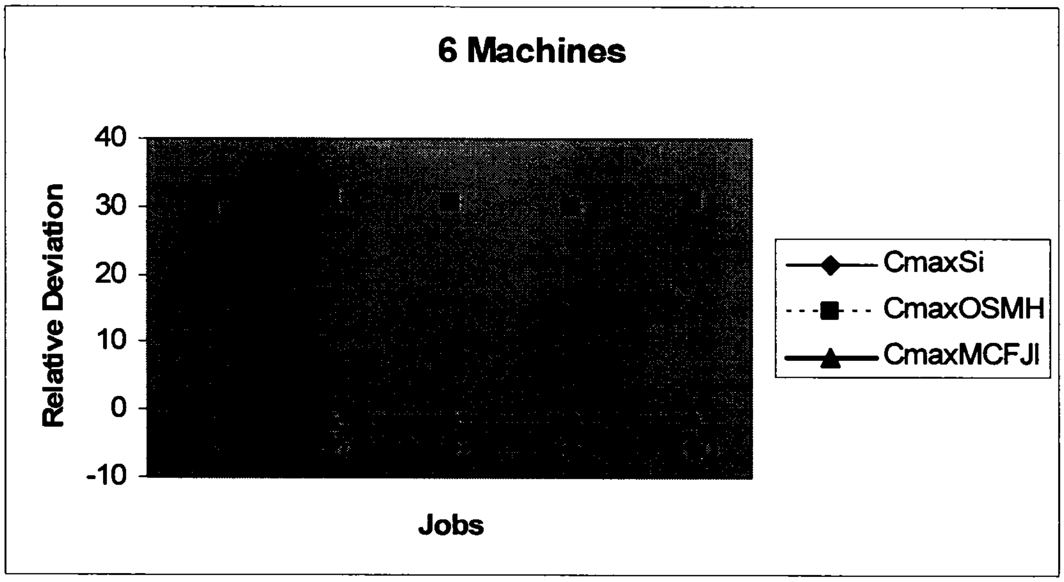

Figure 17. Relative Deviation percent from $Cmax_R$ (*0 on the Y-axis*) for 6 machines

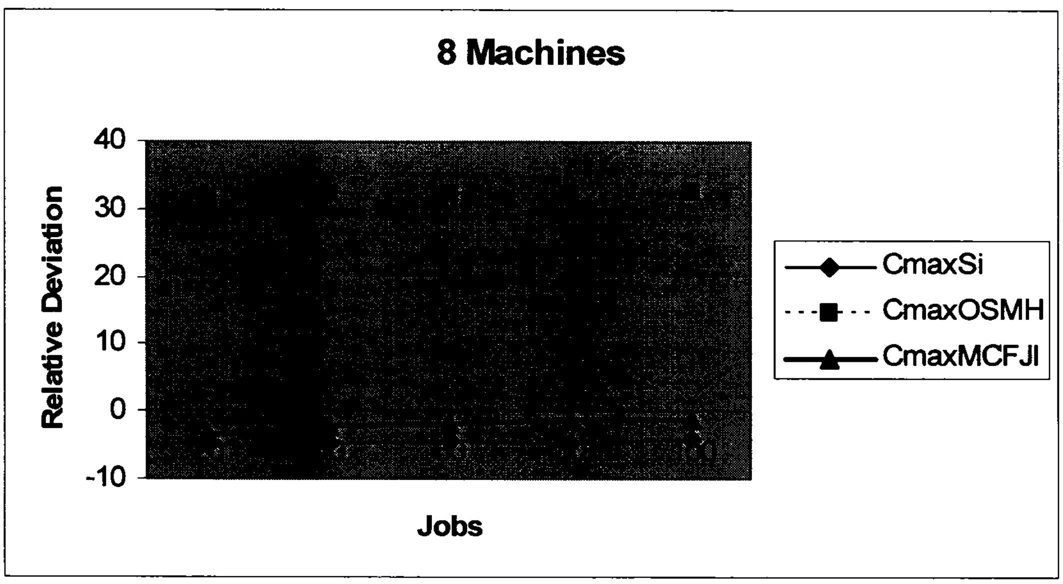

Figure 18. Relative Deviation percent from $Cmax_R$ (*0 on the Y-axis*) for 8 machines

As can be seen, *MCFJI* almost overlapped with the realized schedule, indicating a superior robustness, in contrast to *OSMH* and the initial schedule which were far from $Cmax_R$ and laid on opposite sides. It is also worth noting that even though the predictable schedule reached through *OSMH* is always larger than $Cmax_R$ (i.e. the schedule has the ability of smoothing out the breakdown effects without any delay in $Cmax_{OSMH}$), the idle time inserted is overestimated, leading to an underutilized pre-schedule.

SUMMARY

In this chapter, a learning parameter **α** was introduced for the idle time insertion rule *CFJI*. **α** readjusts the amount of idle time inserted in the schedule by using information from previous problem iterations. The computational tests indicate that this methodology will improve the performance of the proposed robust reactive scheduling system.

CHAPTER VI

REPAIR AND RESCHEDULING RULES

As previously explained in Chapter 1, a robust predictable-reactive scheduling construct will be implemented in this dissertation. The predictable schedule (*discussed in Chapters 4 and 5*) has the ability to absorb the disruptions without affecting planned external activities. If a disruption occurs during the schedule execution, repair rules and rescheduling will only be necessary if the disruption duration exceeds the inserted idle time. We recall that two main alternatives will be used for the reaction process: schedule repair and complete rescheduling.

Schedule repair refers to a minimum modification of the pre-schedule, leading to a higher stability in the system, while complete rescheduling refers to a complete rescheduling of all jobs, which could result in better solutions but will jeopardize system stability. Moreover, complete rescheduling will lead to system nervousness and could be very costly, as all the pre-arranged plans have to be changed. In practice, most rescheduling has been done using schedule repair, except in some severe situations where complete rescheduling had to be done (Abumaizar and Svestka, 1997).

In this chapter, new and existing repair rules and rescheduling strategies are explained and tested under extensive computational tests to determine superiority and dominance among them. These rules are respectively Right Shift Repair, Fit Job Repair, Partial Rescheduling, and Complete Rescheduling. The performance measures used to evaluate the rules are also explained.

PERFORMANCE MEASURES

The repair and rescheduling rules will be judged based on both the schedule quality and stability. The schedule quality is evaluated based on two performance measures: *Cmax Difference* and *CPU Time*. *Cmax Difference* refers to the difference between the realized and predictable schedules (i.e. *Cmax Difference* = $Cmax_R - Cmax_P$), and *CPU Time* refers to the time in seconds required by each rule during schedule execution. The schedule Stability is also assessed with two performance measures: *Match-up Time* and *Shifted Jobs*. *Match-up Time* refers to the time required by a rule to come back to the initial predictable schedule after a disruption, and *Shifted Jobs* refers to the number of jobs that will be shifted from one machine to another. The four performance measures are shown in Figure 19.

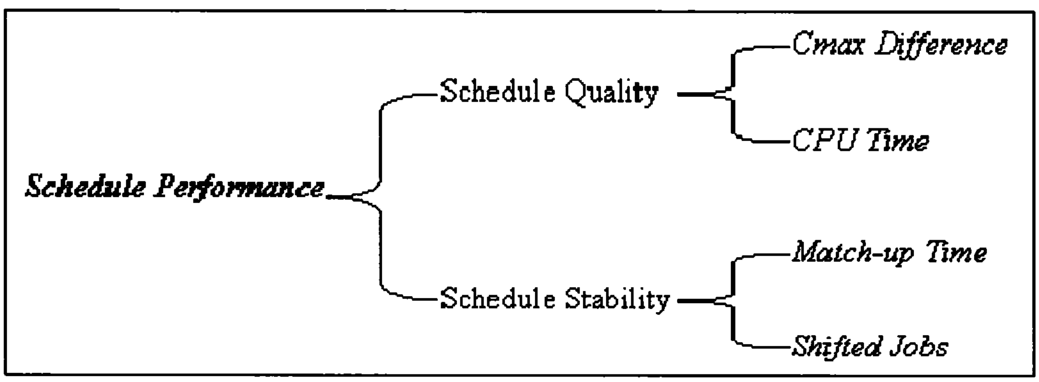

Figure 19. Repair and Rescheduling Rules' Performance Measures

Numerous publications used all or some of the above performance measures (e.g., Mendez and Cerda, 2004; Alagoz and Azizoglu, 2003; Raheja and Subramaniam, 2002; Akturk and Gorgulu, 1999).

RIGHT SHIFT REPAIR (*RSR*)

Right Shift Repair rule implies delaying the entire schedule by the disruption duration. In other words, whenever a disruption occurs, the disrupted operation and the activities succeeding it are shifted to the right by the amount of down time. This rule is similar to what an operator would instinctively do in the case of disruptions if no other strategies were in place. It should be noted that as preemption is not allowed, a disrupted job will have to be processed again from the beginning. *RSR* has been used frequently in the literature to compare with rescheduling and repair rules (Abumaizar, 1997; Akturk and Gorgulu, 1999; Bean *et al.*, 1991; Alagoz and Azizoglu, 2003).

The *RSR* algorithm is described below, where $S_{i,j}$ and $F_{i,j}$ refer respectively to the Start and Finish of job *j* on machine *i*, RF_i is the repair finish on machine *i*, *D* and D_J are respectively the down machine and down job, and N_i is the number of jobs scheduled on machine *i*.

If ($S_{D,D_J} < RF_D$) *//Apply RSR as D_J was scheduled to start before the repair finishes*

Step 1: {Calculate the new start and finish time for the interrupted job D_J}

- $S_{D,D_J} = RF_D$;
- $F_{D,D_J} = S_{D,D_J} + p_{D,D_J}$;

Step 2: {Update the start and finish of the remaining jobs on the down machine D}

- Let an integer Q = 0;
- while ($F_{D,D_J+Q} > S_{D,D_J+Q+1}$)

  {

  $S_{D,D_J+Q+1} = F_{D,D_J+Q}$;

  $F_{D,D_J+Q+1} = S_{D,D_J+Q+1} + p_{D,D_J+Q+1}$;

```
        If (D_J + Q + 1 = = N_D)
                Exit the while loop;
        else
                Q = Q + 1;
}
```

Note: As can be seen from the algorithm, this is at the most O(mn), assuming that time is incremented with integer values Q.

FIT JOB REPAIR (*FJR*)

FJR is a new repair rule introduced in this dissertation. The rationale of *FJR* is to fit the down job (D_J), i.e. the job that needs to be rescheduled, on a residle of any machine, where $residle_i$ refers to the remaining idle time on machine *i*.

In other words, when a breakdown occurs, *FJR* determines the down job (D_J) and the positions of the jobs not processed yet on the up machines (the machines that did not incur a disruption). Next, the residle on each machine is calculated and D_J is fitted on the machine *i* with the highest $residle_i$. In the case D_J does not fit in any residle, each machine's residle is increased by shifting to the right one job at a time. If after all jobs have been shifted ($residle_i$ cannot be increased anymore) D_J still cannot be fitted on any machine, then it will be assigned at the end of the machine that will result in the smallest makespan, i.e. where C_i is less than the completion time of the other machines.

FJR algorithm is described below, where $JP_{i,k}$ refers to the job in position *k* on machine *i*, J_D is the position of the interrupted job D_J, and $Path_i$ is the processing location on machine *i* if it is to process D_J.

Let tempS, tempF, tempJP be temporary arrays equal respectively to S, F, and JP

Step 1: Determine on each machine the jobs' locations (J_i) following a breakdown; also determine the ES_i on each machine, where ES_i is the earliest start of a job on machin *i* after the occurrence of a breakdown.
P.S.: In the case of *D*, $\mathbf{ES}_D = \mathbf{RF}_D$

Step 2: Determine the down job D_J, i.e. the job that needs to be rescheduled or fitted:

- $D_J = JP_{D,J_D}$

<u>Step 3</u>: Calculate the location on each machine assuming D_J will processed on it:

- $Path_i = ES_i + p_{i,DJ}$ $(i = 1, \ldots , m)$
- Let an integer variable Count = 0;

  While ((J_i + Count) <= N_i; for i = 1, …, m)

  {

  Check if the job can be fitted on any of the machines:

  If $(Path_i \le S_{i,J_{i+Count}})$ $(i = 1, \ldots , m)$

  {

  $residle_i = S_{i,J_{i+Count}} - Path_i$;

  Fit = **true** ; *//The job can be fitted*

  - Get the minimum fit cost over all the machines and determine the recipient machine *RC*

    MAX $(residle_i)$ $(i = 1, \ldots , m)$
  - Update *RC* by increasing N_{RC} and shifting the jobs to the right so $\boldsymbol{D_J}$ can be fitted.
  - Fit $\mathbf{D_J}$ in the receiver machine after the breakdown

    $S_{RC,J_{RC}} = ES_{RC}$;

    $F_{RC,JRC} = S_{RC,JRC} + p_{RC,DJ}$;

    $JP_{RC,JRC} = D_J$;

  If (Count > 0) *//Several jobs on RC need to be shifted for fitting*

  For (s = J_{RC}, …, J_{RC} + Count -1)

  $S_{RC,s+1} = F_{RC,s}$;

  $F_{RC,s+1} = S_{RC,s+1} + p_{RC,JPRC,s+1}$;

  - Update the sender machine *SD* by decreasing N_{SD} and shifting the jobs to the left as D_J location is available now.

  } *//end of IF*

  Else *// Need to shift more jobs in order to fit* $\boldsymbol{D_J}$

  {

  Count = Count +1;

  $Path_i = Path_i + p_{i,Ji+Count}$; (i =1, …, m)

  }

  } *//End of while*

<u>Step 4</u>: If D_J did not fit on any machine, assign it to the machine that minimizes Cmax

If ((J_i + Count) > N_i AND Fit = False)

{

$Path_i = 0$; $(i = 1, \ldots , m)$

$Path_i = ES_i + p_{i,DJ}$ $(i = 1, \ldots , m)$

$Path_i = Path_i + p_{i,JPi,j}$; (i = 1, ..., m ; j = Ji, ..., Ni)

Let *RC* be the machine with the minimum $Path_i$

- Update the receiver machine RC and the sender machine SD.

} *//End of IF*

STOP; once all jobs have been processed.

PARTIAL RESCHEDULING (*PR*)

Another rule introduced in this dissertation is the Partial Rescheduling rule. The rationale behind *PR* depends strongly on the concept of match up rescheduling. We recall that the match up strategy refers to trying to bring back the initial schedule as fast as possible once a perturbation occurs. Akturk and Gorgulu (1999) defined the match-up point as the schedule's point following a disruption, where the state reached by the revised schedule is the same as that reached by the initial schedule, and the pre-schedule can be followed again. It is advantageous to minimize the match-up point, i.e. the period of time where a new schedule is used instead of the pre-schedule, in order to ensure schedule's stability as the resource planning was done according to the pre-schedule.

Consequently, the strategy of *PR* is to minimize the match-up time so the initial schedule can be used for the execution. By coming back to the initial optimal schedule, the final makespan will remain the same, i.e. the best possible makespan.

PR works as follows: first, once a disruption occurs, *PR* will generate a pool of jobs for each machine (ResJobs_i) that need to be rescheduled in order to match up with the original schedule. The initial jobs included in the pool are the following:

- The down job D_J plus the value of the Match counter (*Match*) of succeeding jobs. For example, if *Match* = 0, only the down job will be added, if *Match* = 2, then D_J will be added plus the 2 jobs following it.
- For each of the up machines, the job that will start directly after the breakdown is added plus the Match counter of succeeding jobs. If the up machine was processing a

job when the breakdown occurred, it would continue processing that job as the disruption did not hit its assigned machine, and the succeeding job is added to $ResJobs_i$.

Next, the earliest start ES_i and latest finish LF_i on each machine are calculated. For the down machine *D*, the ES_D is the point where the repair finishes; and for the up machines, ES_i is the exact point in time when the job that was being processed during the breakdown finishes. The LF_i on machine *i* is the scheduled start S_{ik} of the job in the k^{th} position on machine *i*, where k in this case refers to the job right after the last one added to $ResJobs_i$. Following this, the span on each machine ($Span_i$) is calculated, where $Span_i$ is the time on machine *i* necessary to reschedule the jobs and is computed as $Span_i = LF_i - ES_i$. Now that we know the jobs that need to be rescheduled on each machine ($ResJobs_i$) and the minimum match-up time on each machine ($Span_i$), we will try to optimally solve for the number of jobs that will be shifted from one machine to another. The following MIP is used:

$$\text{Min } G = \sum_{i=1}^{m} \sum_{j=1}^{JobsNo} \left| XO_{ij} - X_{ij} \right|$$

Subject to:

$$\sum_{i=1}^{m} X_{ij} = 1, \text{ for } j = 1, \ldots, JobsNo, \qquad (C1)$$

$$\sum_{j=1}^{JobsNo} X_{ij} * p_{ij} \leq Span_i, \text{ for } i = 1, \ldots, m, \qquad (C2)$$

$$X_{ij} \in \{0,1\}, \ (i = 1,\ldots,m; j = 1,\ldots,JobsNo), \qquad (C3)$$

where,

G: objective function

p_{ij}: processing time of job *j* on machine *i*.

X_{ij}: binary decision variables = 1, if job *j* is assigned to machine *i*, 0 otherwise.

XO_{ij}: initial job machine assignment.

The objective is to minimize the total (XOij – Xij); i.e. the number of jobs that will change their position or shift from one machine to another. Constraints (1) ensure that all the jobs will be assigned and each job will be assigned to only one machine. Constraints (2) guarantee that the completion time of jobs on each machine does not exceed the Span. The disadvantage of the above MIP is that it is non linear as the objective function contains an absolute value. In order to change the optimization to a linear one, G is modified as follows:

Let X'ij = XOij – Xij, and let Yij ≥ |X'ij|, then G is replaced by:

$$G' = \sum_{i=1}^{m} \sum_{j=1}^{JobsNo} Y_{ij}$$

As X'ij ≤ Yij $\Rightarrow \begin{cases} X'_{ij} - Y_{ij} \le 0 \\ -X'_{ij} - Y_{ij} \le 0 \end{cases} \Rightarrow \begin{cases} XO_{ij} - X_{ij} - Y_{ij} \le 0 \\ -XO_{ij} + X_{ij} - Y_{ij} \le 0 \end{cases}$

In summary, the MIP that will minimize the number of shifted jobs is described below (referred to as MIP [2]):

$$\text{Min } G' = \sum_{i=1}^{m} \sum_{j=1}^{JobsNo} (Y_{ij})$$

Subject to:

$$\sum_{i=1}^{m} X_{ij} = 1, \text{ for } j = 1, \ldots, JobsNo, \quad \text{(C1)}$$

$$\sum_{j=1}^{JobsNo} X_{ij} * p_{ij} \leq Span_i, \text{ for } i = 1, \ldots, m, \quad \text{(C2)}$$

$$X_{ij} \in \{0,1\}, \ (i = 1,\ldots,m; j = 1,\ldots,JobsNo), \quad \text{(C3)}$$

$$XO_{ij} - X_{ij} - Y_{ij} \leq 0 \quad \text{(C4)}$$

$$-XO_{ij} + X_{ij} - Y_{ij} \leq 0 \quad \text{(C5)}$$

where,

G': new objective function

Constraints (4) and (5) replace the absolute value in order for the model to be linear.

MIP [2] was implemented in Lingo 9.0 from Lindo Systems. The schedule execution was in fact done in Microsoft Visual C++ 6.0 running on Windows XP with a Pentium 4 processor, and whenever *PR* needs to minimize the number of shifted jobs, it sends the necessary information to Lingo where it gets solved optimally (if it is possible) and the new job locations are sent back to the C++ program to continue executing the schedule.

In the case where no feasible solution can be found, i.e. MIP [2] cannot fit the jobs within the Span time allocated for each machine, *Match* is increased by 1 (one job is added from each machine to the pool $ResJobs_i$), then MIP [2] is run again.

Match will keep on increasing until a solution is found or no more jobs can be added to the pool. In the latter case, complete rescheduling will take place with the objective of minimizing both *Cmax Difference* and *Shifted Jobs*. This becomes a bicriteria optimization problem and two approaches exist in the literature to deal with such problems (Alagoz and Azizoglu. 2003): the *hierarchical approach*, i.e. minimizing the less important measure (*Shifted Jobs*) subject to the constraint that the more important measure (*Cmax Difference*) is kept at its optimum, and the *simultaneous approach*, i.e. generation of efficient schedules or optimization of a weighted combination of the two performance measures. Since we assume in this research that *Cmax Difference* is more important than *Shifted Jobs*, the hierarchical approach will be used in *PR* and *CR*.

Complete rescheduling works as follows: all the unprocessed jobs along with D_J are added to the pool ResJobs, then they are solved optimally using a MIP in order to minimize the makespan. In other words, as it is impossible to match up with the initial schedule, a new schedule will be generated for the remaining jobs that will reduce the makespan as much as possible. The MIP is described as follows (referred to as MIP [3]):

Min L

Subject to: $\sum_{i=1}^{m} X_{ij} = 1, \text{ for } j = 1, \ldots, \text{JobsNo}, \quad \text{(C1)}$

$$\sum_{j=1}^{\text{JobsNo}} (X_{ij} * p_{ij}) + ES_i \leq L, \text{ for } i = 1, \ldots, m, \quad \text{(C2)}$$

$$X_{ij} \in \{0,1\}, \ (i = 1,\ldots,m; j = 1,\ldots, \text{JobsNo}), \quad \text{(C3)}$$

where,

L: makespan $Cmax_R$

MIP [3] reshuffles the jobs in order to obtain the smallest $Cmax_R$ possible. However, as the number of shifted jobs is also a stability performance measure, we will attempt to reduce it. In fact, there could be several possible solutions for the same $Cmax_R$, and for this reason, an addition to MIP [3] is the following MIP [4], which will try to reduce *Shifted Jobs* while maintaining the same $Cmax_R$. There is no guarantee that MIP [4] will be able to minimize *Shifted Jobs* because it is constrained by an optimal $Cmax_R$.

$$\text{Min } G' = \sum_{i=1}^{m} \sum_{j=1}^{JobsNo} (Y_{ij})$$

Subject to:

$$\sum_{i=1}^{m} X_{ij} = 1, \text{ for } j = 1, \ldots, JobsNo, \qquad (1)$$

$$\sum_{j=1}^{JobsNo} (X_{ij} * p_{ij}) + ES_i \le L, \text{ for } i = 1, \ldots, m, \qquad (2)$$

$$X_{ij} \in \{0,1\}, \ (i = 1,\ldots,m; j = 1,\ldots,JobsNo), \qquad (3)$$

$$XO_{ij} - X_{ij} - Y_{ij} \le 0 \qquad (4)$$

$$-XO_{ij} + X_{ij} - Y_{ij} \le 0 \qquad (5)$$

where,

L: $Cmax_R$ obtained from MIP [3]

Following this, the above two MIPs guarantee an optimal $Cmax_R$ while minimizing *Shifted Jobs* whenever possible.

The *PR* algorithm is described as follows:

Let tempS, tempF, tempJP be temporary arrays equal respectively to S, F, and JP;
Let an integer variable *Match* = 0, and an integer variable array $ResJobs_i = 0$;
ProcJobs[][] is a double array used to send the jobs' processing time to Lingo.

Step 1: Determine on each machine the jobs' locations (J_i) following a breakdown; also determine the ES_i on each machine.
P.S.: In the case of *D*, $\mathbf{ES_D = RF_D}$

Step 2: Determine the number of jobs to be added to the rescheduling pool from each machine: *Match* = *Match* + *MIncrease*;

If (*Match* + Ji ≤ Ni) for any i = 1, …, m. *//we can still match with the preschedule*
{

$LF_i = S_{i,J_i+\text{Match}}$; (i = 1, …, m)

$Span_i = LF_i - ES_i$; (i = 1, …, m)

$$\left.\begin{array}{l}\text{JobsNo} = \text{JobsNo} + 1; \\ \text{ResJobs[JobsNo]} = \text{JP[i][J}_i + \text{j - 1]}; \\ \text{XO[i][JobsNo]} = 1;\end{array}\right\} \quad (\text{i} = 1,\ldots,\text{m}; \text{j} = 1,\ldots,\text{Match})$$

ProcJobs[i][j] = p[i][ResJobs[j]]; (i = 1,…,m; j = 1,…,JobsNo)

Solve to optimality using MIP [2];
If (optimal solution is found)
Update job-machine assignment and continue schedule execution;

Else
Go back to Step 2;

}
Else *//We ran out of jobs and still cannot match, i.e. start complete rescheduling*
{

$$\left.\begin{array}{l}\text{JobsNo} = \text{JobsNo} + 1; \\ \text{ResJobs[JobsNo]} = \text{JP[i][j]}; \\ \text{XO[i][JobsNo]} = 1;\end{array}\right\} \quad (\text{i} = 1,\ldots,\text{m}; \text{j} = \text{Ji},\ldots,\text{Ni})$$

ProcJobs[i][j] = p[i][ResJobs[j]]; (i = 1,…,m; j = 1,…,JobsNo)

Solve to optimality using MIP [3];
Try to reduce number of shifted jobs using MIP [4];
Update job-machine assignment and continue schedule execution;

}

STOP; once all jobs have been processed.

PR Design of Experiments

After describing the *PR* algorithm, one important question arises. Every time *PR* attempts to reschedule the Job Pool and fails to match with the initial schedule, the Job Pool is increased by *MIncrease* = 1 for each machine. But what if *MIncrease* was larger than 1, i.e. what if every time *PR* attempts to match, the Job Pool is increased by more than 1 job? It is important to note however that the larger *MIncrease*, the more *PR* approaches the Complete Rescheduling (*CR*), as the time to match up is being increased.

It is not reasonable to determine the appropriate *MIncrease* value by running replications of the same problem design (e.g. 4 machines and 20 jobs), as a single problem design is not sufficient to guarantee the best *MIncrease* for all problem combinations. Therefore, Design of Experiments (DoE) was used to determine the appropriate levels (parameters) of *MIncrease* that will contribute to better objective function values in the various problem configurations. Numerous publications provide a good review of DoE (e.g., Fisher, 1960; Taguchi, 1993; *NIST/SEMATECH* e-Handbook of Statistical Methods, 2006).

DoE Factors

The factors considered for the experiments along with their levels are shown in Table 11. Three levels were considered because non-linearity was suspected.

As can be seen, four factors are to be studied at three levels. Three-factor interactions (and above) are not considered as they are known to have usually weak effects (Ross, 1996); however, all 2nd degree interactions will be considered. Quadratic terms are to be analyzed as well. If we want to conduct a full factorial experimental design for four factors, we will need 3^k experiments, where 3 refers to the 3 levels that we want to analyze, and k refers to the number of main factors; this is a total of $3^4 = 81$ experiments. As it can be observed, this is a large number of trials given that for each experiment setting we will run 15 instances. Through a D-Optimal Design, we will be able to reduce this number dramatically.

Table 11. *PR* Design of Experiments Factors

Factor	Abbreviation	Value	Setting	Level
		[1, 50]	Low	-1
Processing time Range	**A**	[1, 100]	Medium	0
		[1, 150]	High	1
		20	Low	-1
Number of Jobs	**B**	60	Medium	0
		100	High	1
		2	Low	-1
Number of Machines	**C**	5	Medium	0
		8	High	1
		1	Low	-1
MIncrease	**D**	4	Medium	0
		7	High	1

D-optimal designs are typically generated by a computer algorithm and they are mainly useful when classical designs do not apply (*NIST/SEMATECH*, 2006). In the case of the *PR* Experimental Design, a D-Optimal design was generated because of the large number of experiments required by a classical one.

JMP 6.0 from SAS was used to generate the D-Optimal design, and the following Design Diagnostics were reported.

Table 12. *PR* D-Optimal Design Diagnostics

D Efficiency	76.40377
G Efficiency	100
A Efficiency	48.2308
Average Variance of Prediction	2.073364

The D-Optimal Design is presented in Table 13 and as can be seen, through DoE, we were able to reduce the number of experiments from 81 to 33 experiments.

For each of the 33 experiments, 15 replicates were run. The total number of replicates is 15 × 33 = 495. The following four performance measures are reported: *CPU*, *Cmax Difference*, *Shifted Jobs*, and *Match-up Time*. Moreover, the minimum and maximum values of each performance measure are included to indicate the variability in the results. The results for the 33 experiments are shown in Table 14.

Table 13. *PR* D-Optimal Design

Run	A	B	C	D	AB	AC	AD	BC	BD	CD	AA	BB	CC	DD
1	-1	-1	-1	1	1	1	-1	1	-1	-1	1	1	1	1
2	0	0	0	0	0	0	0	0	0	0	0	0	0	0
3	1	-1	1	-1	-1	1	-1	-1	1	-1	1	1	1	1
4	-1	-1	1	1	1	-1	-1	-1	-1	1	1	1	1	1
5	-1	0	0	1	0	0	-1	0	0	0	1	0	0	1
6	0	-1	0	1	0	0	0	0	-1	0	0	1	0	1
7	1	0	0	-1	0	0	-1	0	0	0	1	0	0	1
8	1	-1	-1	-1	-1	-1	-1	1	1	1	1	1	1	1
9	-1	1	1	1	-1	-1	-1	1	1	1	1	1	1	1
10	1	1	-1	1	1	-1	1	-1	1	-1	1	1	1	1
11	1	0	1	1	0	1	1	0	0	1	1	0	1	1
12	1	-1	-1	1	-1	-1	1	1	-1	-1	1	1	1	1
13	-1	-1	1	-1	1	-1	1	-1	1	-1	1	1	1	1
14	0	0	1	-1	0	0	0	0	0	-1	0	0	1	1
15	1	1	0	1	1	0	1	0	1	0	1	1	0	1
16	0	1	0	0	0	0	0	0	0	0	0	1	0	0
17	1	-1	0	0	-1	0	0	0	0	0	1	1	0	0
18	-1	-1	0	0	1	0	0	0	0	0	1	1	0	0
19	0	1	-1	-1	0	0	0	-1	-1	1	0	1	1	1
20	0	1	0	-1	0	0	0	0	-1	0	0	1	0	1
21	1	1	1	-1	1	1	-1	1	-1	-1	1	1	1	1
22	-1	1	0	-1	-1	0	1	0	-1	0	1	1	0	1
23	-1	0	1	0	0	-1	0	0	0	0	1	0	1	0
24	1	1	1	0	1	1	0	1	0	0	1	1	1	0
25	0	0	-1	1	0	0	0	0	0	-1	0	0	1	1
26	-1	1	-1	0	-1	1	0	-1	0	0	1	1	1	0
27	1	-1	0	-1	-1	0	-1	0	1	0	1	1	0	1
28	0	1	1	1	0	0	0	1	1	1	0	1	1	1
29	0	-1	1	-1	0	0	0	-1	1	-1	0	1	1	1
30	0	-1	-1	0	0	0	0	1	0	0	0	1	1	0
31	1	0	-1	0	0	-1	0	0	0	0	1	0	1	0
32	0	-1	1	0	0	0	0	-1	0	0	0	1	1	0
33	-1	0	-1	-1	0	1	1	0	0	1	1	0	1	1

Table 14. *PR* Rule's D-Optimal Design Results

	Avg. Cmax Difference (min)			Avg. CPU Time (sec)			Avg. # Shifted Jobs			Avg. Match-up Time (min)		
Run	**Min**	**Avg**	**Max**	**Min**	**Avg**	**Max.**	**Min**	**Avg**	**Max**	**Min**	**Avg**	**Max**
1	7.92	49.34	99.25	1.53	3.43	6.1	2	11.45	26	476.88	1115.24	2342.01
2	-11.28	28.05	80.34	2.59	21.03	52.55	3	44.3	82	286.42	1483.28	2571.76
3	-7.52	15.4	39.13	0	1.67	5	0	5.15	22	0	149.83	592.45
4	-0.83	5.1	13.93	0	2.57	7.86	0	6.6	20	0	66.9	192.86
5	3.7	14.02	26.02	10.92	25.22	48.48	15	36.93	61	334.74	805.4	1364.02
6	-14.75	18.97	47.53	0	2.3864	6.36	0	6.8	25	0	235.52	667.75
7	-2.19	33.48	102.2	1.44	31.84	94.61	0	43.35	124	321.09	1798.54	3472.56
8	40.59	158.56	305.6	3	13.73	34.72	0	12.6	29	876.28	3149.75	7648.29
9	-1.62	6.66	18.21	47.89	348.27	1778	25	68.4	156	329.87	599.824	1090.04
10	423.24	795.34	1405	38.59	75.87	145.8	65	264.8	518	33324	70605.1	135537
11	-8.14	14.24	46.15	2.73	21.4	53.36	4	26.13	48	143.74	580.92	1235.36
12	40.6	157.67	268.5	1.56	3.83	7.2	2	12.47	26	876.29	3952.06	8780.64
13	0	5.1	13.93	0.02	4.09	12.56	0	6.6	20	0	66.9	192.86
14	-4.3	10.23	34.47	1.28	47.75	276.2	0	22.8	62	24	350.1	730.28
15	-13.88	45.77	145	6.84	50	109.7	38	85.8	162	2363.6	5217.4	7944.14
16	-6.18	32.74	52.77	9.56	55.84	118.4	34	100.6	169	1607.2	3271.71	5056.64
17	-15.05	22.5	67.27	0.015	3.58	6.98	0	5.47	16	0	357.23	812.56
18	-3.98	5.12	20.39	0.45	2.5	7.22	0	6.13	14	30.98	115.88	302.1
19	351.04	514.93	711.2	53.67	179.7	281.8	313	742.1	1630	16838	41192.1	54648.6
20	-12.25	33.78	95.94	7.59	73.07	212.4	9	120.5	277	609.81	2534.72	5898.8
21	-10.03	16	67.14	3.12	118.03	525.7	15	71.87	137	391.88	1421.09	2597.84
22	2.54	19.57	43.94	1.97	51.37	151.3	3	154.2	389	119.45	1365.81	3214.89
23	-2.65	7.49	19.07	11.23	57.02	274.1	17	34.13	59	84.1	267.06	537.57
24	-13.86	13.93	77.35	7.67	323.74	1313	25	79.6	143	731.07	1916.14	3140.31
25	76.1	245.33	362	9.39	15.91	23.3	17	80.6	237	9973.8	15828	22576.3
26	81.11	257.18	396.3	50.14	117.42	189.7	158	341.1	772	7532.4	22474.2	37705
27	-15.05	21.87	67.27	0.016	2.14	4.11	0	4.73	16	0	343.99	812.56
28	-0.41	15.43	40.92	47.16	257.91	818.2	20	68.87	130	553.69	1105.62	2008.1
29	-2.4	10.8	29.63	0.015	1.81	5.52	0	5.53	12	0	92.15	208.19
30	63.38	91.4	128.5	2.23	4.08	5.89	6	12.67	28	1268	2264.08	3350.87
31	206.12	444.88	800	25.15	41.03	74.44	46	123.3	246	14606	24419.3	42333.3
32	-2.4	10.8	29.63	0	2.96	8.14	0	5.53	12	0	92.15	208.19
33	64.69	145.67	218.4	31.56	75.87	128.9	41	161.5	295	3225.8	6789.81	12862.5

To be able to determine the significance of the factors and their interactions, statistical analyses are carried out for each performance measure.

Cmax Difference Analysis

Table 15. *Cmax Difference* Regression Results for *PR* rule

Regression Statistics	
Multiple R	0.94673398
R Square	0.89630524
Adjusted R Square	0.81565375
Standard Error	75.5293918
Observations	33

The regression statistics reported in Table 15 indicate a Multiple R = 0.947; this is a very good value, indicating the success of the regression in predicting the values of the dependent variable *Cmax Difference* within the sample. However, the smaller R Square (0.896) indicates that not all the factors have significant effects.

Table 16. *Cmax Difference* ANOVA Test for *PR* rule

ANOVA

	df	*SS*	*MS*	*F*	*Significance F (p-value)*
Regression	14	887572	63398	11.11331372	3.86E-06
Residual	18	102684.4	5704.689		
Total	32	990256.4			

Based on the p-value listed for the whole model (Table 16), one can conclude the model is significant since the p-value is very small. This means that at least some of the factors used in the experiment, and/or their interactions have significant influence on *Cmax Difference*. To determine which factors and interactions are the most significant, further analysis is needed. Table 17 summarizes the effect test for all factors and their interactions.

Table 17. *Cmax Difference* Effect Test for *PR* rule

	Coefficients	*Standard Error*	*t Stat*	*P-value*
A	61.1911609	17.45448	3.505757	***0.00252425***
B	78.7776971	16.81677	4.684473	***0.000184627***
C	-146.602623	17.45448	-8.39914	***1.2167E-07***
D	13.4694165	17.26671	0.78008	0.445481078
AB	29.6004878	20.96409	1.411961	0.175015829
AC	-63.8021371	20.8407	-3.06142	***0.006722824***
AD	-12.1847602	20.70825	-0.5884	0.563574026
BC	-108.049782	20.96409	-5.15404	***6.65995E-05***
BD	3.5314943	20.41114	0.173018	0.864567999
CD	-5.10584124	20.70825	-0.24656	0.808038158
AA	9.04774589	28.5484	0.316927	0.754945662
BB	2.1285494	29.70274	0.071662	0.943661359
CC	125.306665	28.5484	4.389271	***0.00035376***
DD	-2.89895209	29.12558	-0.09953	0.921815345

At significance level of 5% (i.e. 95% Confidence Interval), the significant factors and/or interactions are bolded. These factors were determined to be significant due to a relatively large t-Stat and a small p-value (less than 0.05). One can conclude that choosing any value for *MIncrease* (Factor D) within the limits addressed in this experiment does not affect the *Cmax Difference.*

CPU Time Analysis

From Tables 18 and 19, and based on the R-squared and p-value listed for the whole model, one can conclude that the model is significant since the p-value is very small. This means that at least some of the factors used in the experiment, and/or their interactions have significant influence on *CPU.* Following this, the effect test for all factors and their interactions is summarized in Table 20.

Table 18. *CPU* Regression Statistics for *PR* rule

Regression Statistics	
Multiple R	0.898498
R Square	0.807298
Adjusted R Square	0.657418
Standard Error	52.99221
Observations	33

Table 19. *CPU* ANOVA Results for *PR* rule

ANOVA

	df	*SS*	*MS*	*F*	*Significance F (p value)*
Regression	14	211759.8	15125.7	5.386311	0.000588939
Residual	18	50547.13	2808.174		
Total	32	262306.9			

Table 20. Factors and Interactions Effect test for *PR* rule

	Coefficients	*Standard Error*	*t Stat*	*P-value*
A	-9.07562	12.24624	-0.74109	0.468195
B	71.19263	11.79882	6.033877	***1.05E-05***
C	20.33954	12.24624	1.66088	0.114056
D	3.56697	12.1145	0.294438	0.771792
AB	-5.41185	14.70862	-0.36794	0.717211
AC	1.379783	14.62205	0.094363	0.925863
AD	-14.5722	14.52912	-1.00296	0.329172
BC	33.16255	14.70862	2.254633	***0.036851***
BD	15.63612	14.32067	1.091857	0.289295
CD	24.62712	14.52912	1.695018	0.107302
AA	-0.90818	20.02985	-0.04534	0.964334
BB	39.31008	20.83975	1.886303	0.075495
CC	55.06732	20.02985	2.749262	***0.013193***
DD	-13.0209	20.43481	-0.63719	0.532021

At significance level of 5% (i.e. 95% Confidence Interval), the only significant factor with a small p-value in the model is *Number of Jobs* (Factor B). One can conclude that choosing any value for *MIncrease* (Factor D) within the limits addressed in this experiment does not affect the *CPU.*

Shifted Jobs Analysis

From Tables 21 and 22, and based on the R-squared and p-value listed for the whole model, one can conclude that the model is significant since the p-value is very small. This means that at least some of the factors used in the experiment, and/or their interactions have significant influence on *Shifted Jobs*.

Table 21. *Shifted Jobs* R-Square for *PR* rule

Regression Statistics	
Multiple R	0.911723
R Square	0.831239
Adjusted R Square	0.69998
Standard Error	77.29007
Observations	33

Table 22. *Shifted Jobs* ANOVA Results for *PR* rule

ANOVA

	df	*SS*	*MS*	*F*	*Significance F (p-value)*
Regression	14	529631.4	37830.81	6.332837	0.000207857
Residual	18	107527.6	5973.754		
Total	32	637159			

The effect test for all factors and their interactions is summarized in Table 23. At significance level of 5% (i.e. 95% Confidence Interval), the significant factors with a small p-value in the model are *Number of Jobs* (Factor B) and *Number of Machines* (Factor C) along with their interaction and the interaction between *Number of Machines* and *MIncrease*. We then solved for the significant factors/interactions' levels of the regression model in Excel Solver with the objective of minimizing *Shifted Jobs*; i.e. solver determined the optimal combination of level settings of the factors that minimizes *Shifted Jobs*. *MIncrease* was determined to be at level (-1); i.e. *MIncrease* = 1 job will minimize *Shifted Jobs*.

Table 23. Factors and Interactions' Effect Test for *PR* rule

	Coefficients	*Standard Error*	*t Stat*	*P-value*
A	6.939794	17.86136	0.388537	0.702176
B	94.18346	17.20879	5.472987	***3.37E-05***
C	-80.2806	17.86136	-4.49465	***0.00028***
D	-25.337	17.66922	-1.43396	0.168727
AB	1.32434	21.45279	0.061733	0.951456
AC	2.377129	21.32652	0.111464	0.912482
AD	-10.0374	21.19098	-0.47366	0.641434
BC	-92.7491	21.45279	-4.3234	***0.000409***
BD	-35.2147	20.88695	-1.68597	0.109058
CD	47.5911	21.19098	2.245819	***0.037509***
AA	-20.6108	29.21389	-0.70551	0.489521
BB	45.12646	30.39514	1.48466	0.154936
CC	72.5657	29.21389	2.483945	***0.023064***
DD	10.65194	29.80453	0.357393	0.724953

Match-up Time Analysis

From Tables 24 and 25, and based on the R-squared and p-value listed for the whole model, one can conclude that the model is significant since the p-value is very small. This means that at least some of the factors used in the experiment, and/or their interactions have significant influence on *Match-up Time*.

Table 24. *Match-up Time* Regression Results for *PR* rule

Regression Statistics	
Multiple R	0.9274572
R Square	0.8601768
Adjusted R Square	0.7514254
Standard Error	7245.4466
Observations	33

Table 25. *Match-up Time* ANOVA Results for *PR* rule

ANOVA

	df	*SS*	*MS*	*F*	*Significance F (p-value)*
Regression	14	5.81E+09	4.15E+08	7.909569547	4.58E-05
Residual	18	9.45E+08	52496496		
Total	32	6.76E+09			

Table 26 summarizes the effect test for all factors and their interactions. At significance level of 5% (i.e. 95% Confidence Interval), the significant factors/interactions with a small p-value in the model are bolded. One can conclude that choosing any value for *MIncrease* (Factor D) within the limits addressed in this experiment does not affect the *Match-up Time*.

Table 26. Factors/Interactions' Effect Test for *PR* rule

	Coefficients	*Standard Error*	*t Stat*	*P-value*
A	4341.0824	1674.388	2.592639	***0.018381543***
B	8206.4979	1613.213	5.087051	***7.69278E-05***
C	-10536.5	1674.388	-6.29275	***6.22254E-06***
D	1962.029	1656.376	1.184532	0.251608203
AB	3157.969	2011.061	1.5703	0.133757204
AC	-4152.892	1999.224	-2.07725	0.052368414
AD	-1223.242	1986.518	-0.61577	0.545753373
BC	-10657.35	2011.061	-5.29937	***4.87922E-05***
BD	1007.1609	1958.017	0.514378	0.613245508
CD	-1109.795	1986.518	-0.55866	0.583275264
AA	819.73818	2738.614	0.299326	0.768120085
BB	1877.7087	2849.349	0.658996	0.51824076
CC	8261.9759	2738.614	3.016845	***0.007408835***
DD	53.695233	2793.983	0.019218	0.984878492

PR Experimental Design Conclusion

It can be concluded from the above statistical analyses that any value of *MIncrease* within the tested range can be used in *PR* without significantly affecting the performance measures, except in the case of *Shifted Jobs* where *MIncrease* = 1 will lead to a better performance measure.

Following this, and as the larger *MIncrease* the closer *PR* gets to *CR*, *MIncrease* = 1 will be used in *PR*.

COMPLETE RESCHEDULING (*CR*)

In the complete rescheduling rule, all remaining jobs after a breakdown occurrence will be optimally rescheduled without trying to match up with the initial schedule; i.e. a new optimal schedule is generated for the remaining unprocessed jobs. *CR* was embedded in *PR* and used whenever the latter was not able to match up with the initial schedule. In the *CR* approach, the unprocessed jobs along with D_J are added to the pool ResJobs_i, then they are solved optimally using MIP [3] in order to minimize the makespan. In other words, a new schedule will be generated for the remaining jobs in order to reduce the makespan as much as possible.

MIP [3] reshuffles the jobs in order to obtain the smallest Cmax_R possible. However, as the number of shifted jobs is also a stability performance measure (*Shifted Jobs*), we will minimize it as well using MIP [4]. The latter attempts to reduce *Shifted Jobs* while maintaining the same Cmax_R reported by MIP [3]. As such, the above two MIPs guarantee an optimal Cmax_R while minimizing the number of shifted jobs whenever possible.

COMPUTATIONAL TESTS AND EXPERIMENTAL DESIGN

Following the description of the repair rules, computational tests are undertaken to prove superiority and dominance. However, there are several factors that could impact the dominance of a rule over another. Therefore, the computational tests will follow an experimental design (DoE) that will analyze 6 factors as shown in Table 27.

The studied factors are respectively *Processing Time*, *Number of Jobs*, *Number of Machines*, *Repair Duration*, *Idle Time*, and *Breakdown.* They are tested at 3 settings or levels as non-linearity is suspected and to investigate a broader combination of problem settings. We recall that the repair time follows a uniform distribution between $\beta_1 E[M_i]$ and $\beta_2 E[M_i]$, where (β_1, β_2) are set to (0.1,0.2), (0.1,0.5), and (0.1,1) respectively for levels -1, 0, and 1 in the DoE. Furthermore, the idle time is calculated using *CFJI* insertion rule (Chapter 4), and the different levels of Idle Time in Table 27 refer to the value computed by *CFJI* multiplied by 80%, 100%, or 120%. Moreover, the time between breakdowns (TBB_i) will follow an exponential distribution with mean $\theta * E[M_i]$, where θ is 1, 5, and 10 respectively for the levels -1, 0, and 1 in order to test different breakdown rates.

The factors presented in Table 27 are to be studied at three levels. Three-factor interactions and above are assumed insignificant; however, all 2nd degree interactions will be considered. Quadratic terms are to be analyzed as well. If we want to conduct full factorial DoE for six factors, we will need 3^k experiments, where 3 refers to the 3 levels that we want to analyze, and k refers to the number of main factors; this is a total of $3^6 = 729$ experiments for each of the 4 rules. As it can be observed, this is a large number of trials given that each setting will be run for 50 replicates. Thus, a D-Optimal Design will be used again to reduce the number of experiments.

Table 27. Factors analyzed in the Experimental Design of the Repair and Rescheduling rules

Factor	Abbreviation	Value	Setting	Level
		[1, 50]	Low	-1
Processing time Range	**A**	[1, 100]	Medium	0
		[1, 150]	High	1
		20	Low	-1
Number of Jobs (n)	**B**	60	Medium	0
		100	High	1
		2	Low	-1
Number of Machines (m)	**C**	5	Medium	0
		8	High	1
		(0.1, 0.2)	Low	-1
Repair Duration (β_1, β_2)	**D**	(0.1, 0.5)	Medium	0
		(0.1, 1)	High	1
		80%	Low	-1
Idle Time (*CFJI* Levels)	**E**	100%	Medium	0
		120%	High	1
		1	Low	-1
Breakdown (θ)	**F**	5	Medium	0
		10	High	1

JMP 6.0 from SAS was used to generate the D-Optimal design, and the following Design Diagnostics were reported.

Table 28. Rules' D-Optimal Design Diagnostics

D Efficiency	70.64045
G Efficiency	100
A Efficiency	39.79054
Average Variance of Prediction	2.51316

The D-Optimal Design is presented in Table 29. As it can be seen, through DoE, we were able to reduce the number of experiments from 729 to 73 experiments.

For each of the experiments in Table 29, 51 replicates were run. The total number of replicates is 51 × 73 = 3723 for each of the 4 rules (i.e. a total of 14892 replicates for this DoE). The following four performance measures are reported: *CPU*, *Cmax Difference*, *Shifted Jobs*, and *Matching Time*. Moreover, the 95% Confidence Interval (*CI*) attained from running ≈ 50 iterations of each rule was also included. This *CI* was determined using Equation 5 that was described by Law and Kelton (2000) using the *t* distribution:

$$\overline{X} \pm t_{n-1,1-\alpha/2}\sqrt{\frac{S^2}{n}} \tag{5}$$

As 51 iterations were run for each problem setting (i.e. n = 51), then the confidence intervals will be:

$$\overline{X} \pm t_{50,0.975}\sqrt{\frac{S^2}{51}} \Rightarrow \overline{X} \pm 2.009\sqrt{\frac{S^2}{51}}$$

The performance measures' averages along with the confidence intervals for the *RSR*, *FJR*, *PR*, and *CR* are presented respectively in Tables 30, 31, 32, and 33.

Table 29. D-Optimal Design for the Rules' Experiments

Run	A	B	C	D	E	F	AB	AC	AD	AE	AF	BC	BD	BE	BF	CD	CE	CF	DE	DF	EF	AA	BB	CC	DD	EE	FF
1	-1	-1	-1	1	1	0	1	1	-1	-1	0	1	-1	-1	0	-1	-1	0	1	0	0	1	1	1	1	1	0
2	0	0	-1	-1	-1	-1	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	0	0	1	1	1	1
3	1	1	0	-1	-1	0	1	0	-1	-1	0	0	-1	-1	0	0	0	0	1	0	0	1	1	0	1	1	0
4	1	-1	-1	-1	-1	1	-1	-1	-1	-1	1	1	1	1	-1	1	1	-1	1	-1	-1	1	1	1	1	1	1
5	-1	0	0	-1	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	1	0	0
6	0	-1	0	-1	-1	1	0	0	0	0	0	0	1	1	-1	0	0	0	1	-1	-1	0	1	0	1	1	1
7	0	0	1	-1	1	1	0	0	0	0	0	0	0	0	0	-1	1	1	-1	-1	1	0	0	1	1	1	1
8	-1	1	0	1	-1	-1	-1	0	-1	1	1	0	1	-1	-1	0	0	0	-1	-1	1	1	1	0	1	1	1
9	1	-1	1	1	1	1	-1	1	1	1	1	-1	-1	-1	-1	1	1	1	1	1	1	1	1	1	1	1	1
10	1	1	-1	-1	0	0	1	-1	-1	0	0	-1	-1	0	0	1	0	0	0	0	0	1	1	1	1	0	0
11	-1	1	1	0	-1	0	-1	-1	0	1	0	1	0	-1	0	0	-1	0	0	0	0	1	1	1	0	1	0
12	1	-1	0	0	0	1	-1	0	0	0	1	0	0	0	-1	0	0	0	0	0	0	1	1	0	0	0	1
13	1	0	-1	0	1	-1	0	-1	0	1	-1	0	0	0	0	0	-1	1	0	0	-1	1	0	1	0	1	1
14	1	-1	1	-1	1	0	-1	1	-1	1	0	-1	1	-1	0	-1	1	0	-1	0	0	1	1	1	1	1	0
15	-1	0	0	1	1	1	0	0	-1	-1	-1	0	0	0	0	0	0	0	1	1	1	1	0	0	1	1	1
16	1	1	0	1	0	1	1	0	1	0	1	0	1	0	1	0	0	0	0	1	0	1	1	0	1	0	1
17	1	-1	1	0	-1	-1	-1	1	0	-1	-1	-1	0	1	1	0	-1	-1	0	0	1	1	1	1	0	1	1
18	0	1	-1	1	1	1	0	0	0	0	0	-1	1	1	1	-1	-1	-1	1	1	1	0	1	1	1	1	1
19	0	0	-1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	-1	0	0	0	0	0	1	0	0	1
20	1	-1	-1	1	0	-1	-1	-1	1	0	-1	1	-1	0	1	-1	0	1	0	-1	0	1	1	1	1	0	1
21	1	1	1	1	1	0	1	1	1	1	0	1	1	1	0	1	1	0	1	0	0	1	1	1	1	1	0
22	0	0	-1	1	1	-1	0	0	0	0	0	0	0	0	0	-1	-1	1	1	-1	-1	0	0	1	1	1	1
23	-1	-1	-1	0	-1	1	1	1	0	1	-1	1	0	1	-1	0	1	-1	0	0	-1	1	1	1	0	1	1
24	1	-1	0	1	-1	-1	-1	0	1	-1	-1	0	-1	1	1	0	0	0	-1	-1	1	1	1	0	1	1	1
25	-1	-1	0	1	0	1	1	0	-1	0	-1	0	-1	0	-1	0	0	0	0	1	0	1	1	0	1	0	1
26	0	0	0	1	-1	0	0	0	0	0	0	0	0	0	0	0	0	0	-1	0	0	0	0	0	1	1	0
27	0	1	0	-1	0	-1	0	0	0	0	0	0	-1	0	-1	0	0	0	0	1	0	0	1	0	1	0	1
28	0	-1	0	0	-1	-1	0	0	0	0	0	0	0	1	1	0	0	0	0	0	1	0	1	0	0	1	1
29	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	1	0	1	0
30	1	1	0	0	1	-1	1	0	0	1	-1	0	0	1	-1	0	0	0	0	0	-1	1	1	0	0	1	1
31	1	0	0	1	0	-1	0	0	1	0	-1	0	0	0	0	0	0	0	0	-1	0	1	0	0	1	0	1
32	1	0	-1	-1	1	0	0	-1	-1	1	0	0	0	0	0	1	-1	0	-1	0	0	1	0	1	1	1	0
33	0	0	0	-1	1	-1	0	0	0	0	0	0	0	0	0	0	0	0	-1	1	-1	0	0	0	1	1	1
34	1	0	1	-1	0	1	0	1	-1	0	1	0	0	0	0	-1	0	1	0	-1	0	1	0	1	1	0	1
35	-1	0	1	1	1	-1	0	-1	-1	-1	1	0	0	0	0	1	1	-1	1	-1	-1	1	0	1	1	1	1
36	-1	-1	1	0	0	0	1	-1	0	0	0	-1	0	0	0	0	0	0	0	0	0	1	1	1	0	0	0
37	0	1	1	0	1	-1	0	0	0	0	0	1	0	1	-1	0	1	-1	0	0	-1	0	1	1	0	1	1
38	-1	0	-1	0	-1	0	0	1	0	1	0	0	0	0	0	0	1	0	0	0	0	1	0	1	0	1	0
39	1	0	0	1	1	0	0	0	1	1	0	0	0	0	0	0	0	0	1	0	0	1	0	0	1	1	0
40	-1	-1	1	-1	1	1	1	-1	1	-1	-1	-1	1	-1	-1	-1	1	1	-1	-1	1	1	1	1	1	1	1
41	0	-1	1	1	-1	-1	0	0	0	0	0	-1	-1	1	1	1	-1	-1	-1	-1	1	0	1	1	1	1	1
42	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	1	1
43	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0
44	-1	1	-1	1	1	-1	-1	1	-1	-1	1	-1	1	1	-1	-1	-1	1	1	-1	-1	1	1	1	1	1	1
45	1	-1	1	1	0	0	-1	1	1	0	0	-1	-1	0	0	1	0	0	0	0	0	1	1	1	1	0	0
46	-1	0	1	0	-1	-1	0	-1	0	1	1	0	0	0	0	0	-1	-1	0	0	1	1	0	1	0	1	1
47	0	-1	-1	-1	1	-1	0	0	0	0	0	1	1	-1	1	1	-1	1	-1	1	-1	0	1	1	1	1	1
48	-1	1	0	-1	1	0	-1	0	1	-1	0	0	-1	1	0	0	0	0	-1	0	0	1	1	0	1	1	0
49	0	1	-1	0	-1	-1	0	0	0	0	0	-1	0	-1	-1	0	1	1	0	0	1	0	1	1	0	1	1
50	0	1	1	1	0	1	0	0	0	0	0	1	1	0	1	1	0	1	0	1	0	0	1	1	1	0	1
51	-1	1	0	0	0	1	-1	0	0	0	-1	0	0	0	1	0	0	0	0	0	0	1	1	0	0	0	1
52	-1	1	-1	-1	-1	1	-1	1	1	1	-1	-1	-1	-1	1	1	1	-1	1	-1	-1	1	1	1	1	1	1
53	1	1	1	0	1	1	1	1	0	1	1	1	0	1	1	0	1	1	0	0	1	1	1	1	0	1	1
54	1	-1	-1	0	1	0	-1	-1	0	1	0	1	0	-1	0	0	-1	0	0	0	0	1	1	1	0	1	0
55	0	0	1	0	-1	1	0	0	0	0	0	0	0	0	0	0	-1	1	0	0	-1	0	0	1	0	1	1
56	0	1	1	-1	-1	0	0	0	0	0	0	1	-1	-1	0	-1	-1	0	1	0	0	0	1	1	1	1	0
57	-1	0	-1	-1	0	-1	0	1	1	0	1	0	0	0	0	1	0	1	0	1	0	1	0	1	1	0	1
58	-1	-1	0	0	1	-1	1	0	0	-1	1	0	0	-1	1	0	0	0	0	0	-1	1	1	0	0	1	1
59	1	0	1	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	0	0	0
60	-1	1	1	1	-1	1	-1	-1	-1	1	-1	1	1	-1	1	1	-1	1	-1	1	-1	1	1	1	1	1	1
61	-1	-1	0	0	-1	0	1	0	0	1	0	0	0	1	0	0	0	0	0	0	0	1	1	0	0	1	0
62	0	-1	-1	1	-1	0	0	0	0	0	0	1	-1	1	0	-1	1	0	-1	0	0	0	1	1	1	1	0
63	0	0	1	-1	0	-1	0	0	0	0	0	0	0	0	0	-1	0	-1	0	1	0	0	0	1	1	0	1
64	1	0	-1	1	-1	-1	0	-1	1	-1	-1	0	0	0	0	-1	1	1	-1	-1	1	1	0	1	1	1	1
65	-1	1	-1	1	0	0	-1	1	-1	0	0	-1	1	0	0	-1	0	0	0	0	0	1	1	1	1	0	0
66	1	1	1	0	0	-1	1	1	0	0	-1	1	0	0	-1	0	0	-1	0	0	0	1	1	1	0	0	1
67	-1	-1	0	-1	-1	-1	1	0	1	1	1	0	1	1	1	0	0	0	1	1	1	1	1	0	1	1	1
68	0	-1	0	1	1	0	0	0	0	0	0	0	-1	-1	0	0	0	0	1	0	0	0	1	0	1	1	0
69	1	1	0	-1	1	1	1	0	-1	1	1	0	-1	1	1	0	0	0	-1	-1	1	1	1	0	1	1	1
70	1	0	0	0	-1	1	0	0	0	-1	1	0	0	0	0	0	0	0	0	0	-1	1	0	0	0	1	1
71	1	1	1	-1	-1	-1	1	1	-1	-1	-1	1	-1	-1	-1	-1	-1	-1	1	1	1	1	1	1	1	1	1
72	-1	0	1	-1	-1	0	0	-1	1	1	0	0	0	0	0	-1	-1	0	1	0	0	1	0	1	1	1	0
73	-1	-1	-1	-1	0	0	1	1	1	0	0	1	1	0	0	1	0	0	0	0	0	1	1	1	1	0	0

Table 30. *Right Shift Rule* Computational Results (Average Numbers)

	RSR							
Run	Cmax	Cmax 95% *CI*	CPU (sec)	CPU 95% *CI*	Match	Match 95% CI	Shifted Jobs	S. Jobs 95% *CI*
1	17.57	[9.4-25.7]	0.02	[0.017-0.02]	139.69	[87.69-191.7]	0	0
2	642.81	[557.3-728.3]	0.22	[0.2-0.24]	19503.9	[17461.1-21546.7]	0	0
3	39.28	[24.9-53.6]	0.64	[0.4-0.8]	1018.53	[712.3-1324.7]	0	0
4	27.93	[12.7-43.2]	0.66	[0.5-0.8]	353.91	[176.8-531]	0	0
5	7.29	[4.2-10.3]	0.37	[0.2-0.5]	112.69	[67.2-158.1]	0	0
6	2.52	[0-5.04]	0.15	[0.1-0.2]	3.66	[0-7.32]	0	0
7	4.29	[1.1-7.5]	0.61	[0.2-1]	19.15	[0-45.3]	0	0
8	18.91	[3.7-34.1]	3.49	[2.5-4.5]	187.11	[0-375]	0	0
9	5.99	[0.2-11.8]	0.02	[0.01-0.03]	5.53	[0-11.55]	0	0
10	256.39	[195.8-317]	0.099	[0.07-0.1]	10550.7	[7969.7-13131.8]	0	0
11	7.97	[5.6-10.3]	1.83	[1.3-2.4]	79.4	[57.6-101.2]	0	0
12	6.49	[0-13.04]	0.01	[0-0.01]	13.56	[0-27]	0	0
13	680.8	[546.1-815.5]	0.211	[0.2-0.24]	14064.9	[11279.6-16850.2]	0	0
14	7.1	[1.2-13]	0.02	[0.01-0.02]	9.74	[0-21]	0	0
15	6.51	[3.6-9.4]	0.02	[0.01-0.03]	66.57	[37.5-95.6]	0	0
16	16.65	[8.6-24.7]	0.055	[0.04-0.07]	323.23	[171.4-475.1]	0	0
17	34.17	[19.5-48.8]	0.17	[0.1-0.2]	57.61	[0-132.3]	0	0
18	45.94	[28.1-63.8]	1.71	[1.1-2.3]	1019.03	[552.9-1485.1]	0	0
19	37.24	[25.2-49.2]	3.03	[2.5-3.5]	978.62	[716.1-1241.2]	0	0
20	259.69	[191.9-327.5]	0.08	[0.07-0.09]	2101.25	[1612.5-2590]	0	0
21	10.97	[5.9-16]	0.78	[0.2-1.3]	163.56	[88.2-238.9]	0	0
22	370.08	[272.3-467.9]	0.21	[0.17-0.25]	4739.92	[2250.2-7229.6]	0	0
23	14.97	[9.7-20.2]	0.014	[0.01-0.01]	82.91	[42.2-123.6]	0	0
24	50.07	[34.7-65.4]	1.74	[1.3-2.1]	201.18	[89.9-312.5]	0	0
25	2.59	[1.2-4]	0.58	[0.4-0.7]	6.13	[0.1-12.1]	0	0
26	18.37	[11.6-25.1]	0.83	[0.6-1.1]	232.22	[130.3-334.2]	0	0
27	106.35	[86.9-125.7]	4.56	[3.6-5.5]	2449.61	[1977.2-2922]	0	0
28	43.28	[30.8-55.7]	0.06	[0.03-0.09]	257.3	[187.1-327.5]	0	0
29	9.11	[5.6-12.6]	0.16	[0.1-0.2]	75.76	[42.9-108.6]	0	0
30	54.04	[33.3-74.7]	3.39	[2.4-4.4]	683.33	[245.3-1121.3]	0	0
31	60.69	[35.3-86]	0.49	[0.4-0.6]	632.38	[293.2-971.6]	0	0
32	121.44	[95.7-147.2]	0.08	[0.07-0.1]	3288	[2596-3979.9]	0	0
33	45.83	[24.6-67.1]	0.76	[0.6-0.9]	532.64	[304.9-760.4]	0	0
34	4.23	[0-8.5]	0.22	[0.1-0.3]	32.85	[0-69.6]	0	0
35	5.62	[1.9-9.3]	1.06	[0.8-1.3]	16.5	[0-39.4]	0	0
36	1.02	[0-2.3]	0.014	[0.01-0.02]	0.5	[0-1.5]	0	0
37	28.68	[13.9-43.4]	6.5	[4-9.1]	275.37	[120.7-430]	0	0
38	49.19	[33.4-64.9]	0.07	[0.06-0.08]	1089.29	[746.33-1432.2]	0	0
39	12.77	[6.9-18.6]	0.025	[0.02-0.03]	203.15	[86.5-319.7]	0	0
40	2.51	[0.14-4.9]	0.01	[0-0.01]	0.56	[0-5.14]	0	0
41	35.05	[18.5-51.6]	0.03	[0.03-0.04]	95.16	[42.9-147.4]	0	0
42	15.36	[8.6-22.1]	0.02	[0.016-0.02]	146.34	[93.9-198.7]	0	0
43	25	[14.4-35.6]	0.05	[0.04-0.06]	510.02	[315.5-704.5]	0	0
44	268.34	[171.6-365]	0.36	[0.3-0.4]	5586.18	[3157.1-8015.3]	0	0
45	19.76	[3.6-35.9]	0.016	[0.012-0.02]	21	[16.5-25.5]	0	0
46	13.26	[8.6-17.9]	0.06	[0.05-0.07]	101.16	[60.7-141.6]	0	0
47	266.05	[206.8-325.3]	0.1	[0.08-0.1]	2844.79	[2117.7-3571.9]	0	0
48	13.65	[8.7-18.6]	0.42	[0.2-0.7]	241.6	[170.1-313]	0	0
49	1041.83	[920.4-1163.2]	0.43	[0.4-0.5]	43751.8	[38410.8-49092.7]	0	0
50	10.62	[6.5-14.7]	1.41	[0.7-2.1]	90.52	[58.3-122.7]	0	0
51	9.4	[6.3-12.4]	0.09	[0.06-0.1]	193.66	[131.5-255.8]	0	0
52	39.36	[30.2-48.5]	0.05	[0.045-0.06]	1665.08	[1291.8-2038.4]	0	0
53	12.98	[7.9-18.1]	5.02	[4.3-5.7]	154.54	[95.79-213.3]	0	0
54	51.34	[35.6-67.1]	0.02	[0.01-0.02]	429.84	[290.2-569.4]	0	0
55	6.12	[2.8-9.4]	0.73	[0.5-1]	32.11	[14.1-50.1]	0	0
56	16.92	[11.8-22]	0.63	[0.5-0.8]	230.78	[169-292.5]	0	0
57	292.83	[262.5-323.1]	2.07	[1.9-2.2]	8964.24	[7960.6-9967.8]	0	0
58	20.16	[13-27.3]	1.25	[0.9-1.6]	99.01	[45.1-152.9]	0	0
59	10.41	[5.8-15]	1.31	[0.4-2.2]	90.11	[51.8-128.4]	0	0
60	5.03	[2.6-7.4]	0.26	[0.2-0.4]	61.77	[40.1-83.4]	0	0
61	6.72	[3.5-9.95]	0.15	[0.09-0.2]	20.78	[10.3-31.3]	0	0
62	32.37	[17.9-46.8]	0.41	[0.2-0.6]	271.76	[129.1-414.4]	0	0
63	27.39	[18.1-36.7]	3.25	[2.5-4]	324.81	[237.3-412.3]	0	0
64	949.77	[806.9-1092.7]	2.5	[1.5-4]	16425	[13041.2-19808.9]	0	0
65	53.84	[38.2-69.4]	2.36	[1.3-3.6]	1461.08	[1052-1870.1]	0	0
66	36.6	[16.6-56.6]	6.47	[5.4-7.5]	364.87	[112.9-616.8]	0	0
67	16.7	[12.7-20.7]	1.66	[1.2-2.1]	104.48	[72.1-136.9]	0	0
68	9.51	[3.2-15.8]	0.54	[0.3-0.8]	8.71	[0-23.9]	0	0
69	21.34	[13.8-28.9]	0.17	[0.1-0.2]	560.38	[384.7-736.1]	0	0
70	23.1	[12.8-33.4]	0.74	[0.5-1]	262.2	[178.4-346]	0	0
71	77	[48.8-105.2]	4.45	[3.1-5.8]	1190	[936.5-1443.5]	0	0
72	5.38	[3.4-7.3]	0.3	[0.2-0.4]	39.17	[24.5-53.8]	0	0
73	19.6	[13.4-25.8]	0.27	[0.2-0.3]	203.75	[140.6-266.9]	0	0

Table 31. *Fit Job Repair* Computational Tests (Average Numbers)

	FJR							
Run	**Cmax**	**Cmax 95% *CI***	**CPU (sec)**	**CPU 95% *CI***	**Match**	**Match 95% *CI***	**Shifted Jobs**	**S. Jobs 95% *CI***
1	9.96	[5-14.9]	0.05	[0.04-0.06]	121.31	[64.9-177.7]	0.44	[0.2-0.7]
2	311.97	[284.1-339.8]	22.8	[15.1-30.5]	9688.87	[8768.6-10609.1]	5.78	[4.8-6.7]
3	26.79	[14.9-38.7]	6.2	[4.8-7.6]	572.7	[375-770.4]	0.75	[0.2-1.3]
4	23.4	[7.8-39]	0.59	[0.37-0.8]	256.28	[140.8-371.8]	0.05	[0-0.15]
5	5.32	[2.4-8.3]	1.78	[1.2-2.4]	79.82	[51.1-108.5]	0.2	[0-0.4]
6	2.14	[0-5.05]	2.5	[1.3-3.7]	7.62	[0-15.4]	0.05	[0-0.15]
7	2.09	[0-4.4]	1.51	[0.8-2.2]	15.55	[3.4-27.7]	0.2	[0-0.4]
8	6.29	[1.5-11.1]	1.42	[1.2-1.6]	448.81	[302.3-595.3]	7.05	[5.1-8.9]
9	5.43	[1.95-8.9]	0.02	[0.01-0.021]	11.73	[5.2-18.2]	0.08	[0-0.16]
10	163.16	[138.4-187.9]	0.77	[0.7-0.8]	5092.67	[4359-5826.3]	1.52	[1.1-1.9]
11	6.64	[4.5-8.7]	0.4	[0.22-0.6]	77.99	[57.9-98.1]	0.6	[0.3-0.9]
12	6.24	[1.1-11.4]	2.14	[1.4-2.8]	22.04	[9.5-34.6]	0.1	[0-0.2]
13	289.88	[248.2-331.5]	19.24	[5.8-32.7]	9037.92	[7594.9-10480.9]	8.68	[7.7-9.7]
14	3.76	[0.5-7]	0.024	[0.02-0.03]	16.12	[5.3-26.9]	0.24	[0.1-0.4]
15	3.57	[1.9-5.1]	11.57	[8.8-14.3]	35.3	[21.3-49.3]	0.38	[0.2-0.6]
16	18.86	[11.6-26.1]	0.65	[0.3-0.9]	269.81	[186.4-353.2]	0.72	[0.4-1]
17	8.59	[5-12.2]	6.41	[5-7.8]	109.04	[85.4-132.7]	1.22	[0.9-1.6]
18	17.29	[7.7-26.8]	20.77	[16-25.5]	629.89	[347.3-912.5]	1.1	[0.8-1.3]
19	30.33	[20.8-39.9]	7.23	[5.1-9.3]	620.71	[458.7-782.7]	0.64	[0.4-0.8]
20	109.97	[84.7-135.2]	4.59	[3.8-5.4]	1863.42	[1497.9-2228.9]	3.86	[3.2-4.5]
21	7.88	[4.1-11.7]	15.79	[9.4-22.1]	130.75	[83.9-177.6]	0.77	[0.4-1.1]
22	128.09	[95-161.2]	1.57	[1.4-1.7]	4106.68	[3185.9-5027.5]	8.88	[7.4-10.3]
23	9.32	[4.5-14.1]	0.03	[0.02-0.03]	107.31	[72.5-142.1]	0.125	[0-0.2]
24	20.63	[13.3-27.9]	0.08	[0.07-0.09]	246.7	[191.4-301.9]	1.525	[1.1-1.9]
25	1.3	[0.24-2.3]	0.02	[0.01-0.02]	6.24	[2.6-9.9]	0.075	[0-0.16]
26	12.73	[7.1-18.3]	0.12	[0.1-0.15]	180.48	[99.1-261.8]	0.367	[0.12-0.6]
27	41.27	[28.9-53.6]	2.21	[1.6-2.8]	1283.69	[909.4-1658]	5.3	[4.1-6.5]
28	18.01	[11.9-24.1]	0.12	[0.1-0.14]	191.77	[137.1-246.5]	1.2	[0.8-1.6]
29	6.97	[3.4-10.5]	0.14	[0.1-0.2]	64.11	[37.4-90.8]	0.23	[0.02-0.4]
30	6.91	[0-13.3]	3.6	[1.4-5.7]	837.09	[471.1-1203.1]	4.4	[2.9-5.9]
31	3.45	[0-8.3]	0.91	[0.8-1]	807.87	[614.3-1001.5]	4.87	[3.7-6]
32	100.9	[72.9-128.9]	0.4	[0.3-0.5]	2208.62	[1631.3-2785.9]	1.2	[0.8-1.6]
33	9.85	[3.4-16.3]	18.93	[13.9-23.9]	445.47	[302.8-588.1]	3.6	[2.7-4.5]
34	5.81	[1.9-9.7]	1.01	[0.6-1.4]	49.98	[26-73.9]	0.133	[0-0.29]
35	0.16	[0-0.95]	6.85	[4.6-9.1]	114.8	[87.3-142.3]	3.45	[2.7-4.2]
36	1.016	[0.2-1.8]	0.03	[0.02-0.03]	4.31	[1.9-6.7]	0.05	[0-0.12]
37	1.21	[0-2.6]	20.97	[18.5-23.4]	256.88	[186.9-326.9]	3.4	[2.6-4.2]
38	32.43	[25.1-39.7]	5.15	[3.9-6.4]	674.53	[513.9-835.1]	1.1	[0.8-1.4]
39	8.66	[4.4-12.9]	1.48	[0.9-2.1]	214.68	[153.9-275.4]	0.94	[0.6-1.2]
40	0.85	[0.02-1.7]	0.05	[0.04-0.06]	3.8	[0-7.6]	0.04	[0-0.1]
41	7.4	[4.6-10.2]	1.83	[1-2.7]	96.48	[73.5-119.4]	1.64	[1.2-2.1]
42	9.2	[4.6-13.8]	0.82	[0.5-1.1]	61.66	[40.9-82.4]	0.2	[0.1-0.3]
43	13.43	[8.8-18.1]	4.25	[2.97-5.5]	345.15	[257.4-432.9]	1.24	[0.8-1.6]
44	86.2	[60-112.4]	33.62	[30.8-36.4]	3620.11	[2388.5-4851.7]	12.6	[9.8-15.3]
45	3.21	[0.2-6.2]	0.026	[0.02-0.03]	13.59	[4.1-23.1]	0.27	[0.05-0.5]
46	5.22	[3.6-6.8]	15.51	[10.7-20.3]	129.49	[101.5-157.4]	2.64	[2.1-3.2]
47	100.89	[83.6-118.1]	11.87	[8.8-14.9]	1528.55	[1279.6-1777.5]	1.92	[1.5-2.3]
48	9.41	[6.4-12.4]	9.22	[7.1-11.3]	180.8	[135.5-226.1]	0.77	[0.4-1.1]
49	474.03	[422.2-525.8]	28.38	[21.2-35.6]	21779.3	[18620.5-24938]	13.08	[11.8-14.4]
50	8.14	[4.8-11.4]	4.51	[2.7-6.3]	63.41	[39.5-87.3]	0.4	[0.2-0.6]
51	4.79	[2.5-7.1]	0.52	[0.3-0.7]	80.85	[45.1-116.6]	0.4	[0.2-0.6]
52	36.21	[27.9-44.5]	15.92	[11.9-19.9]	1069.03	[824.2-1313.8]	0.82	[0.4-1.2]
53	11.67	[5.95-17.4]	20.17	[13.4-26.9]	112.8	[71.2-154.4]	0.32	[0.1-0.5]
54	43.13	[28.4-57.8]	2.02	[1.6-2.4]	379.56	[270.2-488.9]	0.325	[0.1-0.5]
55	4.64	[2.2-7.1]	2.28	[1.2-3.3]	29.4	[15.9-42.9]	0.14	[0.04-0.2]
56	11.38	[7.5-15.2]	11.57	[9.5-13.6]	152.88	[117.7-188]	0.47	[0.2-0.7]
57	142.8	[124.9-160.7]	26.32	[23.1-29.5]	4424.96	[3801.1-5048.8]	6.65	[5.8-7.5]
58	5.13	[3.1-7.1]	0.18	[0.1-0.2]	99.36	[79.7-119]	1.35	[0.9-1.7]
59	11.56	[5.7-17.4]	2.98	[1.9-4.1]	109.39	[66.6-152.2]	0.37	[0.2-0.6]
60	2.1	[0-4.3]	0.31	[0.1-0.5]	31.94	[12.6-51.3]	0.25	[0-0.5]
61	3.92	[1.7-6.1]	0.65	[0.3-1]	14.43	[8.9-19.9]	0.3	[0.1-0.4]
62	27.39	[17.7-37]	3.56	[2.8-4.3]	274.67	[205.2-344.1]	0.48	[0.3-0.7]
63	14.16	[9.7-18.6]	8.74	[6.4-11]	303.5	[236.6-370.4]	2.17	[1.6-2.7]
64	323.75	[266.5-381]	10.23	[7.03-13.4]	9305.8	[7399.1-11212.5]	9.17	[7.8-10.5]
65	38.45	[27.1-49.8]	39.21	[28.1-50.3]	1162.47	[793-1531.9]	3.37	[2.7-4.1]
66	7.72	[2-13.4]	7.69	[5-10.4]	567.18	[418.9-715.4]	4.467	[3.4-5.5]
67	10.14	[7.6-12.6]	6.01	[4.2-7.8]	101.47	[83.5-119.4]	0.75	[0.5-1]
68	3.57	[0.7-6.5]	0.95	[0.7-1.2]	11.88	[4.3-19.5]	0.15	[0-0.3]
69	15.1	[7.3-22.8]	4.49	[2.3-6.7]	296.81	[198.3-395.3]	0.37	[0.1-0.6]
70	22.33	[12-32.6]	0.1	[0.08-0.12]	183.4	[116.1-250.7]	0.4	[0.1-0.6]
71	36.94	[30.5-43.3]	10.16	[7.1-13.2]	924.64	[763.2-1086]	3.93	[3.2-4.7]
72	3.18	[1.4-5]	3.08	[1.7-4.4]	26.85	[16.6-37.1]	0.14	[0.02-0.3]
73	9.15	[5.6-12.6]	0.82	[0.5-1.1]	113.69	[77-150.3]	0.18	[0.1-0.3]

Table 32. *Partial Rescheduling* Computational Tests (Average Numbers)

	PR							
Run	Cmax	Cmax 95% *CI*	CPU (sec)	CPU 95% *CI*	Match	Match 95% *CI*	Shifted Jobs	S. Jobs 95% *CI*
1	9.89	[4.7-15.05]	1.03	[0.6-1.4]	136.26	[75.74-196.8]	4.84	[2.7-6.9]
2	316.54	[288.3-344.8]	88.87	[70.7-107]	14693.1	[13159-16227.2]	162.92	[137.6-188.2]
3	12.53	[3.8-21.2]	38.26	[18.6-57.9]	793.13	[572.3-1013.9]	21.5	[14.3-28.6]
4	23.13	[7.1-39.1]	1.01	[0.5-1.5]	357.73	[173.1-542.4]	1.2	[0-2.4]
5	3.68	[2.2-5.12]	7.12	[5.15-9.1]	130.87	[105.4-156.4]	8.3	[6.2-10.4]
6	2.78	[0.9-4.6]	0.17	[0.1-0.3]	13.82	[6.5-21.1]	0.48	[0.1-0.8]
7	3.78	[1.4-6.2]	23.58	[0-48]	34.25	[19.5-48.9]	2.78	[1.4-4.2]
8	8.52	[0.5-16.6]	157.77	[55.3-260.2]	1057.91	[601.3-1514.5]	139.05	[82.1-196]
9	3.35	[0-8]	0.27	[0-0.54]	8.94	[0-18.8]	0.35	[0-0.7]
10	187.13	[155.6-218.7]	66.01	[44.3-87.7]	7543.95	[5732.7-9355.2]	123.55	[87.9-159.2]
11	2.47	[1-3.9]	42.73	[18-67.4]	83.35	[61.5-105.2]	11.04	[7.8-14.2]
12	3.42	[0-7.5]	1.11	[0.5-1.7]	17.73	[5.5-30]	0.225	[0.01-0.4]
13	388.41	[326.5-450.3]	152.25	[83.2-221.3]	17681	[14171.9-21190.1]	223.02	[174.9-271.1]
14	4.35	[0.16-8.5]	0.25	[0.1-0.4]	25.26	[6.4-44.1]	1.23	[0.1-2.3]
15	2.97	[1-5]	1.58	[0.7-2.4]	48.51	[27.1-69.9]	4.47	[2.2-6.7]
16	4.63	[0-11.2]	4.03	[1-7.1]	211.6	[114.3-308.9]	12.35	[3-21.7]
17	9.71	[5.5-13.9]	3.9	[3.1-4.76]	111.38	[86.7-136.04]	5.12	[3.85-6.4]
18	28.97	[5.9-52]	3.64	[0.6-6.6]	813.84	[153.5-1474.1]	57.1	[29.9-84.3]
19	35.82	[21.5-50.1]	13.18	[9.8-16.6]	845.02	[607.2-1082.8]	24.83	[15.1-34.5]
20	116.09	[86-146.1]	10.22	[7.8-12.6]	2430.62	[1865.9-2995.3]	21.55	[16.2-26.9]
21	4.56	[1.4-7.7]	51.23	[29.4-73.1]	140.88	[95.2-186.6]	9.52	[5-14]
22	206.26	[157.6-254.9]	49.94	[35.9-64]	8941.1	[6601.1-11281.1]	181.9	[130.2-233.6]
23	7.95	[4.1-11.8]	2.79	[1.9-3.6]	118.14	[83.86-152.4]	1.7	[1-2.3]
24	12.36	[4.9-19.7]	4.34	[3.4-5.3]	277.92	[211.7-344.1]	7.72	[5.8-9.7]
25	1.08	[0.04-2.1]	0.34	[0.1-0.5]	6.23	[2.5-9.9]	0.45	[0.1-0.7]
26	5.46	[0.9-10]	12.16	[6.3-18]	216.31	[138.7-293.9]	9.82	[6.4-13.3]
27	36.73	[27.9-45.6]	95.36	[64.7-126]	2378.51	[1881-2876]	116.77	[93.2-140.3]
28	11.5	[5.7-17.3]	5.09	[3.9-6.3]	207.53	[156.6-258.4]	5.62	[4.1-7.1]
29	2.43	[0-5]	7.05	[3.8-10.3]	64	[41.9-86.1]	5.43	[3.3-7.6]
30	22.85	[6.9-38.8]	70.67	[29.9-111.5]	2457.27	[1751.9-3162.6]	115.03	[75.7-154.4]
31	0.9	[0-2.1]	25.31	[16.7-33.9]	1135.44	[896.9-1374]	32.44	[24.8-40.1]
32	95.75	[75.6-115.9]	19.78	[15.5-24]	2681.13	[2151.9-3210.3]	29.42	[21.3-37.5]
33	9.99	[3.6-16.3]	24.15	[16-32.3]	903.33	[686.7-1119.9]	40.58	[29.7-51.5]
34	3.4	[1.4-5.3]	7.77	[3.1-12.4]	57.47	[34.8-80.1]	3.1	[1.7-4.5]
35	0	[0-0.1]	50.53	[33.3-67.7]	165.26	[115.7-214.8]	29.22	[20.8-37.6]
36	0.92	[0.1-1.7]	0.89	[0.4-1.3]	4.28	[1.8-6.7]	0.4	[0.1-0.7]
37	0	[0-1.5]	209.86	[176.5-243.2]	522.63	[387.7-657.5]	38.97	[28.4-49.5]
38	30.86	[22.7-39]	37.97	[25.5-50.4]	883.97	[628.1-1139.9]	34.8	[25.6-43.9]
39	3.1	[0-6.3]	8.89	[5.1-12.7]	258.47	[168.9-348]	9.75	[6.6-12.9]
40	0.82	[0-1.7]	0.35	[0-0.8]	3.95	[0-8.4]	0.52	[0-1.2]
41	6.24	[2.2-10.2]	9.36	[7.4-11.3]	104.77	[80.4-129.1]	8.35	[6.4-10.3]
42	3.89	[1.3-6.5]	4.57	[2.6-6.6]	85.71	[59.8-111.6]	2.52	[1.6-3.4]
43	7.85	[1.6-14.1]	14.96	[5.2-24.7]	375.46	[224.3-526.6]	25.57	[15.5-35.6]
44	196.05	[151.2-240.9]	238.13	[200.1-276.2]	13928.8	[10391.6-17465.9]	393.89	[300.3-487.5]
45	3.76	[0.1-7.4]	0.99	[0.5-1.5]	16.06	[4.8-27.3]	1.25	[0.2-2.3]
46	3.97	[2.1-5.9]	58.87	[33.7-84]	185.14	[144.2-226.1]	27.35	[20.9-33.8]
47	102.55	[87-118]	21.76	[18.1-25.4]	1952.4	[1631-2273.8]	15.78	[12.4-19.1]
48	7.17	[3.9-10.4]	38.39	[0-77.4]	216.54	[159.4-273.7]	14	[9.8-18.2]
49	554.15	[496.4-611.9]	303.82	[281.2-326.4]	39900.2	[34707.8-45092.7]	315.51	[250.7-380.3]
50	3.89	[0.8-7]	109.4	[0-223.6]	87.61	[58.5-116.7]	7.27	[4.5-10]
51	2.43	[0.7-4.1]	36.77	[20.6-52.9]	113.04	[63.7-162.3]	10.13	[5.5-14.8]
52	36.11	[26.6-45.6]	170.09	[102.8-237.3]	1643.03	[1268.9-2017.2]	48.54	[33.7-63.4]
53	3.86	[0.4-7.3]	135.85	[72.3-199.4]	141.53	[96.5-186.5]	6.42	[4-8.8]
54	35.06	[23.2-46.9]	11.65	[8.5-14.8]	474.75	[357.8-591.7]	2.38	[1.4-3.4]
55	3.08	[1.1-5.1]	3.24	[1.3-5.2]	29.63	[17-42.2]	2.94	[1.4-4.5]
56	7.17	[4.2-10.1]	231.52	[137.7-325.3]	203.02	[158.2-247.8]	14.32	[10.2-18.5]
57	148.72	[131.7-165.8]	140.31	[123.2-157.4]	6960.47	[6103.5-7817.4]	196.71	[164.1-229.3]
58	4.38	[2.3-6.5]	17.07	[12.9-21.2]	113.28	[92.4-134.2]	8.27	[6.6-9.9]
59	6.13	[1.4-10.9]	25.66	[12.1-39.2]	126.3	[81.1-171.4]	6.28	[3.8-8.8]
60	1.51	[0.6-2.4]	139.42	[100.1-178.7]	56.69	[38.6-74.7]	8.29	[5.5-11]
61	2.98	[0.8-5.1]	3.24	[1.9-4.6]	13.91	[8.2-19.6]	1	[0.5-1.5]
62	26.4	[17.2-35.6]	20.23	[15.4-25]	325.21	[242.5-407.9]	3.44	[2.3-4.5]
63	10.49	[6.5-14.4]	70.66	[48.7-92.6]	394.71	[320.1-469.3]	26.23	[20.1-32.4]
64	459.71	[376.2-543.3]	122.52	[71.5-173.5]	17884.6	[13503.9-22265.3]	206.4	[160.8-251.9]
65	47.08	[31.6-62.5]	15.81	[10.1-21.5]	1576.42	[1027.2-2125.6]	147.2	[109.9-184.5]
66	1	[0-2.1]	226.92	[106.1-347.7]	1085.26	[763.8-1406.7]	67.08	[43.9-90.2]
67	8.75	[6.4-11.1]	2.65	[2.1-3.2]	110.45	[85.1-135.8]	6.56	[4.5-8.6]
68	4.96	[1.3-8.6]	0.44	[0.12-0.8]	10.31	[3.7-17]	0.7	[0.2-1.2]
69	4.84	[1.9-7.8]	14.36	[7.4-21.3]	503.94	[340.5-667.4]	9.13	[5.7-12.5]
70	11.76	[6.8-16.7]	4.31	[2.7-5.9]	197.96	[135.3-260.6]	4.05	[2.5-5.6]
71	18.77	[12-25.5]	269.14	[134.3-404]	1167.61	[954.9-1380.3]	48.24	[37.6-58.9]
72	2.74	[1.1-4.3]	4.69	[1.6-7.7]	31.1	[19.8-42.4]	4.3	[2.6-6]
73	9.03	[6-12]	1.63	[1.1-2.2]	147.95	[98-197.9]	2.12	[1.1-3.1]

Table 33. *Complete Rescheduling* Computational Tests (Average Numbers)

	CR							
Run	Cmax	Cmax 95% *CI*	CPU (sec)	CPU 95% *CI*	Match	Match 95% *CI*	Shifted Jobs	S. Jobs 95% *CI*
1	8.53	[4.2-12.8]	0.58	[0.4-0.7]	181.58	[112.9-250.3]	2.9	[1.8-3.9]
2	298.48	[273.8-323.1]	44.24	[37-51.5]	20571.4	[19079.5-22063.2]	60.88	[56-65.7]
3	19.07	[11.8-26.3]	51.66	[27.01-76.3]	1118.05	[884.3-1351.8]	15.25	[11.2-19.3]
4	23.13	[7.1-39.1]	0.38	[0.2-0.5]	357.73	[173.1-542.4]	1.2	[0-2.4]
5	3.29	[1.5-5.1]	5.62	[2.9-8.3]	134.13	[82.7-185.6]	6.5	[3.9-9]
6	2.78	[0.9-4.6]	0.26	[0.1-0.4]	13.82	[6.5-21.1]	0.48	[0.1-0.8]
7	3.41	[1.21-5.6]	24.32	[0-50.1]	34.78	[20.3-49.3]	2.9	[1.5-4.3]
8	2.32	[0-7.6]	177.8	[127.6-227.9]	2179.87	[1782.4-2577.4]	108	[90.2-125.8]
9	4.28	[1.4-7.1]	0.95	[0.5-1.3]	10.58	[4.6-16.5]	0.64	[0.3-1]
10	195.29	[161.7-228.8]	20.36	[16.4-24.3]	13939.2	[11387.9-16490.4]	20.3	[17.2-24]
11	2.98	[1.5-4.5]	256.59	[91.4-421.7]	100.51	[75.6-125.4]	13.18	[9.8-16.6]
12	3.42	[0-7.5]	0.3	[0.1-0.5]	17.73	[5.5-30]	0.225	[0.01-0.4]
13	255.64	[211.9-299.3]	117.74	[88.3-147.2]	33103.1	[29683.1-36523]	71.25	[63.4-79.1]
14	2.64	[0-6]	0.71	[0.2-1.2]	15.94	[0-32.5]	1.08	[0-2.4]
15	2.98	[0.4-5.5]	3.07	[1.1-5.1]	59.92	[30-89.8]	4.12	[2.1-6.2]
16	6.4	[0-14.8]	18.01	[5.8-30.2]	460.19	[244.1-676.3]	8.25	[4.1-12.4]
17	9.71	[5.5-13.9]	9.85	[6.9-12.8]	111.38	[86.7-136]	5.12	[3.8-6.4]
18	14.78	[4.9-24.7]	4.32	[2.9-5.7]	3360.66	[2309.1-4412.2]	8.96	[6.3-11.6]
19	27.49	[16-39]	1.1	[0.8-1.4]	1311.79	[942-1681.6]	4.48	[3.1-5.9]
20	90.55	[67.2-113.8]	9.63	[7.8-11.4]	3734.3	[3175.9-4292.6]	16.36	[13.2-19.5]
21	3.75	[0.2-7.3]	184.1	[94.1-274.1]	277.8	[199.5-356.1]	16.3	[11.7-20.9]
22	125.22	[95-155.5]	27.63	[24.9-30.3]	21629.7	[18913-24346.4]	78.98	[69.7-88.2]
23	7.39	[4.2-10.6]	0.68	[0.5-0.8]	116.05	[84.4-147.7]	1.46	[0.9-2]
24	12.47	[5.3-19.6]	7	[5.5-8.5]	288.57	[228-349.1]	7.66	[6.1-9.2]
25	1.35	[0.1-2.6]	0.55	[0.3-0.8]	5.86	[2.6-9.1]	0.44	[0.2-0.7]
26	7.55	[3.3-11.8]	11.82	[8.1-15.6]	305.31	[228.5-382.1]	10.34	[8-12.7]
27	38.23	[30.9-45.5]	147.55	[119.4-175.7]	4466.79	[4030.2-4903.39]	99.72	[88.7-110.8]
28	12.24	[7-17.5]	2.86	[2.2-3.5]	215.02	[167.6-262.4]	6.34	[4.8-7.9]
29	2.61	[0.4-4.8]	13.14	[8.6-17.7]	58	[40.4-75.6]	4.86	[3.1-6.6]
30	3.55	[0-7.1]	255.24	[153.1-357.4]	6037.55	[4941.4-7133.7]	99.93	[81.7-118.1]
31	3	[0-6.1]	78.22	[62.4-94]	2308.3	[1961.9-2654.8]	55.62	[48.3-62.9]
32	84.5	[62.1-106.9]	9.56	[7.7-11.4]	3990.99	[3152.9-4829.1]	8.45	[6.5-10.4]
33	10.7	[5.1-16.3]	72.4	[58.9-85.8]	1443.49	[1190.8-1696.2]	43.35	[36.3-50.4]
34	3.91	[1.3-6.5]	8.01	[3.7-12.3]	57.6	[34.9-80.2]	2.87	[1.4-4.3]
35	0	[0-0]	94.5	[66.8-122.2]	222.25	[168.1-276.3]	47.6	[37.2-58]
36	0.92	[0.1-1.7]	0.26	[0.1-0.4]	4.28	[1.8-6.7]	0.4	[0.1-0.7]
37	0	[0-1.3]	472.8	[400-545.6]	1193.98	[1021-1366.9]	78.2	[67.2-89.1]
38	32.14	[24.9-39.3]	22.9	[17.5-28.3]	1441.23	[1138.1-1744.4]	10.22	[8-12.4]
39	3.41	[0-6.8]	11.63	[7.8-15.5]	376.94	[263.2-490.6]	9.55	[6.7-12.4]
40	0.82	[0-1.7]	0.29	[0-0.6]	3.95	[0-8.4]	0.52	[0-1.2]
41	6.24	[2.2-10.2]	8.57	[6.7-10.4]	104.77	[80.4-129.1]	8.35	[6.4-10.3]
42	3.7	[1-6.4]	3.66	[2.4-4.9]	91.72	[64.1-119.3]	3.15	[2.1-4.2]
43	8.6	[1.8-15.4]	32.63	[21.7-43.5]	655.46	[450.2-860.7]	18.32	[11.6-25.1]
44	78.1	[50.3-105.9]	113.95	[98.3-129.6]	30368.2	[25741.9-34994.4]	140.4	[118.9-161.9]
45	4.29	[0.1-8.4]	1.14	[0.6-1.7]	18.36	[5.7-31]	1.43	[0.2-2.6]
46	4.99	[2.75-7.2]	81.34	[53.9-108.7]	240.66	[191.9-289.3]	37.23	[27.8-46.6]
47	96.97	[80.1-113.8]	34.49	[27.9-41.1]	2559.07	[2206.7-2911.5]	10.84	[8.9-12.7]
48	5.38	[3.1-7.7]	56.34	[18.4-94.2]	313.84	[239.9-387.7]	13.03	[9.7-16.3]
49	473.33	[414.6-532]	158.45	[121.6-195.3]	62882.9	[57233.6-68532.3]	126.17	[113.3-138.9]
50	2.96	[0.2-5.7]	279.45	[39.2-519.7]	105.81	[72.2-139.4]	9.35	[6.5-12.2]
51	1.87	[0.02-3.7]	13.12	[4.8-21.4]	123.59	[53.5-193.7]	5.9	[2.8-8.9]
52	36.71	[27.9-45.5]	12.17	[10-14.3]	2227.53	[1829.4-2625.7]	10.1	[8.4-11.8]
53	3.31	[0.04-6.6]	128.94	[67.7-190.2]	150.21	[102.7-197.7]	6.5	[4.1-8.8]
54	38.02	[26.3-49.8]	3.12	[2.5-3.7]	568.76	[436.6-700.9]	2.54	[1.9-3.1]
55	3.089	[1.1-5.1]	6.83	[3.2-10.5]	29.63	[17.05-42.2]	2.7	[1.3-4.1]
56	6.44	[4-8.9]	186.96	[128.4-245.5]	207.94	[164.7-251.1]	13.25	[10.2-16.3]
57	132.58	[115-150.1]	81.59	[71.5-91.6]	10322.7	[9333-11312.5]	56.45	[50.4-62.5]
58	3.3	[1.2-5.3]	5.75	[4.6-6.9]	119.96	[97-142.9]	9.17	[7.5-10.9]
59	7.31	[1.9-12.7]	27.6	[12.1-43]	137.71	[81.3-194.1]	7.5	[3.8-11.2]
60	0.7	[0-1.8]	313.28	[52.4-574.1]	56.58	[37.4-75.7]	9.15	[5.9-12.4]
61	3.27	[0.8-5.7]	3.03	[1.8-4.2]	14.38	[8.1-20.6]	1	[0.5-1.5]
62	19.17	[8.7-29.6]	5.8	[4.5-7.1]	375.03	[273.3-476.8]	2.75	[2-3.5]
63	10.56	[7.3-13.8]	70.1	[53.4-86.8]	433.26	[357.3-509.2]	32.2	[25.6-38.8]
64	297.94	[237.4-358.5]	176.4	[150.6-202.2]	29841.1	[26097.7-33584.4]	66.1	[58.2-74]
65	31.53	[19.1-44]	124	[98.8-149.2]	4958.36	[4184.1-5732.6]	24	[20.5-27.5]
66	7.42	[0-15.8]	512.97	[475.2-550.7]	1610.68	[1292.3-1929.1]	74.4	[61.2-87.6]
67	8.82	[6.4-11.2]	7.35	[5.7-9]	111.52	[85.9-137.2]	6.53	[4.5-8.6]
68	4.96	[1.3-8.6]	0.91	[0.3-1.5]	10.31	[3.7-16.9]	0.7	[0.2-1.2]
69	4.91	[1.5-8.3]	13.06	[5.4-20.7]	508.8	[289.9-727.7]	7.9	[4.6-11.2]
70	11.95	[6.2-17.7]	5.26	[3.1-7.4]	235.24	[157-313.5]	4.93	[3-6.8]
71	23.73	[16.6-30.9]	604.2	[539.1-669.3]	1791.8	[1505-2078.6]	77.72	[65.1-90.3]
72	3.29	[1.3-5.3]	5.42	[1.4-9.4]	33.25	[19.5-47]	4.67	[2.6-6.7]
73	9.41	[6-12.8]	0.98	[0.7-1.3]	152.11	[96.1-208.2]	2.35	[1.2-3.5]

Performance Measures' Statistical Analyses

To be able to determine the significance of the factors and their interactions, statistical analyses are carried out for each performance measure in the case of each of the four rules. Minitab 14.2 Statistical Software was used for the analyses.

Cmax Difference Statistical Analysis

In this section, the significance of the factors and their interactions is determined for each of the four rules in the case of the *Cmax Difference* performance measure.

Cmax Difference in the RSR rule

Table 34. *Cmax Difference* Regression Results for *RSR* rule

Regression Statistics	
R Square	0.793
Adjusted R Square	0.669
Standard Error	114.267
Observations	73

The regression statistics reported in Table 34 indicate a R Square = 0.793; this is an acceptable value, indicating the success of the regression in predicting the values of the dependent variable *Cmax Difference* within the sample. However, it also indicates that not all the factors have significant effects (as R Square is not very big).

Table 35. *Cmax Difference* ANOVA Test for *RSR* rule

ANOVA

	df	*SS*	*MS*	*F*	*Significance F (p-value)*
Regression	27	2254011	83482	6.39	0.000
Residual	45	587559	13057		
Total	72	2841570			

Based on the small p-value listed for the whole model (Table 35), one can conclude the significance of the model. This means that at least some of the factors used in the experiment, and/or their interactions have significant influence on *Cmax Difference.* To determine which factors and interactions are the most significant, further analysis is needed. Table 36 summarizes the effect test for all factors and their interactions. At significance level of 5% (i.e. 95% Confidence Interval), the significant factors and/or interactions are bolded. These factors were determined to be significant due to a relatively large t-Stat and a small p-value (less than 0.05). Factor C (*Number of Machines*) has a negative effect on *Cmax Difference*, i.e. when the number of machines increases, *Cmax Difference* decreases. This is logical because the jobs' load will be split over the machines, meaning that more machines will lead to smaller loads. Interaction DF (*Repair Duration* and *Breakdown*) has a positive effect on *Cmax Difference.* This makes sense because if the repair durations and breakdown rate are higher, the delays will be more frequent and longer; i.e. $Cmax_R$ will increase.

Table 36. *Cmax Difference* Effect Test for *RSR* rule

Predictor	Coefficients	SE Coef	t Stat	P-value
Constant	37.41	54.93	0.68	0.499
A	32.21	17.75	1.81	0.076
B	22.89	38.47	0.6	0.555
C	-119.61	40.91	-2.92	**0.005**
D	-24.19	41.35	-0.59	0.561
E	-3.9	39.89	-0.1	0.922
F	-57.31	42.45	-1.35	0.184
AB	0.6936	0.9494	0.73	0.469
AC	0.117	0.9763	0.12	0.905
AD	0.758	1.024	0.74	0.463
AE	-0.6894	0.9271	-0.74	0.461
AF	-0.929	1.084	-0.86	0.396
BC	-44.93	23.43	-1.92	0.062
BD	15.68	25.83	0.61	0.547
BE	-19.41	22.22	-0.87	0.387
BF	-18.49	23.65	-0.78	0.438
CD	-4.24	23.37	-0.18	0.857
CE	-21.44	22.33	-0.96	0.342
CF	-19.48	24.41	-0.8	0.429
DE	37.17	24.47	1.52	0.136
DF	121.69	26.7	4.56	**0**
EF	42.59	22.07	1.93	0.06
AA	-12.74	30.75	-0.41	0.681
BB	-24.12	30.94	-0.78	0.44
CC	80.36	30.89	2.6	**0.013**
DD	-53.62	29.98	-1.79	0.08
EE	27.5	31.62	0.87	0.389
FF	49.76	33.34	1.49	0.143

Cmax Difference in the FJR rule

The same approach implemented in analyzing *Cmax Difference* in the *RSR* rule was used here. The regression statistics are reported in Table 37, ANOVA test in Table 38, and Effect test in Table 39. The results indicate the success of the regression in predicting the values of *Cmax Difference* and that the model is significant since the p-value is very small.

Table 37. *Cmax Difference* Regression Results for *FJR* rule

Regression Statistics	
R Square	0.788
Adjusted R Square	0.66
Standard Error	49.0274
Observations	73

Table 38. *Cmax Difference* ANOVA Test for *FJR* rule

ANOVA

	df	*SS*	*MS*	*F*	*Significance F (p-value)*
Regression	27	401059	14854	6.18	0.000
Residual	45	108166	2404		
Total	72	509225			

The factors were determined to be significant due to a relatively large t-Stat and a small p-value (less than 0.05). In addition to Factor C (*Number of Machines*) and interaction DF (*Repair Duration* and *Breakdown*) that were determined to have a significant effect on *Cmax Difference* from the *RSR* rule analysis, interaction EF (*Idle Time* and *Breakdown*) had also a positive effect on *Cmax Difference.* It was anticipated that EF interaction impacts *Cmax Difference* because if the breakdown rate was high and the idle time inserted is low, then $Cmax_R$ would be much higher than $Cmax_P$. In other words, E and F are very interdependent as a larger *Idle Time* can absorb more *Breakdowns.*

Table 39. *Cmax Difference* Effect Test for *FJR* rule

Predictor	Coefficients	SE Coef	t Stat	P-value
Constant	24.61	23.57	1.04	0.302
A	14.413	7.617	1.89	0.065
B	11.43	16.5	0.69	0.492
C	-63.18	17.55	-3.6	**0.001**
D	-15.32	17.74	-0.86	0.392
E	-3.83	17.11	-0.22	0.824
F	-25.35	18.21	-1.39	0.171
AB	0.2492	0.4074	0.61	0.544
AC	0.2391	0.4189	0.57	0.571
AD	0.3036	0.4395	0.69	0.493
AE	-0.2593	0.3978	-0.65	0.518
AF	-0.2372	0.4651	-0.51	0.612
BC	-19.66	10.05	-1.96	0.057
BD	2.72	11.08	0.25	0.807
BE	-4.463	9.535	-0.47	0.642
BF	-2.94	10.15	-0.29	0.773
CD	-3.99	10.03	-0.4	0.692
CE	-12.087	9.582	-1.26	0.214
CF	-6.92	10.47	-0.66	0.512
DE	18.98	10.5	1.81	0.077
DF	48.49	11.46	4.23	**0**
EF	21.931	9.468	2.32	**0.025**
AA	-6.78	13.19	-0.51	0.61
BB	-8.33	13.28	-0.63	0.533
CC	37.66	13.25	2.84	**0.007**
DD	-24.12	12.86	-1.87	0.067
EE	7.28	13.57	0.54	0.594
FF	14.22	14.3	0.99	0.325

Cmax Difference in the PR rule

The same approach implemented earlier was used here. The results indicate the success of the regression in predicting the values of *Cmax Difference* (Table 40) and that the model is significant since the p-value is very small (Table 41).

The factors that were determined to be significant due to a relatively large t-Stat and a small p-value are bolded in Table 42; these factors are *Number of Machines* and the interaction between *Repair Duration* and *Breakdown.*

Table 40. *Cmax Difference* Regression Results for *PR* rule

Regression Statistics	
R Square	0.784
Adjusted R Square	0.654
Standard Error	61.9666
Observations	73

Table 41. *Cmax Difference* ANOVA Test for *PR* rule

ANOVA					
	df	*SS*	*MS*	*F*	*Significance F (p-value)*
Regression	27	625407	23163	6.03	0.000
Residual	45	171794	3840		
Total	72	798201			

Table 42. *Cmax Difference* Effect Test for *PR* rule

Predictor	Coefficients	SE Coef	t Stat	P-value
Constant	13	29.79	0.44	0.665
A	14.332	9.627	1.49	0.144
B	17.58	20.86	0.84	0.404
C	-69.32	22.19	-3.12	**0.003**
D	-18.2	22.42	-0.81	0.421
E	5.39	21.63	0.25	0.804
F	-27.5	23.02	-1.19	0.239
AB	0.2719	0.5149	0.53	0.6
AC	0.0814	0.5295	0.15	0.879
AD	0.5423	0.5555	0.98	0.334
AE	-0.4851	0.5027	-0.96	0.34
AF	-0.4187	0.5878	-0.71	0.48
BC	-23.98	12.71	-1.89	0.066
BD	4.59	14.01	0.33	0.745
BE	-8.73	12.05	-0.72	0.473
BF	-4.73	12.83	-0.37	0.714
CD	-3.69	12.68	-0.29	0.772
CE	-6.66	12.11	-0.55	0.585
CF	-14.31	13.24	-1.08	0.285
DE	14.73	13.27	1.11	0.273
DF	63.66	14.48	4.4	**0**
EF	21.76	11.97	1.82	0.076
AA	-1.57	16.68	-0.09	0.926
BB	-11.23	16.78	-0.67	0.507
CC	49.79	16.75	2.97	**0.005**
DD	-28.99	16.26	-1.78	0.081
EE	14.24	17.15	0.83	0.411
FF	22.29	18.08	1.23	0.224

Cmax Difference in the CR rule

The same approach implemented earlier was used here. The results indicate the success of the regression in predicting the values of *Cmax Difference* (Table 43) and that the model is significant since the p-value is very small (Table 44).

The factors that were determined to be significant due to a relatively large t-Stat and a small p-value are bolded in Table 45; these factors are *Number of Machines* and the interactions between *Repair Duration* and *Breakdown* and *Idle Time* and *Breakdown*, and their analyses were discussed earlier.

Table 43. *Cmax Difference* Regression Results for *CR* rule

Regression Statistics	
R Square	0.773
Adjusted R Square	0.637
Standard Error	49.2828
Observations	73

Table 44. *Cmax Difference* ANOVA Test for *CR* rule

ANOVA

	df	*SS*	*MS*	*F*	*Significance F (p-value)*
Regression	27	372252	13787	5.68	0.000
Residual	45	109296	2429		
Total	72	481547			

Table 45. *Cmax Difference* Effect Test for *CR* rule

Predictor	Coefficients	SE Coef	t Stat	P-value
Constant	20.59	23.69	0.87	0.389
A	12.177	7.656	1.59	0.119
B	14.06	16.59	0.85	0.401
C	-62.89	17.65	-3.56	**0.001**
D	-18.45	17.83	-1.03	0.306
E	-2.61	17.2	-0.15	0.88
F	-23.7	18.31	-1.29	0.202
AB	0.1516	0.4095	0.37	0.713
AC	0.2838	0.4211	0.67	0.504
AD	0.3625	0.4418	0.82	0.416
AE	-0.274	0.3998	-0.69	0.497
AF	-0.2433	0.4675	-0.52	0.605
BC	-18.56	10.11	-1.84	0.073
BD	1.71	11.14	0.15	0.879
BE	-4.034	9.584	-0.42	0.676
BF	-2.36	10.2	-0.23	0.818
CD	-4.85	10.08	-0.48	0.633
CE	-12.309	9.632	-1.28	0.208
CF	-7.74	10.53	-0.74	0.466
DE	19.52	10.55	1.85	0.071
DF	45.77	11.52	3.97	**0**
EF	21.767	9.518	2.29	**0.027**
AA	-7.97	13.26	-0.6	0.551
BB	-5.25	13.34	-0.39	0.696
CC	37.96	13.32	2.85	**0.007**
DD	-22.87	12.93	-1.77	0.084
EE	4.94	13.64	0.36	0.719
FF	13.81	14.38	0.96	0.342

Cmax Difference Analysis Summary

The factors and interactions' effects on *Cmax Difference* were analyzed and it is concluded that the significant factors are *Number of Machines* and the interactions between *Repair Duration* and *Breakdown* and *Idle Time* and *Breakdown*. *Number of Machines* has a negative effect on *Cmax Difference*, while the interactions between *Repair Duration* and *Breakdown* and *Idle Time* and *Breakdown* have a positive one, i.e. increases *Cmax Difference*.

CPU Statistical Analysis

In this section, the significance of the factors and their interactions is determined for each of the four rules in the case of the *CPU* performance measure. This analysis will follow the same approach used for the *Cmax Difference* Statistical Analysis.

CPU in the RSR rule

Table 46. *CPU* Regression Results for *RSR* rule

Regression Statistics	
R Square	0.656
Adjusted R Square	0.449
Standard Error	1.11545
Observations	73

The R Square reported in the regression statistics (Table 46) indicates that 65.6% of the variation in *CPU* can be predicted using the regression model.

Table 47. *CPU* ANOVA Test for *RSR* rule

ANOVA

	df	*SS*	*MS*	*F*	*Significance F (p-value)*
Regression	27	106.597	3.948	3.17	0.000
Residual	45	55.99	1.244		
Total	72	162.587			

Based on the p-value listed for the whole model (Table 47), one can conclude the model is significant since the p-value is very small. This means that at least some of the factors used in the experiment, and/or their interactions have significant influence on *CPU*.

To determine which factors and interactions are the most significant, further analysis is needed. Table 48 summarizes the effect test for all factors and their interactions. At significance level of 5% (i.e. 95% Confidence Interval), the significant factors and/or interactions are bolded.

Table 48. *CPU* Effect Test for *RSR* rule

Predictor	Coefficients	SE Coef	t Stat	P-value
Constant	0.3795	0.5212	0.73	0.47
A	0.1044	0.1687	0.62	0.539
B	0.706	0.1697	4.16	**0**
C	0.2217	0.1731	1.28	0.207
D	0.0545	0.1713	0.32	0.752
E	0.0921	0.1668	0.55	0.584
F	-0.5646	0.1741	-3.24	**0.002**
AB	0.1047	0.2101	0.5	0.621
AC	0.2	0.2174	0.92	0.362
AD	0.1115	0.2286	0.49	0.628
AE	-0.015	0.2165	-0.07	0.945
AF	0.1398	0.2217	0.63	0.532
BC	0.6398	0.2235	2.86	**0.006**
BD	-0.0031	0.2198	-0.01	0.989
BE	0.0652	0.2198	0.3	0.768
BF	-0.4901	0.2262	-2.17	**0.036**
CD	-0.3574	0.2282	-1.57	0.124
CE	0.2746	0.2255	1.22	0.23
CF	-0.569	0.2351	-2.42	**0.02**
DE	-0.2904	0.2195	-1.32	0.192
DF	0.2921	0.2406	1.21	0.231
EF	0.0952	0.2203	0.43	0.668
AA	-0.0035	0.2999	-0.01	0.991
BB	0.3088	0.2885	1.07	0.29
CC	0.282	0.2912	0.97	0.338
DD	-0.1554	0.2907	-0.53	0.596
EE	-0.3519	0.3118	-1.13	0.265
FF	0.8267	0.295	2.8	**0.007**

Factor B (*Number of Jobs*) has a positive effect on *CPU*, i.e. when the number of jobs increases, *CPU* increases too. This is logical because more jobs will need to be shifted when a disruption occurs.

Factor F (*Breakdown*) has a negative effect on *CPU* because the larger the time between breakdowns the less they will occur and less *CPU* will be required.

Interaction BC (*Number of Jobs* and *Number of Machines*) has a positive effect on *CPU* because when the number of machines and jobs increases, the problem size becomes larger and more *CPU* is needed.

Interactions BF (*Number of Jobs* and *Breakdown*) and CF (*Number of Machines* and *Breakdown*) have a negative effect on *CPU.* This is because of their interaction with *Breakdown*, as the latter has a negative effect on *CPU.*

CPU in the FJR rule

The *FJR* regression statistics are reported in Table 49, ANOVA test in Table 50, and Effect test in Table 51. The results indicate the success of the regression in predicting the values of *CPU* and that the model is significant since the p-value is very small.

Table 49. *CPU* Regression Results for *FJR* rule

Regression Statistics	
R Square	0.688
Adjusted R Square	0.501
Standard Error	5.57668
Observations	73

Table 50. *CPU* ANOVA Test for *FJR* rule

ANOVA

	df	*SS*	*MS*	*F*	*Significance F (p-value)*
Regression	27	3089.1	114.41	3.68	0.000
Residual	45	1399.47	31.1		
Total	72	4488.57			

Table 51. *CPU* Effect Test for *FJR* rule

Predictor	Coefficients	SE Coef	t Stat	P-value
Constant	-1.215	2.606	-0.47	0.643
A	-1.012	0.8432	-1.2	0.236
B	4.2585	0.8485	5.02	**0**
C	-2.3371	0.8654	-2.7	**0.01**
D	-0.9629	0.8566	-1.12	0.267
E	1.022	0.8341	1.23	0.227
F	-2.8425	0.8706	-3.26	**0.002**
AB	-1.085	1.05	-1.03	0.307
AC	1.332	1.087	1.23	0.227
AD	1.08	1.143	0.95	0.35
AE	-1.103	1.083	-1.02	0.314
AF	1.003	1.109	0.9	0.37
BC	-1.157	1.117	-1.04	0.306
BD	0.042	1.099	0.04	0.97
BE	2.275	1.099	2.07	**0.044**
BF	-0.498	1.131	-0.44	0.662
CD	-0.142	1.141	-0.12	0.902
CE	0.585	1.127	0.52	0.606
CF	1.517	1.176	1.29	0.203
DE	1.633	1.097	1.49	0.144
DF	2.655	1.203	2.21	**0.032**
EF	0.571	1.101	0.52	0.607
AA	-0.116	1.499	-0.08	0.939
BB	-0.002	1.442	0	0.999
CC	5.238	1.456	3.6	**0.001**
DD	-1.019	1.454	-0.7	0.487
EE	3.33	1.559	2.14	**0.038**
FF	3.377	1.475	2.29	**0.027**

The factors were determined to be significant due to a relatively large t-Stat and a small p-value (less than 0.05). Factors B and F were explained earlier.

Factor C (Number of Machines) has a negative effect on CPU in the case of FJR because the larger the number of machines, the easier for FJR to fit a job as there are more options.

BE positive effect can be attributed to the interaction of factor E with B, as the latter has a strong positive effect on CPU.

DF interaction in FJR has a positive effect; this is logical because for example if the repair duration and breakdown rate are both high, FJR will require more time to fit the down jobs.

CPU in the PR rule

The *PR* regression statistics are reported in Table 52, ANOVA test in Table 53, and Effect test in Table 54. The results indicate the success of the regression in predicting the values of *CPU* and that the model is significant since the p-value is very small.
The factors that were determined to be significant due to a relatively large t-Stat and a small p-value are bolded in Table 54; these factors are *Number of Jobs*, *Breakdown*, and the interaction between *Number of Jobs* and *Breakdown*.

Table 52. *CPU* Regression Results for *PR* rule

Regression Statistics	
R Square	0.783
Adjusted R Square	0.653
Standard Error	43.54
Observations	73

Table 53. *CPU* ANOVA Test for *PR* rule

ANOVA

	df	*SS*	*MS*	*F*	*Significance F (p-value)*
Regression	27	307954	11406	6.02	0.000
Residual	45	85308	1896		
Total	72	393262			

Table 54. *CPU* Effect Test for *PR* rule

Predictor	Coefficients	SE Coef	t Stat	P-value
Constant	-23.66	20.35	-1.16	0.251
A	-3.006	6.584	-0.46	0.65
B	53.166	6.624	8.03	**0**
C	-2.055	6.756	-0.3	0.762
D	-2.762	6.688	-0.41	0.682
E	-10.877	6.513	-1.67	0.102
F	-30.948	6.797	-4.55	**0**
AB	0.477	8.202	0.06	0.954
AC	12.294	8.485	1.45	0.154
AD	1.588	8.921	0.18	0.86
AE	-1.455	8.452	-0.17	0.864
AF	-3.278	8.655	-0.38	0.707
BC	9.934	8.723	1.14	0.261
BD	-8.847	8.58	-1.03	0.308
BE	-15.928	8.581	-1.86	0.07
BF	-25.264	8.828	-2.86	**0.006**
CD	2.852	8.908	0.32	0.75
CE	2.28	8.802	0.26	0.797
CF	13.714	9.178	1.49	0.142
DE	3.555	8.567	0.41	0.68
DF	-2.11	9.392	-0.22	0.823
EF	0.9	8.599	0.1	0.917
AA	-1.17	11.71	-0.1	0.921
BB	25.13	11.26	2.23	**0.031**
CC	39.09	11.37	3.44	**0.001**
DD	-8.71	11.35	-0.77	0.447
EE	15.89	12.17	1.31	0.198
FF	39.07	11.52	3.39	**0.001**

CPU in the CR rule

The *CR* regression statistics are reported in Table 55, ANOVA test in Table 56, and Effect test in Table 57. The results indicate the success of the regression in predicting the values of *CPU* and that the model is significant since the p-value is very small.

The factors that were determined to be significant due to a relatively large t-Stat and a small p-value are bolded in Table 57; these factors are *Number of Jobs*, *Number of Machines,* *Breakdown*, and the interactions between *Processing Time* and *Breakdown*, *Number of Jobs* and *Number of Machines*, and *Number of Jobs* and *Breakdown*.

Table 55. *CPU* Regression Results for *CR* rule

Regression Statistics	
R Square	0.864
Adjusted R Square	0.783
Standard Error	56.911
Observations	73

Table 56. *CPU* ANOVA Test for *CR* rule

ANOVA

	df	*SS*	*MS*	*F*	*Significance F (p-value)*
Regression	27	926954	34332	10.60	0.000
Residual	45	145749	3239		
Total	72	1072703			

AF has a negative effect on *CPU* which is attributed to the interaction between factors A and F, as the latter has a strong negative effect on *CPU.*

Note that factor C has a positive effect on *CPU* in the case of *CR* (versus a negative one in the other rules); this is because *CR* uses a MIP to obtain optimal new schedules every time a disruption occurs, leading to a higher *CPU* especially as the problem size increases, i.e. when B and C increase.

Table 57. *CPU* Effect Test for *CR* rule

Predictor	Coefficients	SE Coef	t Stat	P-value
Constant	-37.09	26.59	-1.39	0.17
A	9.342	8.605	1.09	0.283
B	76.294	8.659	8.81	**0**
C	39.618	8.831	4.49	**0**
D	11.854	8.742	1.36	0.182
E	-6.984	8.512	-0.82	0.416
F	-47.834	8.885	-5.38	**0**
AB	6.31	10.72	0.59	0.559
AC	8.85	11.09	0.8	0.429
AD	-2.65	11.66	-0.23	0.821
AE	-6.85	11.05	-0.62	0.538
AF	-29.5	11.31	-2.61	**0.012**
BC	61.9	11.4	5.43	**0**
BD	12.37	11.22	1.1	0.276
BE	-13.4	11.22	-1.19	0.238
BF	-50.94	11.54	-4.41	**0**
CD	2.99	11.64	0.26	0.799
CE	-2.79	11.51	-0.24	0.81
CF	-10.8	12	-0.9	0.373
DE	-11.63	11.2	-1.04	0.304
DF	19.98	12.28	1.63	0.111
EF	0.31	11.24	0.03	0.978
AA	16.25	15.3	1.06	0.294
BB	42.44	14.72	2.88	**0.006**
CC	41.43	14.86	2.79	**0.008**
DD	-0.78	14.83	-0.05	0.958
EE	13.9	15.91	0.87	0.387
FF	43.4	15.05	2.88	**0.006**

CPU Analysis Summary

The factors and interactions' effects on *CPU* were analyzed and it is concluded that the significant factors and interactions differ among the rules. For example, factor C was significant in rules *FJR* and *CR* but insignificant in *RSR* and *PR*.

Match-up Statistical Analysis

In this section, the significance of the factors and their interactions is determined for each of the four rules in the case of the *Match-up Time* performance measure. This analysis will follow the same approach used earlier.

Match-up in the RSR rule

The *RSR* regression statistics are reported in Table 58, ANOVA test in Table 59, and Effect test in Table 60. The results indicate the success of the regression in predicting the values of *Match-up* and that the model is significant since the p-value is very small.
The factors that were determined to be significant due to a relatively large t-Stat and a small p-value are bolded in Table 60; these factors are *Number of Jobs*, *Number of Machines, Idle Time,* and *Breakdown*, and the interactions between *Number of Machines* and *Idle Time*, and *Number of Machines* and *Breakdown.*

Table 58. *Match-up* Regression Results for *RSR* rule

Regression Statistics	
R Square	0.74
Adjusted R Square	0.584
Standard Error	3974.41
Observations	73

Table 59. *Match-up* ANOVA Test for *RSR* rule

ANOVA

	df	*SS*	*MS*	*F*	*Significance F (p-value)*
Regression	27	2022849784	74920362	4.74	0.000
Residual	45	710817562	15795946		
Total	72	2733667346			

Table 60. *Match-up* Effect Test for *RSR* rule

Predictor	Coefficients	SE Coef	t Stat	P-value
Constant	1542	1857	0.83	0.411
A	842.4	601	1.4	0.168
B	1847.5	604.7	3.06	**0.004**
C	-3165.4	616.7	-5.13	**0**
D	-176	610.5	-0.29	0.774
E	-1444.1	594.5	-2.43	**0.019**
F	-2115.6	620.5	-3.41	**0.001**
AB	807.8	748.7	1.08	0.286
AC	-1335.3	774.6	-1.72	0.092
AD	-102	814.3	-0.13	0.901
AE	-669.6	771.5	-0.87	0.39
AF	-397.2	790	-0.5	0.618
BC	-2627	796.2	-3.3	**0.002**
BD	-106	783.2	-0.14	0.893
BE	-933.7	783.3	-1.19	0.24
BF	-1379	805.9	-1.71	0.094
CD	1164.8	813.1	1.43	0.159
CE	1666.2	803.5	2.07	**0.044**
CF	3537.7	837.8	4.22	**0**
DE	622	782	0.8	0.431
DF	749.6	857.4	0.87	0.387
EF	1488.5	784.9	1.9	0.064
AA	-1333	1069	-1.25	0.219
BB	105	1028	0.1	0.919
CC	2793	1038	2.69	**0.01**
DD	-2348	1036	-2.27	**0.028**
EE	863	1111	0.78	0.442
FF	828	1051	0.79	0.435

Table 61 describes the factors effects on Match-up Time in the case of RSR and lists the causes of these effects.

Table 61. Factors Effects on *Match-up Time* in the case of *RSR*

Match-up Effects Diagnosis for RSR rule		
Factor/ Interaction	**Effect**	**Cause of Effect**
B	**+**	When the number of jobs increases, RSR will shift more jobs to the right, i.e. longer time to match.
C	**-**	When there are more machines, the jobs on each machine will be less, i.e. time to match will be less.
E	**-**	It is easier for RSR to match-up with the initial schedule when the idle time is larger as it will compensate the shifting of the jobs.
F	**-**	When the time between breakdowns is larger, less delay will occur, hence, it is easier to match-up with initial schedule.
BC	**-**	BC effect is negative because C (number of machines) effect is stronger than B (number of jobs). It is obvious that B and C interact as the number of jobs on each machine depends on both of them.
CE	**+**	C (number of machines) and E (Idle Time) interact because the higher the number of machines, the fewer jobs assigned to each machine, i.e. the less idle time.
CF	**+**	C (number of machines) and F (Breakdown) interact because more machines lead to fewer breakdowns on each machine as no more than one breakdown can occur at a time over the machines.

Match-up in the FJR rule

The *FJR* regression statistics are reported in Table 62, ANOVA test in Table 63, and Effect test in Table 64. The results indicate the success of the regression in predicting the values of *Match-up* and that the model is significant since the p-value is very small.

Table 62. *Match-up* Regression Results for *FJR* rule

Regression Statistics	
R Square	0.762
Adjusted R Square	0.62
Standard Error	1957.36
Observations	73

Table 63. *Match-up* ANOVA Test for *FJR* rule

ANOVA

	df	*SS*	*MS*	*F*	*Significance F (p-value)*
Regression	27	552765280	20472788	5.34	0.000
Residual	45	172406102	3831247		
Total	72	725171382			

Table 64. *Match-up* Effect Test for *FJR* rule

Predictor	Coefficients	SE Coef	t Stat	P-value
Constant	771.2	914.7	0.84	0.404
A	501.4	296	1.69	0.097
B	965.8	297.8	3.24	**0.002**
C	-1702.4	303.7	-5.6	**0**
D	-8.6	300.7	-0.03	0.977
E	-670.7	292.8	-2.29	**0.027**
F	-1190.4	305.6	-3.9	**0**
AB	409.9	368.7	1.11	0.272
AC	-706.5	381.5	-1.85	0.071
AD	-34	401.1	-0.08	0.933
AE	-252.6	380	-0.66	0.51
AF	-301.1	389.1	-0.77	0.443
BC	-1331.9	392.1	-3.4	**0.001**
BD	-41.5	385.7	-0.11	0.915
BE	-438.4	385.8	-1.14	0.262
BF	-715.4	396.9	-1.8	0.078
CD	457.8	400.4	1.14	0.259
CE	737.9	395.7	1.86	0.069
CF	1858.3	412.6	4.5	**0**
DE	342.2	385.1	0.89	0.379
DF	272.3	422.2	0.64	0.522
EF	635.2	386.5	1.64	0.107
AA	-576.3	526.3	-1.1	0.279
BB	-61.7	506.2	-0.12	0.904
CC	1476.4	511	2.89	**0.006**
DD	-1161.8	510.2	-2.28	**0.028**
EE	451.4	547.1	0.83	0.414
FF	557.6	517.7	1.08	0.287

The factors that were determined to be significant due to a relatively large t-Stat and a small p-value are bolded in Table 64; these factors are *Number of Jobs*, *Number of Machines, Idle*

Time, and *Breakdown*, and the interactions between *Number of Machines* and *Idle Time*, and *Number of Machines* and *Breakdown.* Their diagnosis is the same as in *RSR* (Table 61).

Match-up in the PR rule

The *PR* regression statistics are reported in Table 65, ANOVA test in Table 66, and Effect test in Table 67. The results indicate the success of the regression in predicting the values of *Match-up* and that the model is significant since the p-value is very small.
The factors that were determined to be significant due to a relatively large t-Stat and a small p-value are bolded in Table 67; these factors are *Number of Jobs*, *Number of Machines*, and *Breakdown*, and the interactions between *Number of Machines* and *Number of Jobs*, *Number of Jobs* and *Breakdown*, and *Number of Machines* and *Breakdown.* Their diagnosis is the same as in *RSR* (Table 61).
The negative effect of BF can be attributed to the interaction between factors B (*Number of Jobs*) and F (*Breakdown*), as the latter has a stronger negative effect on *Match-up Time*.

Table 65. *Match-up* Regression Results for *PR* rule

Regression Statistics	
R Square	0.757
Adjusted R Square	0.611
Standard Error	3690.9
Observations	73

Table 66. *Match-up* ANOVA Test for *PR* rule

ANOVA

	df	*SS*	*MS*	*F*	*Significance F (p-value)*
Regression	27	1906312712	70604175	5.18	0.000
Residual	45	613023257	13622739		
Total	72	2519335969			

Table 67. *Match-up* Effect Test for *PR* rule

Predictor	Coefficients	SE Coef	t Stat	P-value
Constant	769	1725	0.45	0.658
A	726.1	558.1	1.3	0.2
B	1943.8	561.6	3.46	**0.001**
C	-3015.1	572.7	-5.26	**0**
D	248.1	566.9	0.44	0.664
E	-1063.7	552.1	-1.93	0.06
F	-2273.3	576.2	-3.95	**0**
AB	645.6	695.3	0.93	0.358
AC	-1032.5	719.3	-1.44	0.158
AD	5.5	756.3	0.01	0.994
AE	-585.8	716.5	-0.82	0.418
AF	-388.6	733.7	-0.53	0.599
BC	-2681.6	739.4	-3.63	**0.001**
BD	63	727.4	0.09	0.931
BE	-461.7	727.4	-0.63	0.529
BF	-1706.6	748.4	-2.28	**0.027**
CD	479.5	755.1	0.64	0.529
CE	934.7	746.2	1.25	0.217
CF	3729.6	778	4.79	**0**
DE	665.4	726.3	0.92	0.364
DF	185.1	796.2	0.23	0.817
EF	796.2	728.9	1.09	0.281
AA	-851.7	992.4	-0.86	0.395
BB	147.9	954.5	0.15	0.878
CC	2683.3	963.5	2.78	**0.008**
DD	-2241.7	962	-2.33	**0.024**
EE	1156	1032	1.12	0.268
FF	1221.4	976.3	1.25	0.217

Match-up in the CR rule

The *CR* regression statistics are reported in Table 68, ANOVA test in Table 69, and Effect test in Table 70. The results indicate the success of the regression in predicting the values of *Match-up* and that the model is significant since the p-value is very small. The factors that were determined to be significant due to a relatively large t-Stat and a small p-value are bolded in Table 69; these factors are *Number of Jobs*, *Number of Machines* and *Breakdown*, and the interactions between *Number of Machines* and *Number of Jobs*, *Number of Jobs* and *Breakdown*, and *Number of Machines* and *Breakdown*. The effects can be explained in similar fashion like previous rules.

Table 68. *Match-up* Regression Results for *CR* rule

Regression Statistics	
R Square	0.784
Adjusted R Square	0.654
Standard Error	5870.32
Observations	73

Table 69. *Match-up* ANOVA Test for *CR* rule

ANOVA

	df	*SS*	*MS*	*F*	*Significance F (p-value)*
Regression	27	5616516513	208019130	6.04	0.000
Residual	45	1550728815	34460640		
Total	72	7167245328			

Table 70. *Match-up* Effect Test for *CR* rule

Predictor	Coefficients	SE Coef	t Stat	P-value
Constant	876	2743	0.32	0.751
A	1147.8	887.6	1.29	0.203
B	3493.9	893.1	3.91	**0**
C	-5136.1	910.9	-5.64	**0**
D	815.7	901.7	0.9	0.37
E	-1341.2	878.1	-1.53	0.134
F	-3896.8	916.5	-4.25	**0**
AB	1100	1106	0.99	0.325
AC	-1606	1144	-1.4	0.167
AD	-77	1203	-0.06	0.949
AE	-832	1140	-0.73	0.469
AF	-629	1167	-0.54	0.593
BC	-4982	1176	-4.24	**0**
BD	261	1157	0.23	0.823
BE	-166	1157	-0.14	0.886
BF	-3084	1190	-2.59	**0.013**
CD	186	1201	0.16	0.877
CE	749	1187	0.63	0.531
CF	6465	1237	5.22	**0**
DE	1316	1155	1.14	0.261
DF	-85	1266	-0.07	0.947
EF	636	1159	0.55	0.586
AA	-1226	1578	-0.78	0.441
BB	406	1518	0.27	0.79
CC	4640	1532	3.03	**0.004**
DD	-3520	1530	-2.3	**0.026**
EE	1928	1641	1.17	0.246
FF	2153	1553	1.39	0.172

Shifted Jobs Statistical Analysis

In this section, the significance of the factors and their interactions is determined for each of the four rules in the case of the *Shifted Jobs* performance measure. This analysis will follow the same approach used earlier.

Shifted Jobs in the RSR rule

No analysis has been done for the *Shifted Jobs* in the case of *RSR* as the latter will always have zero jobs shifted from one machine to another. Recall that *RSR* only shifts jobs to the right and is not equipped with a mechanism that allows jobs to be shifted from one machine to another.

Shifted Jobs in the FJR rule

The *FJR* regression statistics are reported in Table 71, ANOVA test in Table 72, Effect test in Table 73, and the factors effects diagnosis in Table 74. The results indicate the success of the regression in predicting the values of *Shifted Jobs* and that the model is significant since the p-value is very small.

The factors that were determined to be significant due to a relatively large t-Stat and a small p-value are bolded in Table 73 and explained in Table 74; these factors are *Number of Jobs, Number of Machines, Repair Duration,* and *Breakdown*, and the interactions between *Number of Jobs* and *Number of Machines*, *Number of Machines* and *Breakdown*, and *Number of Jobs* and *Breakdown*.

Table 71. *Shifted Jobs* Regression Results for *FJR* rule

Regression Statistics	
R Square	0.935
Adjusted R Square	0.897
Standard Error	0.935038
Observations	73

Table 72. *Shifted Jobs* ANOVA Test for *FJR* rule

ANOVA

	df	*SS*	*MS*	*F*	*Significance F (p-value)*
Regression	27	569.658	21.098	24.13	0.000
Residual	45	39.343	0.874		
Total	72	609.001			

Table 73. *Shifted Jobs* Effect Test for *FJR* rule

Predictor	Coefficients	SE Coef	t Stat	P-value
Constant	0.5975	0.4369	1.37	0.178
A	0.0081	0.1414	0.06	0.955
B	1.2207	0.1423	8.58	**0**
C	-1.1051	0.1451	-7.62	**0**
D	0.5173	0.1436	3.6	**0.001**
E	-0.1174	0.1399	-0.84	0.406
F	-2.1328	0.146	-14.61	**0**
AB	-0.0172	0.1761	-0.1	0.923
AC	0.0532	0.1822	0.29	0.772
AD	-0.0013	0.1916	-0.01	0.995
AE	-0.0429	0.1815	-0.24	0.814
AF	0.0763	0.1859	0.41	0.683
BC	-0.7411	0.1873	-3.96	**0**
BD	0.1791	0.1843	0.97	0.336
BE	-0.0562	0.1843	-0.31	0.762
BF	-1.2268	0.1896	-6.47	**0**
CD	-0.1801	0.1913	-0.94	0.351
CE	-0.0739	0.189	-0.39	0.698
CF	1.2385	0.1971	6.28	**0**
DE	0.1307	0.184	0.71	0.481
DF	-0.3752	0.2017	-1.86	0.069
EF	0.021	0.1847	0.11	0.91
AA	0.008	0.2514	0.03	0.975
BB	-0.1815	0.2418	-0.75	0.457
CC	0.7258	0.2441	2.97	**0.005**
DD	-0.2386	0.2437	-0.98	0.333
EE	0.0492	0.2613	0.19	0.852
FF	1.6665	0.2473	6.74	**0**

Table 74. Factors' Effects on *Shifted Jobs* in the case of *FJR*

Shifted Jobs Effects' Diagnostic for FJR rule		
Factor/ Interaction	**Effect**	**Cause of Effect**
B	**+**	A higher number of jobs logically indicated a higher number of shifts between the machines
C	**-**	When there are more machines, the jobs on each machine will be less, i.e. fewer jobs will be shifted.
D	**+**	Larger repair durations lead to longer delays; hence, more jobs need to be shifted in order to accommodate the delays.
F	**-**	When the time between breakdowns is larger, less delay will occur, hence, less shifting is required.
BC	**-**	BC effect is negative because C (number of machines) effect is stronger than B (number of jobs). It is obvious that B and C interact as the number of jobs on each machine depends on both of them.
BF	**-**	BF effect is negative because F (Breakdown) effect is stronger than B. B and F interact because the higher the number of jobs, the more they will be hit by a breakdown.
CF	**+**	C and F interact because more machines lead to fewer breakdowns on each machine as no more than one breakdown can occur at a time over the machines.

Shifted Jobs in the PR rule

The *PR* regression statistics are reported in Table 75, ANOVA test in Table 76, and Effect test in Table 77. The results indicate the success of the regression in predicting the values of *Shifted Jobs* and that the model is significant since the p-value is very small.

Table 75. *Shifted Jobs* Regression Results for *PR* rule

Regression Statistics	
R Square	0.929
Adjusted R Square	0.886
Standard Error	25.7356
Observations	73

Table 76. *Shifted Jobs* ANOVA Test for *PR* rule

ANOVA

	df	*SS*	*MS*	*F*	*Significance F (p-value)*
Regression	27	388875	14403	21.75	0.000
Residual	45	29804	662		
Total	72	418680			

Table 77. *Shifted Jobs* Effect Test for *PR* rule

Predictor	Coefficients	SE Coef	t Stat	P-value
Constant	7.49	12.03	0.62	0.537
A	-3.481	3.891	-0.89	0.376
B	38.775	3.916	9.9	**0**
C	-38.457	3.994	-9.63	**0**
D	7.949	3.953	2.01	**0.05**
E	-4.733	3.849	-1.23	0.225
F	-40.09	4.018	-9.98	**0**
AB	-0.677	4.848	-0.14	0.89
AC	3.082	5.015	0.61	0.542
AD	-5.504	5.273	-1.04	0.302
AE	-3.227	4.996	-0.65	0.522
AF	4.671	5.116	0.91	0.366
BC	-37.293	5.156	-7.23	**0**
BD	4.972	5.072	0.98	0.332
BE	8.828	5.072	1.74	0.089
BF	-33.152	5.218	-6.35	**0**
CD	-2.306	5.265	-0.44	0.663
CE	-5.042	5.203	-0.97	0.338
CF	42.493	5.425	7.83	**0**
DE	6.647	5.064	1.31	0.196
DF	-2.442	5.552	-0.44	0.662
EF	-1.013	5.082	-0.2	0.843
AA	3.537	6.92	0.51	0.612
BB	2.199	6.655	0.33	0.743
CC	30.035	6.718	4.47	**0**
DD	-8.616	6.708	-1.28	0.206
EE	2.493	7.193	0.35	0.731
FF	23.766	6.807	3.49	**0.001**

The factors that were determined to be significant due to a relatively large t-Stat and a small p-value are bolded in Table 77; these factors are *Number of Jobs*, *Number of Machines, Repair Duration,* and *Breakdown*, and the interactions between *Number of Jobs* and *Number* *of Machines*, *Number of Machines* and *Breakdown*, and *Number of Jobs* and *Breakdown*. Their diagnosis is the same as in *FJR* (Table 74).

Shifted Jobs in the CR rule

The *CR* regression statistics are reported in Table 78, ANOVA test in Table 79, and Effect test in Table 80. The results indicate the success of the regression in predicting the values of *Shifted Jobs* and that the model is significant since the p-value is very small. The factors that were determined to be significant due to a relatively large t-Stat and a small p-value are bolded in Table 80; these factors are *Number of Jobs*, *Number of Machines, Repair Duration,* and *Breakdown*, and the interactions between *Number of Jobs* and *Number of Machines*, *Number of Machines* and *Breakdown*, and *Number of Jobs* and *Breakdown*. Their diagnosis is the same as in *FJR* (Table 74).

Table 78. *Shifted Jobs* Regression Results for *CR* rule

Regression Statistics	
R Square	0.955
Adjusted R Square	0.928
Standard Error	8.93576
Observations	73

Table 79. *Shifted Jobs* ANOVA Test for *CR* rule

ANOVA

	df	*SS*	*MS*	*F*	*Significance F (p-value)*
Regression	27	76742.8	2842.3	35.6	0.000
Residual	45	3593.1	79.8		
Total	72	80335.9			

Table 80. *Shifted Jobs* Effect Test for *CR* rule

Predictor	Coefficients	SE Coef	t Stat	P-value
Constant	7.608	4.176	1.82	0.075
A	-0.61	1.351	-0.45	0.654
B	19.127	1.36	14.07	**0**
C	-5.858	1.387	-4.22	**0**
D	4.008	1.373	2.92	**0.005**
E	0.303	1.337	0.23	0.821
F	-25.266	1.395	-18.11	**0**
AB	0.564	1.683	0.34	0.739
AC	1.875	1.741	1.08	0.287
AD	-1.511	1.831	-0.83	0.414
AE	2.026	1.735	1.17	0.249
AF	0.818	1.776	0.46	0.647
BC	-4.382	1.79	-2.45	**0.018**
BD	1.353	1.761	0.77	0.446
BE	-0.413	1.761	-0.23	0.816
BF	-21.516	1.812	-11.88	**0**
CD	-0.692	1.828	-0.38	0.707
CE	-1.01	1.807	-0.56	0.579
CF	7.505	1.884	3.98	**0**
DE	0.631	1.758	0.36	0.721
DF	-2.989	1.928	-1.55	0.128
EF	-1.022	1.765	-0.58	0.565
AA	-1.064	2.403	-0.44	0.66
BB	2.169	2.311	0.94	0.353
CC	1.716	2.333	0.74	0.466
DD	-0.898	2.329	-0.39	0.702
EE	0.777	2.497	0.31	0.757
FF	19.931	2.364	8.43	**0**

Repair and Rescheduling Rules Comparisons

Following the analysis of factor and interaction significance, this section will compare the rules based on each performance measure as well as the overall performance. Conclusions are drawn regarding dominance among the rules.

Eigenvalue Normalization Procedure

As our objective is to determine the best rule for both schedule quality (*Cmax Difference* and *CPU*) and stability (*Shifted Jobs* and *Matching Time*), we need to compute the overall performance for each rule. However, since the performance measures are not expressed in commensurate terms, a unique measure is desired. The eigenvalue normalization procedure explained by Akturk and Gorgulu (1999) will be used to have a common unit of measure for each objective (Equation 7).

$$N_{ij} = \frac{A_{ij}}{\sqrt{\sum_{j=1}^{\rho} A_{ij}^2}} \tag{7}$$

where A_{ij} is the value of the i^{th} performance measure in the j^{th} rule, ρ is the number of rules, and N_{ij} is the normalized value of the A_{ij} value. N_{ij} is between 0 and 1, where 0 indicates the best value and 1 the worst among the rules (because for all measures in our case, the lower their values, the better the performance is).

Before judging the rules by their overall performance, they will be compared for each of the objectives to determine superiority.

Cmax Difference Comparison

Following the normalization of the performance measures, the *Cmax Difference* performance of the four rules is presented in Table 81. The boxplot of the rules is also shown in Figure 20. It is obvious that *RSR* performed the worst in the case of *Cmax Difference*; this was expected as *RSR* only shifts the jobs to the right which will eventually increase Cmax. From Figure 20, it is apparent that *CR* performed the best, followed by *PR*, then *FJR*; however, this can not be validated unless tests are undertaken to determine that the differences are statistically significant.

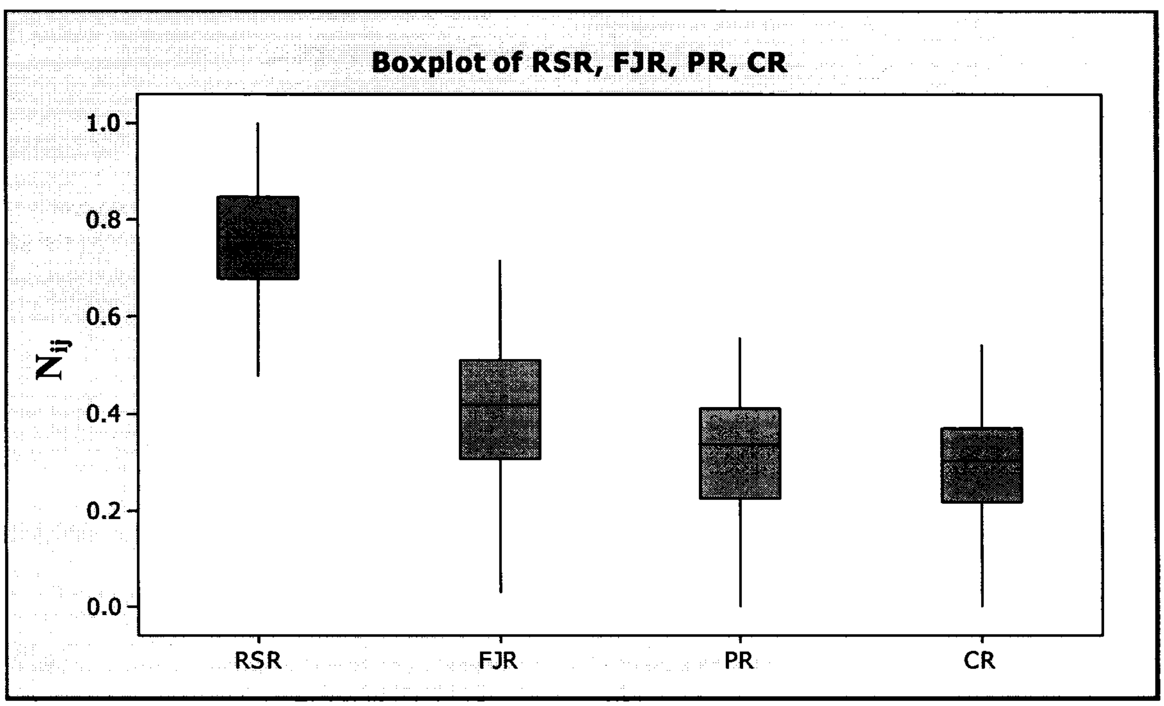

Figure 20. *Cmax Difference* Boxplot

Table 81. *Cmax Difference* Performance among the rules

	Cmax Difference			
Run	RSR	FJR	PR	CR
1	0.730513	0.41411	0.411199	0.354654
2	0.768405	0.372924	0.378387	0.356798
3	0.744815	0.507984	0.23759	0.361599
4	0.570399	0.477886	0.472372	0.472372
5	0.708693	0.51718	0.357749	0.319835
6	0.490577	0.416601	0.541192	0.541192
7	0.614813	0.299524	0.541724	0.488698
8	0.867536	0.288567	0.390873	0.106435
9	0.614866	0.557382	0.343873	0.439336
10	0.63021	0.401049	0.459968	0.480025
11	0.719826	0.599705	0.223083	0.269145
12	0.635023	0.610561	0.334635	0.334635
13	0.77902	0.331701	0.444447	0.292522
14	0.746584	0.395374	0.457414	0.277603
15	0.762846	0.418335	0.348027	0.349198
16	0.631426	0.715236	0.175586	0.24271
17	0.903619	0.227161	0.256779	0.256779
18	0.780199	0.293636	0.491998	0.251009
19	0.564924	0.4601	0.543383	0.417018
20	0.816294	0.345673	0.36491	0.28463
21	0.744193	0.53457	0.309345	0.254396
22	0.804542	0.278463	0.448402	0.272224
23	0.722946	0.45009	0.383929	0.356885
24	0.879522	0.362383	0.217114	0.219046
25	0.76753	0.385246	0.320051	0.400064
26	0.758648	0.525726	0.225488	0.311801
27	0.845429	0.328076	0.291985	0.303909
28	0.869156	0.36168	0.230945	0.245806
29	0.758403	0.580249	0.202296	0.217281
30	0.913078	0.116754	0.386081	0.059982
31	0.997066	0.056679	0.014786	0.049286
32	0.598025	0.496877	0.471516	0.416116
33	0.933229	0.200574	0.203425	0.217882
34	0.477434	0.655767	0.383753	0.441316
35	0.999595	0.028458	0	0
36	0.525644	0.523582	0.47411	0.47411
37	0.999111	0.042152	0	0
38	0.665905	0.439018	0.417765	0.435093
39	0.79302	0.537788	0.192511	0.211762
40	0.867718	0.293849	0.283477	0.283477
41	0.950029	0.200577	0.169135	0.169135
42	0.821741	0.492189	0.20811	0.197945
43	0.814999	0.437818	0.25591	0.28036
44	0.76212	0.244819	0.556807	0.221814
45	0.949272	0.154209	0.180631	0.206092
46	0.849342	0.334357	0.25429	0.319624
47	0.837642	0.317646	0.322872	0.305304
48	0.724238	0.499274	0.380424	0.285451
49	0.767789	0.349342	0.408387	0.348826
50	0.745492	0.571404	0.273066	0.207783
51	0.855586	0.435985	0.221178	0.170207
52	0.530155	0.487727	0.48638	0.494462
53	0.713958	0.641902	0.212317	0.182065
54	0.606282	0.509329	0.414029	0.448984
55	0.692899	0.525335	0.348714	0.349733
56	0.750207	0.504572	0.317907	0.28554
57	0.766801	0.373934	0.389436	0.347172
58	0.9371	0.238458	0.203596	0.153394
59	0.570452	0.63347	0.335915	0.400577
60	0.882578	0.368472	0.264949	0.122824
61	0.750859	0.438001	0.33297	0.365373
62	0.605023	0.511942	0.493438	0.358303
63	0.799978	0.41357	0.306381	0.308425
64	0.830776	0.283188	0.402114	0.260612
65	0.618083	0.441406	0.540478	0.361964
66	0.959439	0.202374	0.026214	0.194509
67	0.721287	0.437955	0.37792	0.380943
68	0.77038	0.289196	0.401796	0.401796
69	0.789321	0.558517	0.179021	0.18161
70	0.637417	0.61617	0.324503	0.329746
71	0.849857	0.407711	0.207166	0.26191
72	0.710182	0.419773	0.361691	0.434294
73	0.775989	0.36226	0.357509	0.372554

The first test is the One-Way ANOVA, which will determine if there is significant difference between the means of the rules. The ANOVA results are shown in Table 82.

Table 82. One-Way ANOVA for *Cmax Difference*

Anova: Single Factor

SUMMARY

Groups	*Count*	*Sum*	*Average*	*Variance*
RSR	73	55.35	0.758193	0.015432
FJR	73	29.55	0.404816	0.021721
PR	73	23.75	0.325307	0.016712
CR	73	21.78	0.298301	0.013518

ANOVA

Source of Variation	*SS*	*df*	*MS*	*F*	*P-value*	*F crit*
Between Groups	9.894449	3	3.29815	195.7849	3.35E-69	2.63595107
Within Groups	4.851586	288	0.016846			
Total	14.74604	291				

As the p-value in Table 82 is less than 0.05, we can reject the hypothesis that all the means are equal, i.e. there is a significant difference between the performances of the rules.

Next, a two-tailed two-sample t test was conducted for *FJR – PR* to determine if the difference between them is statistically significant (Table 83). The t-test evaluates H_0: $m_1 - m_2 = d_0$ versus H_1: $m_1 - m_2 \neq d_0$, where m_1 and m_2 are the population means and d_0 is the hypothesized difference between the two population means. As the p-value is less than 0.05, we can reject the hypothesis that the means are equal and conclude that the difference between *FJR* and *PR* is statistically significant. Moreover, as the difference is greater than zero, we conclude that *PR* performed better than *FJR* in the case of *Cmax Difference*.

Table 83. t test for *FJR – PR* in the case of *Cmax Difference*

Two-sample T for FJR vs PR

	N	Mean	StDev	SE Mean
FJR	73	0.405	0.147	0.017
PR	73	0.325	0.129	0.015

Difference = mu (FJR) - mu (PR)
Estimate for difference: 0.079509
95% CI for difference: (0.034148, 0.124869)
T-Test of difference = 0 (vs not =): T-Value = 3.47 P-Value = 0.001 DF = 141

The next t test is for *PR – CR* (Table 84). Even though the *CR* mean is smaller than *PR* mean (indicating that *CR* performed better), this difference is not statistically significant as the 95% Confidence Interval overlaps with zero. Moreover, the p-value is greater than 0.05.

Based on these tests, we conclude that for the *Cmax Difference*, the best performance was achieved by *CR* and *PR*, followed by *FJR*, then finally *RSR* that had the worst performance.

Table 84. t test for *PR – CR* in the case of *Cmax Difference*

Two-sample T for PR vs CR

	N	Mean	StDev	SE Mean
PR	73	0.325	0.129	0.015
CR	73	0.298	0.116	0.014

Difference = mu (PR) - mu (CR)
Estimate for difference: 0.027006
95% CI for difference: (-0.013221, 0.067234)
T-Test of difference = 0 (vs not =): T-Value = 1.33 P-Value = 0.187 DF = 142

CPU Comparison

The *CPU* performance of the four rules is presented in Table 85. The boxplot of the rules is also shown in Figure 21. It is visually noticeable that *RSR* performed the best, followed by *FJR*, then *PR* and *CR*. The same tests implemented in the *Cmax Difference* comparison will be used here to determine if the differences are statistically significant. The ANOVA results are shown in Table 86.

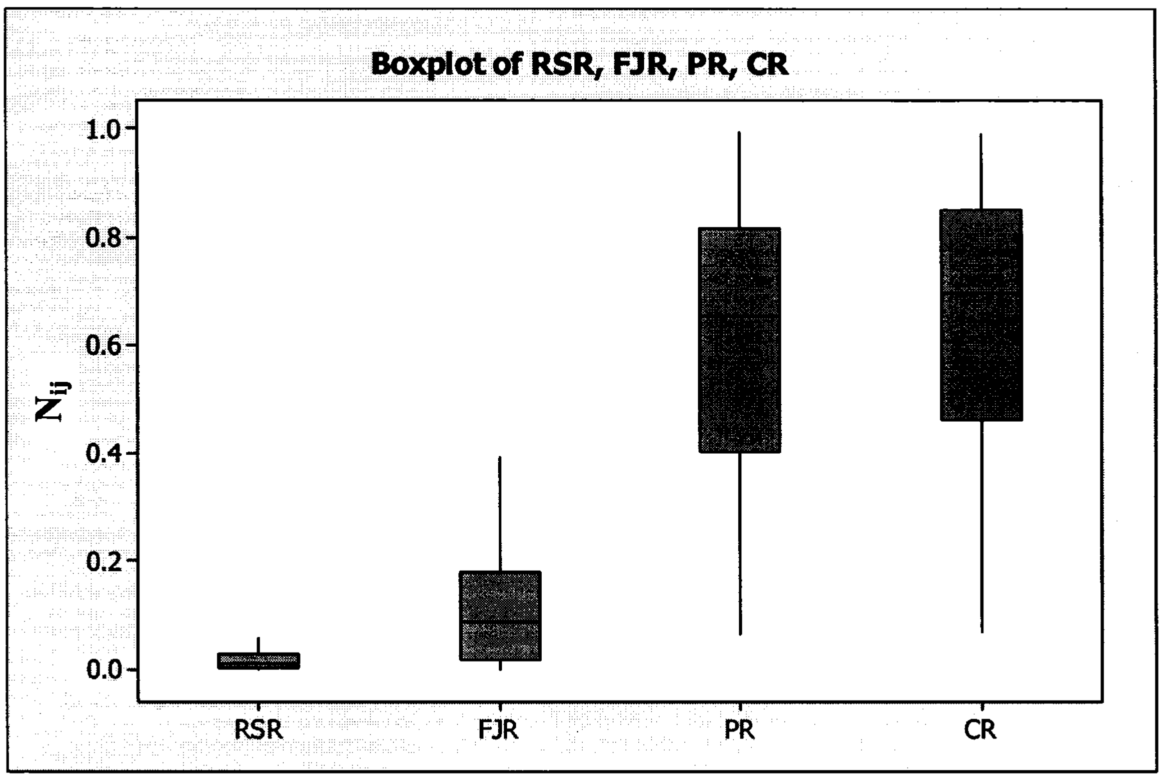

Figure 21. *CPU* Boxplot

Table 85. *CPU* Performance among the rules

	CPU			
Run	RSR	FJR	PR	CR
1	0.0169	0.04225	0.870447	0.490154
2	0.00216	0.22384	0.872494	0.434332
3	0.00991	0.096	0.592383	0.799856
4	0.47285	0.4227	0.72361	0.272249
5	0.03999	0.19241	0.769633	0.607491
6	0.05944	0.99061	0.067362	0.103024
7	0.01799	0.04452	0.695297	0.717117
8	0.01468	0.00597	0.663635	0.747888
9	0.02024	0.02024	0.273272	0.961511
10	0.00143	0.01115	0.955518	0.294718
11	0.00703	0.00154	0.164264	0.98639
12	0.00412	0.88089	0.45691	0.123489
13	0.00109	0.09947	0.787129	0.608713
14	0.02655	0.03186	0.331839	0.942424
15	0.00166	0.95824	0.130857	0.254261
16	0.00298	0.0352	0.218228	0.975258
17	0.01373	0.51763	0.314937	0.795417
18	0.07919	0.96192	0.168578	0.200071
19	0.19708	0.47026	0.857262	0.071547
20	0.00542	0.31069	0.691774	0.651838
21	0.00407	0.08235	0.267174	0.960115
22	0.00368	0.0275	0.87467	0.483924
23	0.00487	0.01045	0.971495	0.23678
24	0.20669	0.0095	0.515537	0.831511
25	0.66755	0.02302	0.391322	0.633021
26	0.04888	0.00707	0.716185	0.69616
27	0.02595	0.01257	0.54257	0.839516
28	0.01027	0.02055	0.871574	0.489725
29	0.01073	0.00939	0.472732	0.881091
30	0.0128	0.01359	0.266791	0.963574
31	0.00596	0.01107	0.307835	0.951357
32	0.00364	0.0182	0.9002	0.435081
33	0.00966	0.24072	0.307104	0.920677
34	0.01963	0.09012	0.693304	0.714719
35	0.00987	0.06379	0.470549	0.88001
36	0.01509	0.03233	0.959268	0.280236
37	0.01255	0.0405	0.405332	0.913185
38	0.00157	0.11537	0.850598	0.513002
39	0.0017	0.10059	0.604217	0.790443
40	0.02186	0.10932	0.765222	0.634041
41	0.00234	0.14272	0.729994	0.668381
42	0.00338	0.1387	0.772987	0.619066
43	0.00138	0.11758	0.413869	0.90271
44	0.00135	0.12633	0.894815	0.428187
45	0.01059	0.01722	0.655553	0.754879
46	0.00059	0.15266	0.579433	0.800595
47	0.00235	0.27947	0.512324	0.812043
48	0.0061	0.13402	0.55801	0.818918
49	0.00125	0.08254	0.883636	0.460839
50	0.0047	0.01503	0.364499	0.931071
51	0.00231	0.01332	0.941754	0.33603
52	0.00029	0.09295	0.993131	0.071059
53	0.02664	0.10703	0.720887	0.684219
54	0.00164	0.16519	0.952688	0.25514
55	0.09206	0.28753	0.408598	0.861335
56	0.00212	0.03885	0.777413	0.627786
57	0.01259	0.16006	0.853254	0.496166
58	0.06923	0.00997	0.945359	0.318442
59	0.03463	0.07878	0.678373	0.72966
60	0.00076	0.0009	0.406587	0.913611
61	0.03344	0.1449	0.722259	0.675446
62	0.01921	0.16676	0.947632	0.271689
63	0.03251	0.08743	0.706819	0.701218
64	0.01163	0.04757	0.569774	0.820341
65	0.01881	0.08155	0.126032	0.988489
66	0.01153	0.01371	0.404485	0.914369
67	0.16607	0.60124	0.265105	0.735292
68	0.36277	0.6382	0.295588	0.611331
69	0.00853	0.22536	0.720742	0.655494
70	0.10817	0.01462	0.630011	0.768877
71	0.00673	0.01536	0.406847	0.913342
72	0.03843	0.39452	0.600745	0.694251
73	0.12927	0.39259	0.780396	0.469195

Table 86. One-Way ANOVA for *CPU Time*

Anova: Single Factor

SUMMARY

Groups	Count	Sum	Average	Variance
RSR	73	3.314858	0.045409	0.011485
FJR	73	12.34003	0.169042	0.056456
PR	73	43.97871	0.602448	0.063979
CR	73	46.79539	0.641033	0.066292

ANOVA

Source of Variation	SS	df	MS	F	P-value	F crit
Between Groups	19.93723	3	6.645742	134.1141	2.21E-54	2.635951
Within Groups	14.27123	288	0.049553			
Total	34.20846	291				

As the p-value in Table 86 is less than 0.05, we can reject the hypothesis that all the means are equal, i.e. there is a significant difference between the performances of the rules. Next, a two-tailed two-sample t test was conducted for *FJR* – *RSR* to determine if the difference between them is statistically significant (Table 87). As the p-value is less than 0.05, we can reject the hypothesis that the means are equal and conclude that the difference between *FJR* and *RSR* is statistically significant. Moreover, as the difference is greater than zero, we conclude that *RSR* performed better than *FJR* in the case of *CPU*. This is expected as *RSR* is very simple and requires very little computation. It is evident from Figure 21 that *FJR* performs better than both *PR* and *CR* in the case of *CPU*. On the other hand, a t test is needed to check if the difference between *CR* and *PR* is statistically significant. This is shown in Table 87.

Table 87. t test for *FJR* - *RSR* in the case of *CPU*

Two-sample T for FJR vs RSR

	N	Mean	StDev	SE Mean
FJR	73	0.169	0.238	0.028
RSR	73	0.045	0.107	0.013

Difference = mu (FJR) - mu (RSR)
Estimate for difference: 0.123633
95% CI for difference: (0.063107, 0.184158)
T-Test of difference = 0 (vs not =): T-Value = 4.05 P-Value = 0.000 DF = 100

From Table 88, even though the *PR* mean is smaller than *CR* mean (indicating that *PR* performed better), this difference is not statistically significant as the 95% Confidence Interval overlaps with zero. Moreover, the p-value is greater than 0.05.

Table 88. t test for *CR* – *PR* in the case of *CPU*

Two-sample T for CR vs PR

	N	Mean	StDev	SE Mean
CR	73	0.641	0.257	0.03
PR	73	0.602	0.253	0.03

Difference = mu (CR) - mu (PR)
Estimate for difference: 0.038585
95% CI for difference: (-0.044918, 0.122087)
T-Test of difference = 0 (vs not =): T-Value = 0.91 P-Value = 0.363 DF = 143

Based on the previous tests, we conclude that for the *CPU*, the best performance was achieved by *RSR*, followed by *FJR*, then finally *PR* and *CR* that had the worst performance.

This conclusion was expected as both *RSR* and *FJR* are heuristics that do not involve MIP solutions.

Match-up Comparison

The *Match-up* performance of the four rules is presented in Table 89. The boxplot of the rules is also shown in Figure 22. Visually it seems that *FJR* performed the best, followed by *PR* and *RSR*, then *CR*. The same tests applied earlier will be used to determine if the differences are statistically significant. The ANOVA results are shown in Table 90.

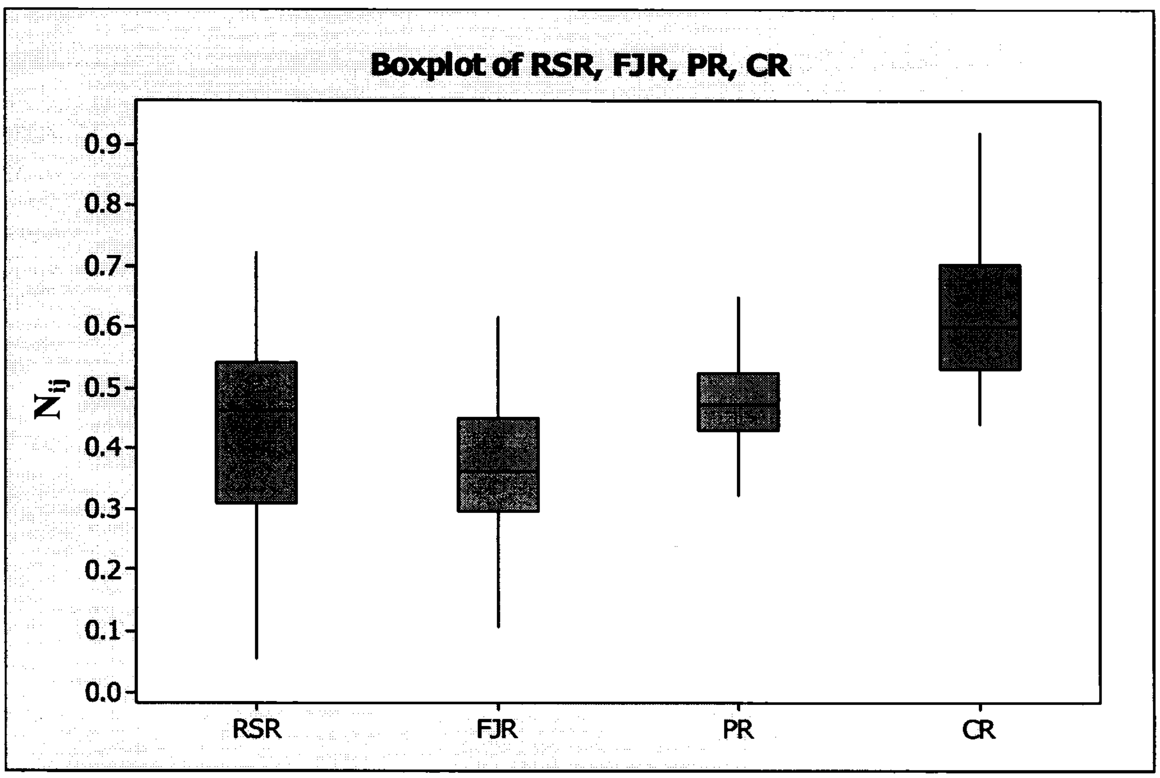

Figure 22. *Match-up* Boxplot

Table 89. *Match-up* Performance among the rules

	Match-Up Time			
Run	RSR	FJR	PR	CR
1	0.476967	0.414145	0.465358	0.620023
2	0.584529	0.290375	0.44035	0.616522
3	0.565442	0.317947	0.440307	0.620737
4	0.529429	0.383421	0.535114	0.535114
5	0.484146	0.342812	0.562331	0.576078
6	0.17397	0.357344	0.648861	0.648861
7	0.35057	0.284838	0.626279	0.635408
8	0.075708	0.181602	0.428067	0.88207
9	0.290396	0.617752	0.469914	0.559673
10	0.535309	0.258388	0.382759	0.707231
11	0.462723	0.454564	0.486034	0.585688
12	0.377845	0.61122	0.491754	0.491754
13	0.342341	0.219983	0.430357	0.805732
14	0.274775	0.456071	0.716682	0.450405
15	0.617782	0.327443	0.449887	0.555633
16	0.490707	0.409631	0.321267	0.698711
17	0.287933	0.544873	0.55687	0.55687
18	0.278456	0.172129	0.222382	0.918357
19	0.503486	0.319348	0.434749	0.674916
20	0.398961	0.353793	0.461483	0.709009
21	0.435842	0.34846	0.375367	0.740079
22	0.195617	0.169484	0.369001	0.892663
23	0.387313	0.501311	0.55177	0.542426
24	0.393181	0.482097	0.543067	0.563977
25	0.499899	0.508094	0.508094	0.483509
26	0.487907	0.379273	0.454497	0.641507
27	0.424763	0.222595	0.412435	0.774548
28	0.586789	0.437412	0.473217	0.490321
29	0.576056	0.48714	0.48638	0.440782
30	0.103413	0.12669	0.371898	0.913755
31	0.228346	0.291716	0.409969	0.833478
32	0.527821	0.354545	0.430396	0.640673
33	0.28962	0.242256	0.491201	0.784954
34	0.325656	0.494918	0.569156	0.570146
35	0.054941	0.382258	0.550411	0.740208
36	0.066983	0.576054	0.576054	0.576054
37	0.202995	0.189359	0.385204	0.880088
38	0.513499	0.317961	0.41672	0.679385
39	0.373324	0.394452	0.474922	0.692449
40	0.087706	0.55547	0.584705	0.584705
41	0.473982	0.480454	0.521778	0.521778
42	0.722842	0.304849	0.423428	0.453073
43	0.523265	0.354178	0.385266	0.672549
44	0.163978	0.106265	0.408867	0.89143
45	0.600311	0.388773	0.460239	0.525987
46	0.293105	0.375071	0.536105	0.697139
47	0.623908	0.335245	0.428191	0.56125
48	0.496903	0.371854	0.445279	0.645398
49	0.491164	0.244498	0.447926	0.705933
50	0.513367	0.35964	0.496916	0.600157
51	0.721324	0.301265	0.420803	0.460277
52	0.489386	0.314187	0.48289	0.654679
53	0.549074	0.400877	0.502873	0.533792
54	0.458744	0.405163	0.506774	0.607105
55	0.531535	0.486826	0.490138	0.490138
56	0.575065	0.380968	0.505798	0.518007
57	0.561421	0.277135	0.435931	0.646503
58	0.457051	0.458897	0.523069	0.554001
59	0.38419	0.466486	0.538548	0.587158
60	0.582519	0.300685	0.534447	0.533504
61	0.644804	0.446403	0.430903	0.446403
62	0.432074	0.436684	0.516963	0.596128
63	0.441529	0.412574	0.53655	0.589023
64	0.414962	0.235102	0.451838	0.753907
65	0.264317	0.210299	0.285175	0.896987
66	0.177481	0.275877	0.527872	0.783418
67	0.48795	0.473942	0.515967	0.520636
68	0.419805	0.574216	0.497011	0.497011
69	0.585874	0.310291	0.526805	0.531928
70	0.590883	0.413303	0.446205	0.530037
71	0.454848	0.353405	0.446286	0.68487
72	0.595284	0.408498	0.472279	0.505687
73	0.646088	0.360452	0.46919	0.482188

Table 90. One-Way ANOVA for *Match-up Time*

Anova: Single Factor

SUMMARY

Groups	*Count*	*Sum*	*Average*	*Variance*
RSR	73	31.23218	0.427838	0.025998
FJR	73	26.80359	0.367172	0.012805
PR	73	34.62358	0.474296	0.006026
CR	73	45.99258	0.630035	0.016247

ANOVA

Source of Variation	*SS*	*df*	*MS*	*F*	*P-value*	*F crit*
Between Groups	2.765779	3	0.921926	60.38014	2.59E-30	2.635951
Within Groups	4.397386	288	0.015269			
Total	7.163165	291				

As the p-value in Table 90 is less than 0.05, we can reject the hypothesis that all the means are equal, i.e. there is a significant difference between the performances of the rules. Next, a two-tailed two-sample t test was conducted for *FJR – RSR* to determine if the difference between them is statistically significant (Table 91). As the p-value is less than 0.05, we can reject the hypothesis that the means are equal and conclude that the difference between *FJR* and *RSR* is statistically significant. Moreover, as the difference is greater than zero, we conclude that *FJR* performed better than *RSR* in the case of *Match-up Time*. Another t test is conducted for *RSR – PR* and the results are shown in Table 92. The small p-value and the negative difference indicate that *RSR* outperformed *PR* in the *Match-up Time* and the difference is statistically significant. Finally, a t test was carried out for CR – PR, and the results indicate that PR outperformed CR in the *Match-up Time* (Table 93).

Table 91. t test for *RSR – FJR* in the case of *Match-up Time*

Two-sample T for RSR vs FJR

	N	Mean	StDev	SE Mean
RSR	73	0.428	0.161	0.019
FJR	73	0.367	0.113	0.013

Difference = mu (RSR) - mu (FJR)
Estimate for difference: 0.060666
95% CI for difference: (0.015051, 0.106281)
T-Test of difference = 0 (vs not =): T-Value = 2.63 P-Value = 0.010 DF = 129

Table 92. t test for *RSR – PR* in the case of *Match-up Time*

Two-sample T for RSR vs PR

	N	Mean	StDev	SE Mean
RSR	73	0.428	0.161	0.019
PR	73	0.4743	0.0776	0.0091

Difference = mu (RSR) - mu (PR)
Estimate for difference: -0.046458
95% CI for difference: (-0.087996, -0.004919)
T-Test of difference = 0 (vs not =): T-Value = -2.22 P-Value = 0.029 DF = 103

Table 93. t test for *CR – PR* in the case of *Match-up Time*

Two-sample T for CR vs PR

	N	Mean	StDev	SE Mean
CR	73	0.63	0.127	0.015
PR	73	0.4743	0.0776	0.0091

Difference = mu (CR) - mu (PR)
Estimate for difference: 0.155740
95% CI for difference: (0.121150, 0.190330)
T-Test of difference = 0 (vs not =): T-Value = 8.92 P-Value = 0.000 DF = 118

Based on the previous tests, we conclude that for the *Match-up Time*, the best performance was achieved by *FJR*, followed by *RSR*, then *PR*, and finally *CR* that had the worst performance.

Shifted Jobs Comparison

The *Shifted Jobs* performance of the four rules is presented in Table 94. The boxplot of the rules is also shown in Figure 23. *RSR* performed the best as the number of shifted jobs in this rule is always zero (no shifting allowed). Visually it is evident that *FJR* performed the best after *RSR*; however, a t test is conducted for *PR* – *CR*. The results shown in Table 95 indicate that *CR* performed better than *PR* and the difference is statistically significant.

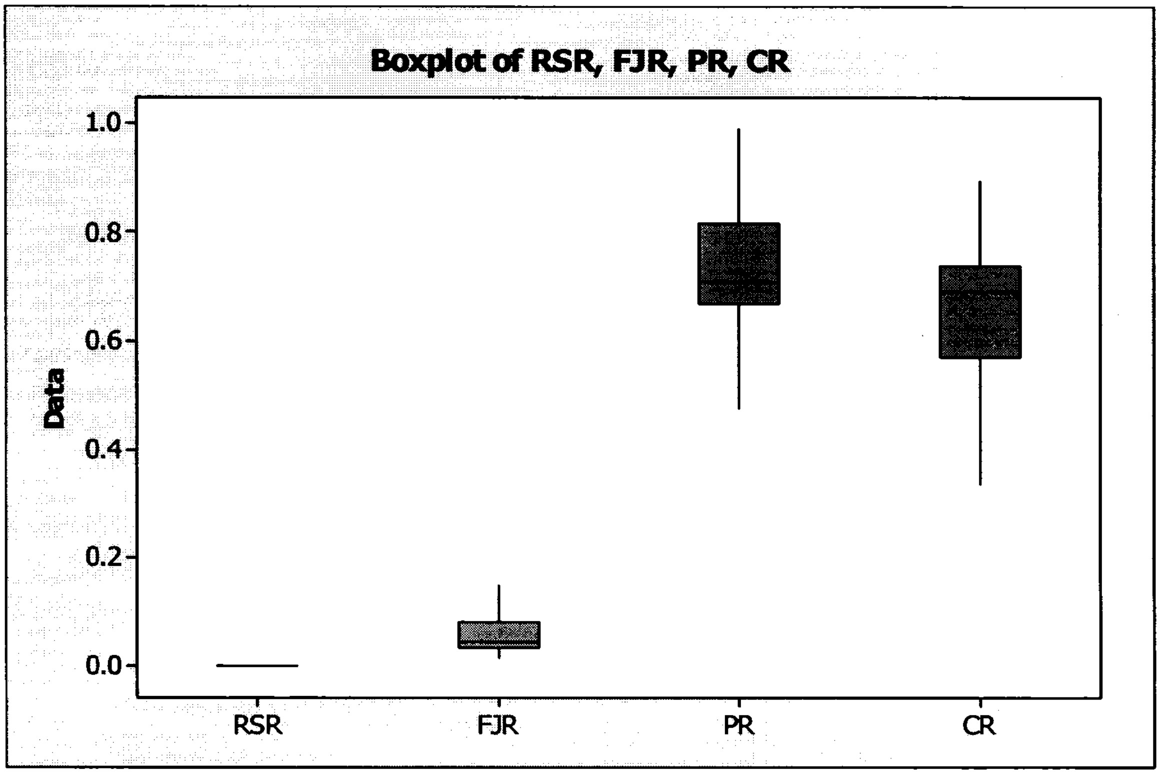

Figure 23. *Shifted Jobs* Boxplot

Table 94. *Shifted Jobs* Performance among the rules

	Shifted Jobs			
Run	**RSR**	**FJR**	**PR**	**CR**
1	0	0.077746	0.855209	0.512419
2	0	0.033215	0.936218	0.349846
3	0	0.028441	0.815322	0.578309
4	0	0.02945	0.7068	0.7068
5	0	0.018968	0.787163	0.616453
6	0	0.073458	0.705196	0.705196
7	0	0.049724	0.691157	0.720992
8	0	0.04001	0.789133	0.612918
9	0	0.109018	0.476953	0.872142
10	0	0.012139	0.986696	0.16212
11	0	0.034877	0.641737	0.766131
12	0	0.299813	0.674579	0.674579
13	0	0.037049	0.951914	0.304116
14	0	0.145071	0.743491	0.652821
15	0	0.062388	0.733875	0.676413
16	0	0.048421	0.830557	0.554825
17	0	0.166148	0.697279	0.697279
18	0	0.019028	0.987732	0.154993
19	0	0.025358	0.983794	0.177503
20	0	0.141235	0.788499	0.5986
21	0	0.040758	0.503913	0.862792
22	0	0.044734	0.916349	0.397874
23	0	0.055695	0.757449	0.650515
24	0	0.138866	0.702982	0.697519
25	0	0.118331	0.709983	0.694206
26	0	0.025728	0.688411	0.724864
27	0	0.034495	0.759989	0.64902
28	0	0.140238	0.656782	0.740925
29	0	0.031546	0.744763	0.666583
30	0	0.028864	0.754603	0.655546
31	0	0.075419	0.502378	0.861353
32	0	0.039173	0.960403	0.275847
33	0	0.060516	0.682145	0.728708
34	0	0.031467	0.733441	0.679024
35	0	0.061652	0.522163	0.850614
36	0	0.088045	0.704361	0.704361
37	0	0.038885	0.445686	0.894345
38	0	0.030314	0.959039	0.281649
39	0	0.068712	0.712707	0.698088
40	0	0.054313	0.706063	0.706063
41	0	0.137561	0.700385	0.700385
42	0	0.049518	0.623929	0.779911
43	0	0.03939	0.812264	0.581958
44	0	0.030118	0.941523	0.335601
45	0	0.140741	0.651581	0.745409
46	0	0.057054	0.591075	0.804596
47	0	0.099789	0.820141	0.563392
48	0	0.040228	0.731418	0.680741
49	0	0.038465	0.927824	0.37103
50	0	0.033754	0.613474	0.788993
51	0	0.034101	0.863616	0.502995
52	0	0.016537	0.978897	0.203685
53	0	0.035005	0.702284	0.711035
54	0	0.092965	0.680789	0.726556
55	0	0.035051	0.736077	0.675989
56	0	0.024084	0.733784	0.678955
57	0	0.032477	0.960697	0.275692
58	0	0.108679	0.665758	0.738211
59	0	0.037797	0.641534	0.766163
60	0	0.020244	0.671285	0.740923
61	0	0.207514	0.691714	0.691714
62	0	0.108348	0.776492	0.620742
63	0	0.052179	0.630711	0.774262
64	0	0.042274	0.951503	0.304721
65	0	0.02259	0.986716	0.160878
66	0	0.044548	0.668962	0.741961
67	0	0.080763	0.70641	0.70318
68	0	0.149813	0.699127	0.699127
69	0	0.030632	0.755853	0.654024
70	0	0.062571	0.633529	0.771185
71	0	0.042923	0.526877	0.848857
72	0	0.022048	0.677199	0.73547
73	0	0.056781	0.668756	0.74131

Table 95. t test for *PR – CR* in the case of *Shifted Jobs*

Two-sample T for PR vs CR

	N	Mean	StDev	SE Mean
PR	73	0.744	0.133	0.016
CR	73	0.621	0.192	0.022

Difference = mu (PR) - mu (CR)
Estimate for difference: 0.123220
95% CI for difference: (0.069133, 0.177308)
T-Test of difference = 0 (vs not =): T-Value = 4.51 P-Value = 0.000 DF = 128

Based on the previous tests, we conclude that for the *Shifted Jobs*, the best performance was achieved by *RSR*, followed by *FJR*, then *CR*, and finally *PR* that had the worst performance.

Overall Performance Comparison

The *overall performance* including all the performance measures for each rule is presented in Table 96 and computed by summing for each rule its performance values for the four performance measures, then dividing by 4; i.e. *overall performance* of rule j = $\frac{\sum_{i=1}^{4} N_{ij}}{4}$.

The boxplot of the rules is also shown in Figure 24. Visually it seems that *FJR* performed the best, followed by *RSR*, then *PR* and *CR*. The same tests implemented earlier will be used to determine if the differences are statistically significant. The ANOVA results are shown in Table 97. As the p-value is less than 0.05, we can reject the hypothesis that all the means are equal, i.e. there is a significant difference between the performances of the rules.

Next, a two-tailed two-sample t test was conducted for *RSR – FJR* (Table 98). As the p-value is less than 0.05, we can reject the hypothesis that the means are equal and conclude that the difference between *FJR* and *RSR* is statistically significant. Moreover, as the difference is greater than zero, we conclude that *FJR* performed better than *RSR* in the case of *Overall Performance*. Another t test is conducted for *CR – PR* and the results are shown in Table 99. Even though the difference was positive indicating that *PR* outperformed *CR*, this difference was not statistically significant.

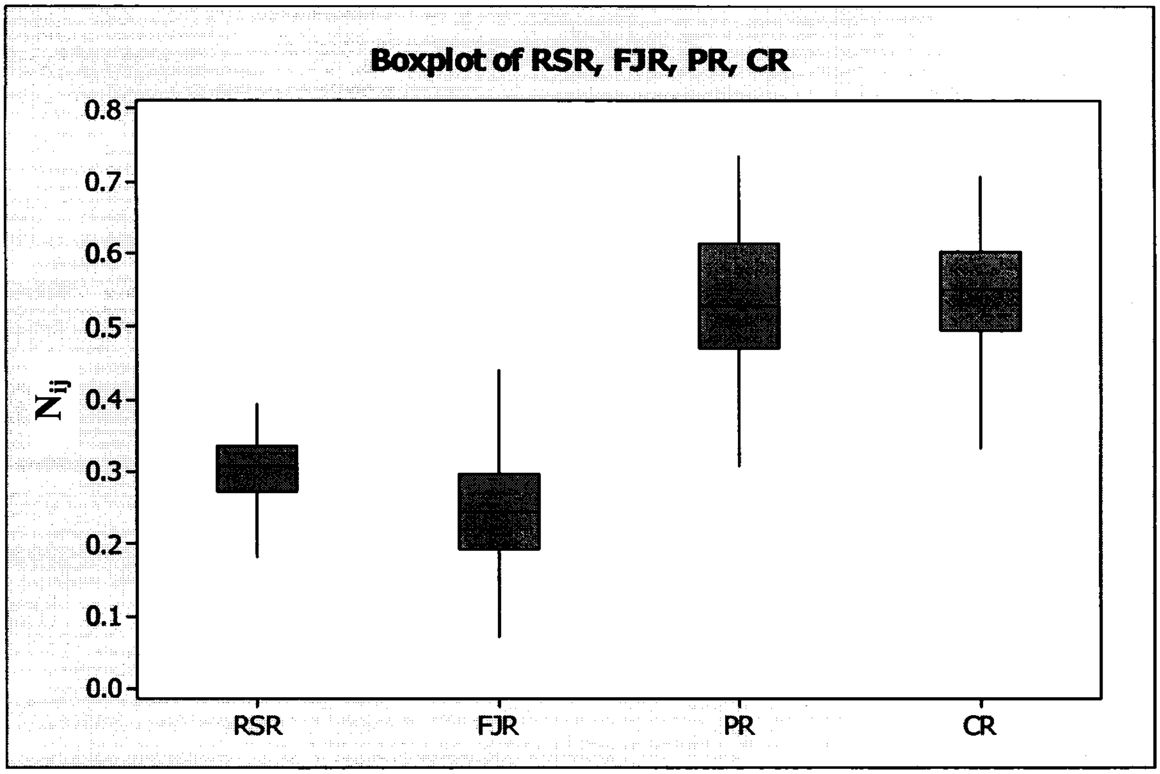

Figure 24. *Overall Performance* Boxplot

Table 96. *Overall Performance* among the rules

	Overall Performance			
Run	RSR	FJR	PR	CR
1	0.306095	0.237064	0.650553	0.494312
2	0.338774	0.230089	0.656862	0.439375
3	0.330042	0.237592	0.5214	0.590126
4	0.393171	0.328365	0.609474	0.496634
5	0.308208	0.267842	0.619219	0.529964
6	0.180996	0.459504	0.490653	0.499568
7	0.245843	0.169653	0.638614	0.640554
8	0.239481	0.129038	0.567927	0.587328
9	0.231376	0.326099	0.391003	0.708165
10	0.291738	0.17068	0.696235	0.411024
11	0.297396	0.272671	0.378779	0.651839
12	0.254246	0.600621	0.489469	0.406114
13	0.280613	0.172051	0.653462	0.502771
14	0.261977	0.257093	0.562357	0.580813
15	0.345571	0.441602	0.415661	0.458876
16	0.281278	0.302122	0.386409	0.617876
17	0.30132	0.363952	0.456466	0.576586
18	0.284462	0.361677	0.467673	0.381107
19	0.316372	0.318766	0.704797	0.335246
20	0.305168	0.287847	0.576667	0.561019
21	0.296026	0.251534	0.36395	0.704345
22	0.250959	0.130045	0.652106	0.511671
23	0.278783	0.254386	0.666161	0.446651
24	0.369848	0.248212	0.494675	0.578013
25	0.483745	0.258673	0.482363	0.5527
26	0.32386	0.234449	0.521145	0.593583
27	0.324034	0.149435	0.501745	0.641748
28	0.366555	0.239969	0.55813	0.491694
29	0.336297	0.277081	0.476543	0.551434
30	0.257322	0.071475	0.444843	0.648214
31	0.307843	0.10872	0.308742	0.673868
32	0.282372	0.2272	0.690629	0.441929
33	0.308128	0.186017	0.420969	0.663055
34	0.20568	0.318068	0.594913	0.601301
35	0.266102	0.134039	0.385781	0.617708
36	0.151929	0.305004	0.678448	0.50869
37	0.303665	0.077724	0.309055	0.671904
38	0.295243	0.225666	0.66103	0.477282
39	0.292011	0.275385	0.496089	0.598186
40	0.244322	0.253237	0.584867	0.552072
41	0.356588	0.240329	0.530323	0.51492
42	0.386991	0.246313	0.507113	0.512499
43	0.334912	0.237241	0.466827	0.609394
44	0.231863	0.126884	0.700503	0.469258
45	0.390045	0.175235	0.487001	0.558092
46	0.28576	0.229785	0.490226	0.655489
47	0.365976	0.258038	0.520882	0.560497
48	0.306812	0.261343	0.528783	0.607627
49	0.315051	0.178711	0.666943	0.471657
50	0.315889	0.244956	0.436989	0.632001
51	0.394804	0.196167	0.611838	0.367377
52	0.254958	0.227851	0.735325	0.355971
53	0.322418	0.296204	0.534591	0.527778
54	0.266665	0.293161	0.63857	0.509446
55	0.329124	0.333686	0.495882	0.594299
56	0.331847	0.237119	0.583725	0.527572
57	0.335203	0.210901	0.65983	0.441383
58	0.365844	0.204001	0.584445	0.441012
59	0.247319	0.304134	0.548592	0.62089
60	0.366464	0.172576	0.469317	0.577716
61	0.357275	0.309204	0.544462	0.544734
62	0.264076	0.305934	0.683631	0.461716
63	0.318504	0.241438	0.545115	0.593232
64	0.314341	0.152035	0.593807	0.534895
65	0.225303	0.188961	0.4846	0.60208
66	0.287113	0.134126	0.406883	0.658564
67	0.343826	0.398475	0.46635	0.585013
68	0.388238	0.412857	0.473381	0.552316
69	0.345932	0.281199	0.545605	0.505764
70	0.334117	0.276665	0.508562	0.599961
71	0.327858	0.204849	0.396794	0.677245
72	0.335973	0.31121	0.527978	0.592425
73	0.387836	0.293021	0.568963	0.516312

Table 97. One-Way ANOVA for the *Overall Performance*

Anova: Single Factor

SUMMARY

Groups	Count	Sum	Average	Variance
RSR	73	22.47377	0.30786	0.002995
FJR	73	18.34526	0.251305	0.007835
PR	73	39.1697	0.536571	0.010081
CR	73	39.97448	0.547596	0.007287

ANOVA

Source of Variation	SS	df	MS	F	P-value	F crit
Between Groups	5.151378	3	1.717126	243.5784	1.13E-78	2.635951
Within Groups	2.030279	288	0.00705			
Total	7.181657	291				

Table 98. t test for *RSR – FJR* in the case of *Overall Performance*

Two-sample T for RSR vs FJR

	N	Mean	StDev	SE Mean
RSR	73	0.3079	0.0547	0.0064
FJR	73	0.2513	0.0885	0.01

Difference = mu (RSR) - mu (FJR)
Estimate for difference: 0.056555
95% CI for difference: (0.032440, 0.080671)
T-Test of difference = 0 (vs not =): T-Value = 4.64 P-Value = 0.000 DF = 120

Following this, we conclude that for the *Overall Performance*, the best performance was achieved by *FJR*, followed by *RSR*, then *PR* and *CR*.

Table 99. t test for *CR – PR* in the case of *Overall Performance*

Two-sample T for CR vs PR

	N	Mean	StDev	SE Mean
CR	73	0.5476	0.0854	0.01
PR	73	0.537	0.1	0.012

Difference = mu (CR) - mu (PR)
Estimate for difference: 0.011024
95% CI for difference: (-0.019471, 0.041520)
T-Test of difference = 0 (vs not =): T-Value = 0.71 P-Value = 0.476 DF = 140

Computational Tests Summary

In this chapter, new repair and rescheduling rules have been introduced for the unrelated parallel machine problem. The rules have been compared to existing ones and evaluated based on four performance measures: *Cmax Difference*, *CPU Time*, *Match-up Time*, and *Shifted Jobs*. Extensive computational tests indicated the following conclusions about each rule:

Right Shift Repair (RSR)

RSR has been used frequently in the literature to compare with rescheduling and repair rules. *RSR* had the worst *Cmax Difference* performance among all rules, the best *CPU* and *Shifted Jobs* performances, and was the second best in the *Match-up Time* and *overall* performances. Recall that *RSR* performed the best in the case of *Shifted Jobs* because it does not shift jobs between machines. Moreover, *RSR* was the finest in *CPU Time* as it is a simple heuristic with a computational complexity of O(mn) at the most.

From the experimental design and its factor analyses, the following was determined about *RSR* performance:

- *Cmax Difference* improves when the number of machines increases.
- *CPU Time* improves when the time between breakdowns increases and the number of jobs decreases.
- *Match-up Time* decreases when the number of machines, idle time, and time between breakdowns increase and the number of jobs decreases.
- *Shifted Jobs* is always zero when using *RSR*.

Fit Job Repair (FJR)

FJR is a new repair rule introduced in this Dissertation. It ranked 3rd between the rules in the case of *Cmax Difference* (after *CR* and *PR*), 2nd for *CPU Time* and *Shifted Jobs* (after *RSR*), and was the best in the case of *Match-up Time* and *Overall Performance*.

The following was determined from the DoE factor analyses about *FJR* performance:

- *Cmax Difference* improves when the number of machines increases.
- *CPU Time* improves when the time between breakdowns and the number of machines increase and the number of jobs decreases.

- *Match-up Time* decreases when the number of machines, idle time, and time between breakdowns increase and the number of jobs decreases.
- *Shifted Jobs* declines when the number of jobs and repair durations decrease and the number of machines and the time between breakdowns increase.

Partial Rescheduling (PR)

PR is a new repair rule introduced in this Dissertation. It ranked 1st among all rules in case of *Cmax Difference* (tied with *CR*), 3rd for *Match-up Time* (after *FJR* and *RSR*), and was the worst in the case of *CPU Time* (tied with *CR*), *Shifted Jobs*, and *Overall Performance* (tied with *CR*).

The following was determined from the experimental design factor analyses about *PR* performance:

- *Cmax Difference* improves when the number of machines increases.
- *CPU Time* improves when the time between breakdowns increases and the number of jobs decreases.
- *Match-up Time* decreases when the number of machines and time between breakdowns increase and the number of jobs decreases.
- *Shifted Jobs* declines when the number of jobs and repair durations decrease and the number of machines and the time between breakdowns increase.

Complete Rescheduling (CR)

CR ranked 1st among all rules in the case of *Cmax Difference* (tied with *PR*), 3rd for *Shifted Jobs*, and was the worst in the case of *CPU Time* (tied with *PR*), *Match-up Time*, and *Overall Performance* (tied with *PR*).

The following was determined from the DoE factor analyses about *CR* performance:

- *Cmax Difference* improves when the number of machines increases.
- *CPU Time* improves when the time between breakdowns increases and the number of jobs and machines decreases.
- *Match-up Time* decreases when the number of machines and time between breakdowns increase and the number of jobs decreases.
- *Shifted Jobs* declines when the number of jobs and repair durations decrease and the number of machines and the time between breakdowns increase.

Finally, as it is obvious that the superiority of each of the four rules strongly depends on which performance measure is being evaluated, Table 100 below summarizes the ranks of the rules for all possible combinations of the four performance measures addressed in this dissertation (15 alternatives). All necessary ANOVA and t tests were carried out to make sure that the reported results are statistically significant. Note that the rules are ranked between 1 and 4, where 1 indicates the best performance and 4 the worst one.

Table 100. Ranks of the Rules for all combinations of Performance Measures
(4 is worst and 1 is best)

Performance Measures				**Repair Rules**			
Cmax Difference	***CPU Time***	***Match-up Time***	***Shifted Jobs***	***RSR***	***FJR***	***PR***	***CR***
●				4	3	1	1
	●			1	2	4	4
		●		2	1	3	4
			●	1	2	4	3
●	●			2	1	4	4
●		●		4	1	1	3
●			●	2	1	4	3
	●	●		1	1	3	4
	●		●	1	2	4	4
		●	●	1	1	4	4
●	●	●		2	1	3	4
●	●		●	2	1	4	4
●		●	●	2	1	4	4
	●	●	●	1	2	4	4
●	●	●	●	2	1	3	3

CHAPTER VII

ROBUST REACTIVE SCHEDULING SYSTEM

The proposed robust scheduling system is presented in this chapter. The system is a combination of the repair rules described in chapter 6, with the objective of delivering superior performance measures, i.e. better schedule quality and stability.

The rationale of the system is very simple. Following the creation of a predictable schedule using *MCFJI* (explained in Chapter 5), the schedule is executed under a dynamic environment subject to breakdowns. Upon the occurrence of any disruption, the system will check its Tidle, where Tidle is the total idle time in the predictable schedule. If Tidle > 0, the system attempts to shift D_J (down job) to the right without impacting its successor. It repeats this operation one more time if necessary for the next job to start on time (this is actually *RSR*). After shifting two jobs, if the schedule is still not repaired, apply *FJR*; i.e. try to fit D_J on any machine while maximizing $residle_i$ (idle time left on each machine). In case all jobs have been shifted and D_J was not fitted on any machine, then apply *PR* (which includes *CR* in case matching up with the initial schedule is not possible).

The system's architecture is presented in Figure 25. Note that the reason only *PR* is used when Tidle = 0 is because both *RSR* and *FJR* are not able to repair the schedule in the absence of idle time; *RSR* shifts the jobs to the right and *FJR* attempts to fit the jobs in the idle time between the machines.

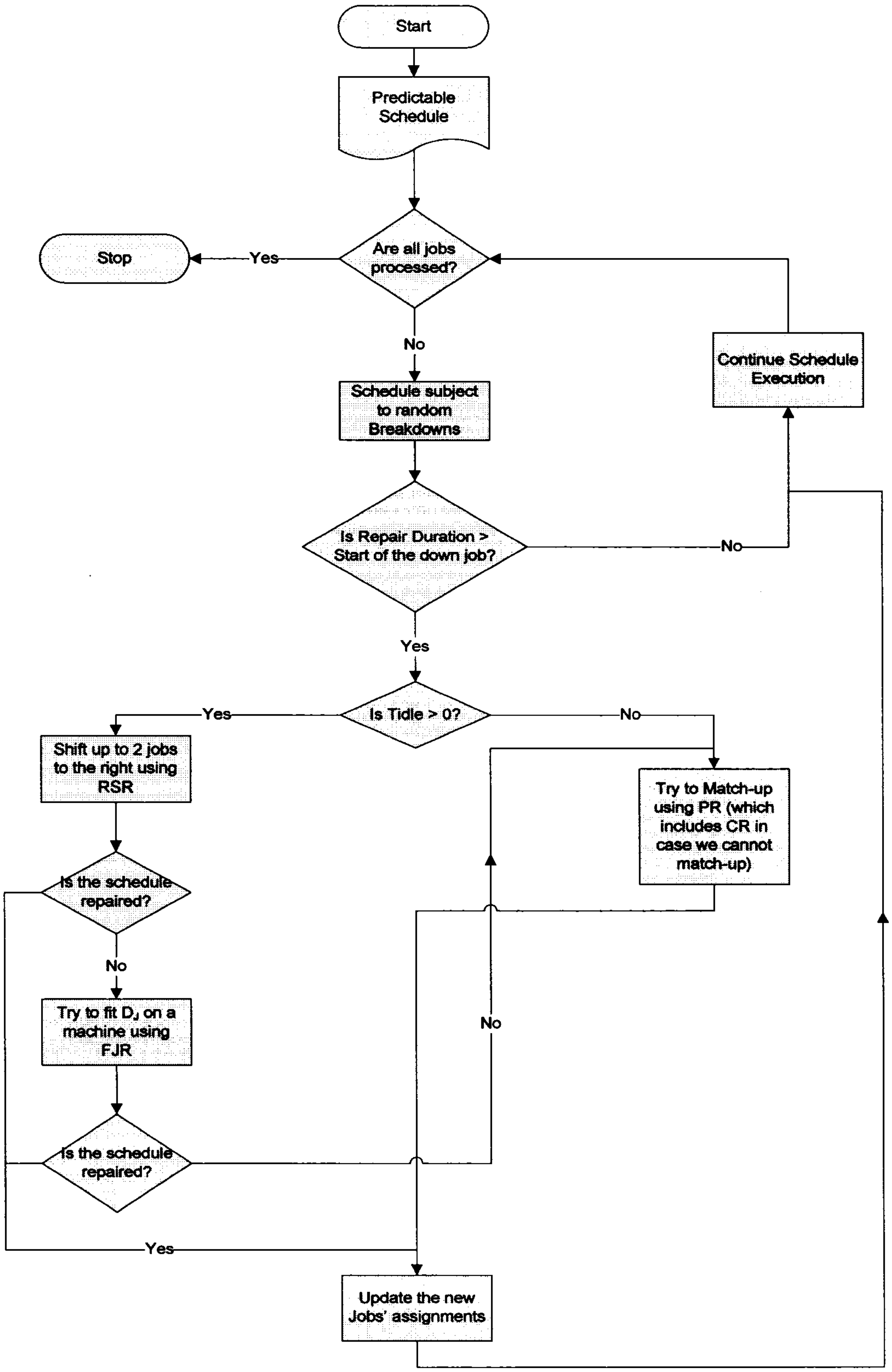

Figure 25. Robust Reactive Scheduling System Architecture

COMPUTATIONAL TESTS AND EXPERIMENTAL DESIGN

Following the description of the robust system, the same computational tests applied in chapter 6 to test the repair rules will be used here. The D-Optimal design experiments shown in Table 29 are carried out and the factors in Table 27 are analyzed to see how they impact the system performance. The Robust System is tested both with and without the learning parameter (explained in Chapter 5). The computational tests are shown in Tables 101 and 102.

Performance Measures Statistical Analyses

Similarly to chapter 6 approach, Minitab 14.2 Statistical Software was used to determine the significance of the factors and their interactions for each performance measure.

Cmax Difference Statistical Analysis

In this section, the significance of the factors and their interactions is determined for the Robust System with and without Learning in the case of the *Cmax Difference* performance measure.

Table 101. Computational Tests for the Robust System w/o Learning
(Average Numbers)

	Robust (no learning)							
Run	Cmax	Cmax 95% CI	CPU (sec)	CPU 95% CI	Match	Match 95% CI	Shifted Jobs	S. Jobs 95% CI
1	9.4	[4.5-14.3]	0.52	[0.2-0.8]	202.22	[125.5-278.9]	1.24	[0.35-2.1]
2	291.58	[264.7-318.5]	76.63	[62.3-91]	9299.75	[8498.7-10100.8]	29.66	[24-35.3]
3	13.67	[5.2-22.1]	7.72	[0.05-15.4]	969.45	[754.1-1184.8]	8.4	[4.1-12.7]
4	20.07	[6.8-33.3]	0.49	[0.1-0.9]	511.85	[281.3-742.3]	0.65	[0-1.3]
5	2.57	[0.8-4.3]	1.74	[0.4-3.05]	147.55	[104.1-190.9]	3.1	[1.3-4.9]
6	2.77	[0.9-4.6]	0.17	[0.1-0.2]	14.69	[7-22.3]	0.48	[0.14-0.8]
7	3.43	[1.3-5.6]	2.23	[0.3-4.2]	45.76	[26.1-65.4]	1.9	[0.8-3]
8	2.28	[0-4.6]	18.19	[8.4-28]	426.24	[367.6-484.9]	7.58	[4.8-10.4]
9	4.28	[1.4-7.1]	1.29	[0.7-1.9]	10.58	[4.6-16.5]	0.64	[0.3-1]
10	174.51	[137.5-211.5]	79.45	[45.6-113.3]	7892.24	[6382.5-9402]	9.9	[6-13.8]
11	2.67	[1.4-3.9]	32.94	[8.3-57.6]	105.28	[81.1-129.4]	7.54	[4.8-10.2]
12	4.82	[0.6-9.1]	1.19	[0.7-1.7]	28	[12.2-43.8]	0.38	[0.05-0.7]
13	271.73	[231-312.4]	26.67	[20.8-32.6]	9588.46	[8215-10961.9]	18.56	[14.5-22.6]
14	4.13	[1-7.3]	1.09	[0.6-1.5]	19.65	[6.3-33]	0.96	[0.2-1.7]
15	2.65	[1-4.2]	1.46	[0.6-2.3]	72.14	[43.7-100.6]	2.15	[1-3.3]
16	3.63	[0-7.4]	4.84	[0-10.9]	504.05	[298.9-709.1]	5.6	[0.9-10.3]
17	8.49	[4.4-12.5]	1.4	[1-1.7]	113.75	[85.7-141.8]	4.64	[3.4-5.8]
18	26.5	[7.5-45.5]	5.62	[1.8-9.4]	2068.88	[1369.1-2768.6]	1.9	[1-2.8]
19	25.78	[13.2-38.4]	8.35	[4.8-11.9]	1402.04	[1110.6-1693.5]	3.87	[0.4-7.3]
20	92.16	[67.9-116.4]	21.3	[14.8-27.8]	1921.2	[1543.3-2299.1]	5.12	[3.8-6.4]
21	3.94	[0.7-7.2]	11.3	[5-17.6]	243.53	[173.9-313.2]	3.72	[1.9-5.5]
22	116.7	[86.7-146.7]	5.48	[3.8-7.2]	4478.6	[3545.8-5411.4]	12.3	[8.9-15.6]
23	7.63	[3.1-12.2]	1.95	[1.2-2.7]	142.95	[95.9-190]	1.43	[0.6-2.2]
24	15.19	[6.3-24.05]	7.62	[3.9-11.3]	215.88	[141.2-290.6]	4.77	[2.8-6.7]
25	0.93	[0-2]	0.37	[0.1-0.6]	8.71	[1.1-16.3]	0.27	[0-0.57]
26	7.93	[1.4-14.4]	2.51	[0.4-4.7]	291.41	[160.6-422.2]	4.95	[1.8-8.1]
27	27.55	[18.4-36.6]	16.98	[7.1-26.8]	1403.05	[1078-1728]	22.25	[11.4-33.1]
28	15.55	[9.2-21.9]	2.99	[2.1-3.9]	182.73	[124.8-240.6]	4.7	[2.7-6.6]
29	2.68	[0.01-5.3]	2.99	[1.3-4.7]	85.74	[53.6-117.9]	4	[1.8-6.2]
30	0	[0-1.3]	1.46	[0.4-2.5]	1064.13	[815.8-1312.4]	5.15	[2.1-8.2]
31	1.53	[0-4.4]	2.96	[1.4-4.5]	742.19	[583.4-900.9]	4.9	[3.2-6.5]
32	94.32	[71-117.6]	10.28	[6.7-13.9]	3398.15	[2755.5-4040.8]	5.71	[3.5-7.9]
33	5.42	[1.3-9.5]	2.2	[0.5-3.9]	507.97	[404.4-611.6]	5.56	[3.5-7.4]
34	2.44	[0.7-4.1]	2.7	[0.8-4.6]	84.78	[40.6-129]	2.07	[0.8-3.3]
35	0	[0-0.1]	4.71	[2.5-6.9]	87.06	[62.8-111.3]	5.1	[3.1-7.1]
36	1.18	[0.3-2]	1.68	[1-2.4]	5.46	[2.8-8.1]	0.64	[0.2-1]
37	0.82	[0-2.87]	17.65	[6.5-28.8]	282.23	[233.9-330.6]	5.48	[3.2-7.7]
38	30.44	[24.5-36.4]	29.16	[22.2-36.1]	1071.34	[902.9-1239.8]	6.11	[2.7-9.5]
39	5.46	[0-11.6]	3.68	[2.2-5.1]	399.57	[289.5-509.6]	2.95	[1.7-4.2]
40	0.82	[0-1.7]	0.36	[0-0.75]	3.95	[0-8.4]	0.52	[0-1.2]
41	5.71	[2-9.4]	8.44	[6.1-10.8]	103.85	[77.2-130.4]	7.8	[5.5-10]
42	3.11	[0.9-5.3]	2.87	[1.3-4.5]	118.84	[82.4-155.3]	1.34	[0.8-1.9]
43	9.82	[5.3-14.3]	9.4	[3.6-15.1]	656.6	[530.5-782.7]	6.65	[3.6-9.7]
44	86.9	[63-110.7]	65.28	[60.6-70]	4511.1	[3417.9-5604.2]	18.87	[14-23.7]
45	4.59	[0.8-8.4]	1.31	[0.6-2]	18.07	[5.3-30.8]	1.022	[0.2-1.8]
46	4.05	[2.3-5.8]	11.31	[6.4-16.2]	144.72	[115-174.4]	12.84	[9.1-16.5]
47	96.62	[79.5-113.7]	4.21	[3.3-5.1]	1740.15	[1476.5-2003.8]	5.66	[4.4-6.9]
48	5.57	[3.4-7.7]	36.09	[14.8-57.4]	310.02	[266.2-353.8]	8.02	[5.6-10.4]
49	455.99	[416.66-495.3]	103.6	[85.8-121.5]	21145.8	[18940.9-23350.8]	30.38	[25.9-34.9]
50	4.06	[1.3-6.8]	33.8	[12.8-54.8]	131.86	[93.6-170.1]	4.02	[2.1-5.9]
51	3.53	[1.8-5.3]	6.52	[3.4-9.6]	185.71	[135.4-236]	4.2	[2.1-6.3]
52	29.4	[22.5-36.3]	31.48	[18-45]	1747.11	[1484.6-2009.6]	6.6	[3.5-9.6]
53	4.05	[1-7.1]	39.01	[23.5-54.5]	186.52	[133.8-239.2]	3.86	[2.3-5.4]
54	36.42	[24.7-48.1]	7.82	[4.7-10.9]	647.88	[499.2-796.5]	1.42	[0.9-1.9]
55	3.41	[1.3-5.5]	1.97	[0.6-3.3]	39.7	[22.3-57.1]	2.16	[0.8-3.5]
56	4.6	[2.8-6.4]	62.68	[45.3-80.1]	228.39	[185.1-271.6]	7.2	[4.8-9.6]
57	135.65	[122-149.3]	73.32	[55.6-91]	4568.32	[4077-5059.6]	25.84	[21.1-30.6]
58	4.92	[2.7-7.2]	8.48	[6.4-10.5]	92.36	[70.1-114.7]	5.22	[3.9-6.5]
59	5.19	[1.3-9]	8.88	[4.4-13.3]	147.63	[100.8-194.4]	4.33	[2.4-6.2]
60	1.45	[0.2-2.7]	44.63	[33-56.26]	73.33	[53.5-93.1]	4.47	[2.7-6.2]
61	3.69	[1.4-5.9]	1.22	[0.8-1.6]	18.33	[11.3-25.3]	0.98	[0.6-1.4]
62	24.47	[15.8-33.2]	8.22	[5.2-11.2]	466.44	[358.1-574.8]	1.94	[1.2-2.7]
63	11.89	[8.7-15.1]	35.53	[24.1-47]	356.91	[296.9-416.8]	16.98	[12.1-21.8]
64	305.59	[252.3-358.8]	65.82	[47.2-84.5]	9863.87	[8247.1-11480.7]	18.78	[15.9-21.7]
65	34.62	[25-44.2]	47.73	[31.7-63.8]	1955.56	[1630.3-2280.8]	7.06	[5.2-8.9]
66	1.82	[0-5]	59.03	[38.2-79.8]	514.32	[417.3-611.3]	9.311	[6.3-12.3]
67	8.74	[6.6-10.8]	5.04	[3.9-6.2]	106.26	[85.3-127.2]	5.18	[3.8-6.5]
68	3.91	[1.5-6.4]	0.23	[0.1-0.4]	20.03	[8.9-31.1]	0.56	[0.2-0.9]
69	5.62	[1.9-9.3]	9.52	[3.1-15.9]	585.99	[428.6-743.38]	3.87	[1.8-5.9]
70	11.15	[6-16.3]	19.65	[12.4-26.9]	235.42	[160.5-310.3]	2.49	[1.5-3.4]
71	19.87	[13.8-25.9]	82.58	[45.7-119.5]	1048.32	[902.8-1193.8]	22.55	[18.9-26.2]
72	2.58	[1.2-4]	15.6	[6-25.1]	40.07	[26.5-53.6]	2.95	[1.6-4.3]
73	7.86	[4.5-11.2]	6.31	[3.9-8.7]	166.75	[114.7-218.8]	1.12	[0.4-1.8]

Table 102.Computational Tests for the Robust System with Learning
(Average Numbers)

	Robust (with learning)							
Run	Cmax	Cmax 95% CI	CPU (sec)	CPU 95% CI	Match	Match 95% CI	Shifted Jobs	S. Jobs 95% CI
1	7.86	[3.7-12]	0.86	[0.4-1.4]	169.86	[98.5-241.2]	1.28	[0.3-2.2]
2	23.75	[10.2-37.3]	0.98	[0.5-1.4]	1688.4	[1326.5-2050.29]	2.62	[0.95-4.3]
3	12.73	[2.2-23.2]	0.98	[0.5-1.5]	1206.96	[986-1427.9]	2.4	[0.6-4.1]
4	10.26	[1.5-19]	0.29	[0-0.6]	282.36	[122.4-442.3]	0.45	[0-0.9]
5	6.56	[3.4-9.7]	2.35	[1.1-3.6]	189	[136.4-241.6]	4.45	[2.2-6.7]
6	2.33	[0.7-3.9]	0.89	[0.4-1.3]	14.11	[6.4-21.8]	0.46	[0.1-0.8]
7	1.56	[0-3.2]	4.07	[0.6-7.6]	39.79	[22.1-57.5]	1.54	[0.4-2.6]
8	1.63	[0-3.9]	10.5	[8.3-12.7]	380.8	[308-453.6]	6.62	[4.1-9.1]
9	1.83	[0-3.8]	0.26	[0.03-0.5]	11.07	[4.8-17.4]	0.78	[0.3-1.2]
10	31.71	[9-54.4]	6.18	[1.5-10.8]	3615.09	[2496.9-4733.2]	2.55	[0.9-4.2]
11	0.52	[0-1.04]	11.53	[0.9-22.2]	93.53	[71.3-115.8]	6	[3.6-8.4]
12	2.92	[0-6.1]	0.16	[0.1-0.3]	24.58	[8.54-40.6]	0.34	[0.1-0.6]
13	134.02	[94.7-173.4]	20.93	[12.9-28.9]	5792.28	[4362-7222.5]	9.24	[6.4-12.1]
14	1.9	[0-4.8]	0.94	[0.1-1.8]	17.52	[5.2-29.8]	0.87	[0.03-1.7]
15	1.09	[0-2.6]	6.36	[2.7-10]	59.3	[29.4-89.1]	1.63	[0.6-2.6]
16	2.73	[0-6.6]	2.38	[0.6-4.1]	526.19	[325.8-726.5]	6.4	[1.4-11.4]
17	0.38	[0-1.6]	2.1	[0.2-4]	72.97	[31.9-114]	3.1	[1.1-5]
18	10.92	[0-23.8]	18.55	[2.7-34.4]	1837.92	[1278.3-2397.6]	1.6	[0.8-2.4]
19	12.95	[4.2-21.7]	11.55	[4.8-18.2]	1171.79	[873.4-1470.1]	1.3	[0.6-1.9]
20	60.05	[36.7-83.4]	5.76	[4.3-7.2]	1426.21	[1037.1-1815.3]	3.47	[2.3-4.6]
21	3.11	[0-7.8]	28.69	[13.3-44]	238.31	[168.1-308.6]	2.8	[1.42-4.2]
22	85.46	[54.9-116]	4.4	[2.7-6.1]	4234.2	[3225.3-5243]	10.8	[7.3-14.3]
23	6.09	[1.8-10.3]	4.62	[1.5-7.7]	144.31	[81.7-206.9]	0.87	[0.3-1.4]
24	2.85	[0-6.1]	2.19	[1.4-2.9]	283.74	[197-370.4]	5.6	[3.7-7.5]
25	0.7	[0-1.7]	0.12	[0.01-0.2]	8.55	[1.2-15.9]	0.27	[0-0.6]
26	4.42	[0.4-8.4]	5.76	[3.7-7.8]	201.9	[134.5-269.3]	1.43	[0.6-2.3]
27	14.72	[4.9-24.5]	6.42	[1.33-11.5]	1139.7	[822-1457.4]	11.05	[6.2-15.9]
28	6.84	[1.7-11.9]	1.68	[1.1-2.2]	160.23	[107.2-213.2]	3.3	[2.3-4.3]
29	3.42	[0.5-6.33]	2.2	[0.7-3.7]	95.59	[56.2-135]	3.57	[1.4-5.7]
30	3.97	[0-10.4]	1.47	[0.6-2.3]	1056.9	[796.3-1317.5]	5.6	[3.1-8.1]
31	1.47	[0-6]	2.55	[1.4-3.6]	788.11	[578.5-997.7]	5.23	[3.1-7.3]
32	25.31	[10.3-40.3]	1.82	[0.9-2.8]	2028.14	[1490.6-2565.7]	3.03	[1.1-5]
33	6.21	[0.1-12.3]	9.83	[4.2-15.5]	565.08	[448.4-681.8]	9	[5.2-12.8]
34	4.18	[0.8-7.5]	3.65	[1.3-6]	57.34	[25.2-89.5]	1.71	[0.6-2.8]
35	0	[0-0.24]	2.32	[0.8-3.9]	76.2	[55.8-96.6]	3.48	[1.6-5.4]
36	2.13	[0.6-3.7]	1.28	[0.3-2.2]	6.56	[2.8-10.3]	0.63	[0.2-1]
37	0.23	[0-2.5]	27.26	[15.2-39.3]	313.31	[254.6-372]	5.38	[3.5-7.2]
38	16.77	[10.4-23.1]	1.75	[0.8-2.6]	778.22	[582.6-973.8]	3.47	[1.9-5]
39	2.18	[0-6.5]	1.25	[0.5-2]	342.7	[235.9-449.5]	3.5	[1.4-5.6]
40	0.156	[0-0.6]	0.43	[0.1-0.7]	3.47	[0.7-6.3]	0.325	[0-0.6]
41	1.08	[0-3.35]	6.93	[4.7-9.2]	69	[51.3-86.7]	5.54	[4.1-7]
42	2.41	[0.2-4.6]	2.96	[1.5-4.4]	141.86	[97.8-185.9]	1.2	[0.6-1.8]
43	4.3	[0.5-8.1]	5.36	[3-7.7]	439.35	[334.5-544.1]	3.6	[1.97-5.2]
44	49.44	[30.5-68.3]	22.28	[16.3-28.3]	3041.59	[2286.9-3796.3]	11.52	[8.1-14.9]
45	2.82	[0-5.9]	1.51	[0.8-2.2]	19.38	[10.6-28.2]	1	[0.4-1.5]
46	1.78	[0.21-3.3]	17.58	[10.6-24.5]	115.03	[92.6-137.5]	9.2	[6.5-11.9]
47	44.09	[32.2-55.9]	13.78	[10.7-16.8]	970.74	[748.2-1193.3]	3.18	[2.3-4.1]
48	2.67	[0.6-4.7]	7.56	[4.5-10.6]	222.19	[169.5-274.9]	5.3	[2.7-7.9]
49	159.45	[130.4-188.5]	23.39	[18.4-28.4]	7813.32	[6311.9-9314.7]	10.04	[8-12.1]
50	1.88	[0.05-3.7]	7.61	[3.4-11.8]	116.94	[77.6-156.3]	3.67	[1.8-5.5]
51	1.76	[0.2-3.3]	10.41	[3.1-17.7]	197.38	[142-252.7]	5.77	[2.5-9.1]
52	8.74	[4.5-13]	13.28	[7.2-19.4]	887.81	[652.2-1123.4]	3.75	[1.3-6.1]
53	6.12	[1.7-10.5]	30.73	[25.5-36]	202.98	[142.5-263.4]	4.44	[2.2-6.7]
54	20.98	[11.7-30.3]	2.92	[2-3.8]	601.21	[471.8-730.6]	1.34	[0.6-2]
55	1.8	[0-3.6]	1.26	[0.3-2.2]	45.35	[26.7-64]	1.62	[0.7-2.5]
56	2.64	[0.9-4.3]	23.06	[15.4-30.7]	205.68	[162.8-248.5]	5.95	[4-7.9]
57	67.53	[52.2-82.9]	22.63	[18.9-26.4]	2488.1	[2005-2971.2]	13.4	[10-16.8]
58	1.99	[0.02-3.9]	14.59	[10.6-18.6]	75.11	[55.6-94.6]	2.91	[1.9-3.9]
59	3.43	[0.3-6.5]	13.99	[6.9-21.1]	150.1	[102.1-198]	3.93	[2.4-5.4]
60	0.33	[0-1]	23.19	[4.5-41.9]	72.8	[51.9-93.6]	4.09	[1.8-6.4]
61	1.52	[0-3.2]	1.06	[0.6-1.5]	11.48	[6.2-16.8]	0.49	[0.15-0.8]
62	16.56	[9.3-23.8]	1.24	[0.7-1.8]	397.62	[296.3-498.9]	1	[0.5-1.5]
63	8.37	[4.7-12]	11.87	[8.6-15.1]	206.36	[155.9-256.8]	8.73	[5.9-11.5]
64	196.04	[144-248]	22.43	[10.1-34.8]	7669.65	[5833.5-9505.8]	9.84	[7-12.7]
65	9.95	[3.7-16.2]	3.95	[0.2-7.7]	1033.58	[813.3-1253.8]	2.4	[1.3-3.4]
66	2.06	[0-5]	39.16	[24.4-54]	437.46	[341.3-533.6]	8.95	[5.7-12.1]
67	4.19	[1.7-6.7]	1.25	[0.9-1.6]	78.74	[58.7-98.8]	3.51	[2.3-4.7]
68	1.63	[0-3.5]	0.19	[0.05-0.3]	22.66	[7.1-38.2]	0.244	[0.05-0.4]
69	4.8	[1.4-8.2]	17.79	[10.3-25.3]	535.34	[405.2-665.4]	3.24	[1.9-4.6]
70	4.83	[0.4-9.3]	6.61	[2.7-10.5]	218.76	[146.8-290.7]	2.18	[1.2-3.2]
71	10.53	[5.9-15.2]	27.73	[16.7-38.8]	687.69	[568.4-806.9]	11.5	[10.2-12.8]
72	1.999	[0.7-3.3]	7.48	[3.8-11.2]	32.04	[19.6-44.4]	2.13	[1-3.2]
73	7.07	[2.9-11.2]	3.07	[1.6-4.5]	151.74	[91.8-211.6]	1.42	[0.65-2.2]

Cmax Difference in the Robust System w/o Learning

The *Robust System w/o Learning* regression statistics are reported in Table 103, ANOVA test in Table 104, and Effect test in Table 105. The results indicate the success of the regression in predicting the values of *Cmax Difference* and that the model is significant since the p-value is very small.

The factors that were determined to be significant due to a relatively large t-Stat and a small p-value are bolded in Table 105; these factors are *Number of Machines,* and the interaction between *Repair Duration* and *Breakdown* and *Idle Time* and *Breakdown.*

Factor C (*Number of Machines*) has a negative effect on *Cmax Difference*, i.e. when the number of machines increases, *Cmax Difference* decreases. This is logical because the jobs' load will be distributed over the machines. Interaction DF (*Repair Duration* and *Breakdown*) has a positive effect on *Cmax Difference.* This makes sense too because if the repair durations and breakdown rate are higher, the delays will be more frequent and longer; i.e. $Cmax_R$ will increase.

Factors E and F interact because a larger idle time is able to absorb a higher rate of breakdowns, and vice versa.

Table 103. *Cmax Difference* Regression Results for *Robust System w/o Learning*

Regression Statistics	
R Square	0.777
Adjusted R Square	0.643
Standard Error	48.3152
Observations	73

Table 104. *Cmax Difference* ANOVA Test for *Robust System w/o Learning*

ANOVA

	df	*SS*	*MS*	*F*	*Significance F (p-value)*
Regression	27	365150	13524	5.79	0.000
Residual	45	105046	2334		
Total	72	470196			

Table 105. *Cmax Difference* Effect Test for *Robust System w/o Learning*

Predictor	Coefficients	SE Coef	t Stat	P-value
Constant	19.45	23.23	0.84	0.407
A	11.856	7.506	1.58	0.121
B	11.45	16.26	0.7	0.485
C	-61.18	17.3	-3.54	**0.001**
D	-16.99	17.48	-0.97	0.336
E	-1.36	16.87	-0.08	0.936
F	-22.45	17.95	-1.25	0.217
AB	0.1971	0.4014	0.49	0.626
AC	0.2386	0.4128	0.58	0.566
AD	0.36	0.4331	0.83	0.41
AE	-0.2779	0.392	-0.71	0.482
AF	-0.2658	0.4583	-0.58	0.565
BC	-19.047	9.908	-1.92	0.061
BD	1.86	10.92	0.17	0.865
BE	-3.923	9.396	-0.42	0.678
BF	-2.7	10	-0.27	0.789
CD	-3.425	9.883	-0.35	0.731
CE	-10.238	9.443	-1.08	0.284
CF	-6.78	10.32	-0.66	0.515
DE	17.59	10.35	1.7	0.096
DF	46.14	11.29	4.09	**0**
EF	21.813	9.331	2.34	**0.024**
AA	-6.67	13	-0.51	0.611
BB	-6.81	13.08	-0.52	0.605
CC	38.33	13.06	2.94	**0.005**
DD	-22.93	12.68	-1.81	0.077
EE	6.48	13.37	0.48	0.63
FF	13.06	14.1	0.93	0.359

Cmax Difference in the Robust System with Learning

The *Robust System with Learning* regression statistics are reported in Table 106, ANOVA test in Table 107, and Effect test in Table 108. The results indicate the success of the regression in predicting the values of *Cmax Difference* and that the model is significant since the p-value is very small.

The factors that were determined to be significant due to a relatively large t-Stat and a small p-value are bolded in Table 108; these factors are *Processing Time Range,* and the interactions between *Repair Duration* and *Breakdown*, *Number of Jobs* and *Number of Machines*, and *Processing Time Range* and *Breakdown.*

Factor A (*Processing Time Range*) has a positive effect on *Cmax Difference*, i.e. when the processing time increases, *Cmax Difference* increases. This is attributed to the fact that a wider processing time range will create a larger variability, i.e. it is harder for the learning parameter to predict $Cmax_R$. Interaction DF (*Repair Duration* and *Breakdown*) was discussed earlier. Interaction BC (*Number of Jobs* and *Number of Machines*) is evident as both the number of jobs and number of machines determine the size of the problem, i.e. the difficulty to attain solutions. Factors A and F interact because the time between breakdowns is a function of the processing time; the larger the processing time, the longer is the time between breakdowns.

Table 106. *Cmax Difference* Regression Results for *Robust System with Learning*

Regression Statistics	
R Square	0.77
Adjusted R Square	0.631
Standard Error	21.2020
Observations	73

Table 107. *Cmax Difference* ANOVA Test for *Robust System with Learning*

ANOVA	*df*	*SS*	*MS*	*F*	*Significance F (p-value)*
Regression	27	67538.5	2501.4	5.56	0.000
Residual	45	20228.6	449.5		
Total	72	87767.1			

Table 108. *Cmax Difference* Effect Test for *Robust System with Learning*

Predictor	Coefficients	SE Coef	t Stat	P-value
Constant	8.21	10.19	0.81	0.425
A	7.107	3.294	2.16	**0.036**
B	2.727	7.137	0.38	0.704
C	-11.262	7.592	-1.48	0.145
D	3.679	7.672	0.48	0.634
E	2.626	7.401	0.35	0.724
F	1.758	7.876	0.22	0.824
AB	0.1546	0.1762	0.88	0.385
AC	-0.2513	0.1812	-1.39	0.172
AD	-0.0265	0.1901	-0.14	0.89
AE	-0.2	0.172	-1.16	0.251
AF	-0.4156	0.2011	-2.07	**0.045**
BC	-11.317	4.348	-2.6	**0.012**
BD	1.885	4.792	0.39	0.696
BE	-3.756	4.123	-0.91	0.367
BF	-3.813	4.389	-0.87	0.39
CD	-2.667	4.337	-0.61	0.542
CE	-1.409	4.144	-0.34	0.735
CF	-1.064	4.528	-0.23	0.815
DE	3.472	4.541	0.76	0.448
DF	22.255	4.955	4.49	**0**
EF	1.691	4.095	0.41	0.682
AA	1.497	5.706	0.26	0.794
BB	-6.263	5.741	-1.09	0.281
CC	15.231	5.731	2.66	**0.011**
DD	-10.753	5.563	-1.93	0.06
EE	3.935	5.867	0.67	0.506
FF	6.079	6.186	0.98	0.331

CPU Statistical Analysis

In this section, the significance of the factors and their interactions is determined for each of the two systems in the case of the *CPU* performance measure. This analysis will follow the same approach used earlier.

CPU Time in the Robust System w/o Learning

The *Robust System w/o Learning* regression statistics are reported in Table 109, ANOVA test in Table 110, and Effect test in Table 111. The results indicate the success of the regression in predicting the values of *CPU Time* and that the model is significant since the p-value is very small.

The factors that were determined to be significant due to a relatively large t-Stat and a small p-value are bolded in Table 111. These factor effects on *CPU Time* in the case of *Robust System w/o Learning* are described in Table 112.

Table 109. *CPU Time* Regression Results for *Robust System w/o Learning*

Regression Statistics	
R Square	0.761
Adjusted R Square	0.618
Standard Error	15.1293
Observations	73

Table 110. *CPU Time* ANOVA Test for *Robust System w/o Learning*

ANOVA

	df	*SS*	*MS*	*F*	*Significance F (p-value)*
Regression	27	32792.6	1214.5	5.31	0.000
Residual	45	10300.3	228.9		
Total	72	43092.9			

Table 111. *CPU Time* Effect Test for *Robust System w/o Learning*

Predictor	Coefficients	SE Coef	t Stat	P-value
Constant	-0.695	7.07	-0.1	0.922
A	-0.163	2.288	-0.07	0.944
B	16.426	2.302	7.14	**0**
C	-6.55	2.348	-2.79	**0.008**
D	-2.88	2.324	-1.24	0.222
E	-7.961	2.263	-3.52	**0.001**
F	-8.272	2.362	-3.5	**0.001**
AB	-0.216	2.85	-0.08	0.94
AC	-0.206	2.948	-0.07	0.945
AD	-0.493	3.1	-0.16	0.874
AE	-5.834	2.937	-1.99	0.053
AF	0.767	3.007	0.25	0.8
BC	-1.105	3.031	-0.36	0.717
BD	-4.166	2.982	-1.4	0.169
BE	-1.552	2.982	-0.52	0.605
BF	-4.103	3.068	-1.34	0.188
CD	0.975	3.095	0.32	0.754
CE	2.868	3.059	0.94	0.353
CF	9.774	3.189	3.06	**0.004**
DE	0.814	2.977	0.27	0.786
DF	6.299	3.264	1.93	0.06
EF	4.672	2.988	1.56	0.125
AA	4.669	4.068	1.15	0.257
BB	4.268	3.913	1.09	0.281
CC	18.858	3.95	4.77	**0**
DD	-1.597	3.943	-0.4	0.687
EE	0.02	4.229	0	0.996
FF	2.5	4.002	0.62	0.535

Table 112. Factors' Effects on *CPU Time* in the case of *Robust System w/o Learning*

CPU Time Effects' Diagnostic for Robust System w/o Learning		
Factor/ Interaction	**Effect**	**Cause of Effect**
B	**+**	A higher number of jobs leads to a higher possibilities of assignments to the machines; the MIP will require more time to attain a solution.
C	**-**	When there are more machines, the jobs on each machine will be less, i.e. the problem becomes a little easier for the MIP to solve.
E	**-**	A larger repair duration leads to longer but fewer delays as no more than one breakdown can occur until the repair finishes, i.e. less rescheduling
F	**-**	When the time between breakdowns is larger, less delay will occur, hence, less shifting is required.
CF	**+**	C (number of jobs) and F (breakdown) interact because more machines lead to fewer breakdowns on each machine as no more than one breakdown can occur at a time over the machines.

CPU Time in the Robust System with Learning

The *Robust System with Learning* regression statistics are reported in Table 113, ANOVA test in Table 114, and Effect test in Table 115. The results indicate the success of the regression in predicting the values of *CPU Time* and that the model is significant since the p-value is very small.

Table 113. *CPU Time* Regression Results for *Robust System with Learning*

Regression Statistics	
R Square	0.658
Adjusted R Square	0.453
Standard Error	6.82670
Observations	73

Table 114. *CPU Time* ANOVA Test for *Robust System with Learning*

ANOVA					
	df	*SS*	*MS*	*F*	*Significance F (p-value)*
Regression	27	4033.32	149.38	3.21	0.000
Residual	45	2097.17	46.60		
Total	72	6130.49			

Table 115. *CPU Time* Effect Test for *Robust System with Learning*

Predictor	Coefficients	SE Coef	t Stat	P-value
Constant	-1.758	3.19	-0.55	0.584
A	0.117	1.032	0.11	0.91
B	6.399	1.039	6.16	**0**
C	0.274	1.059	0.26	0.797
D	0.289	1.049	0.28	0.784
E	0.675	1.021	0.66	0.512
F	-2.032	1.066	-1.91	0.063
AB	1.48	1.286	1.15	0.256
AC	1.882	1.33	1.41	0.164
AD	2.117	1.399	1.51	0.137
AE	-0.405	1.325	-0.31	0.761
AF	0.685	1.357	0.5	0.616
BC	3.124	1.368	2.28	**0.027**
BD	0.009	1.345	0.01	0.995
BE	-0.601	1.345	-0.45	0.657
BF	-0.274	1.384	-0.2	0.844
CD	-0.371	1.397	-0.27	0.792
CE	-1.301	1.38	-0.94	0.351
CF	-0.896	1.439	-0.62	0.537
DE	-1.665	1.343	-1.24	0.221
DF	0.359	1.473	0.24	0.808
EF	0.468	1.348	0.35	0.73
AA	1.823	1.836	0.99	0.326
BB	1.708	1.765	0.97	0.338
CC	6.404	1.782	3.59	**0.001**
DD	-2.084	1.779	-1.17	0.248
EE	2.259	1.908	1.18	0.243
FF	5.075	1.806	2.81	**0.007**

The factors that were determined to be significant due to a relatively large t-Stat and a small p-value are bolded in Table 115; these factors are *Number of Jobs,* and the interaction between *Number of Jobs* and *Number of Machines*.

Interaction BC is logical as both the number of jobs and number of machines determine the size of the problem, i.e. the difficulty to attain solutions.

Shifted Jobs Statistical Analysis

In this section, the significance of the factors and their interactions is determined for each of the two systems in the case of the *Shifted Jobs* performance measure.

Shifted Jobs in the Robust System w/o Learning

The *Robust System w/o Learning* regression statistics are reported in Table 116, ANOVA test in Table 117, Effect test in Table 118, and the factor effects diagnosis in Table 119. The results indicate the success of the regression in predicting the values of *Shifted Jobs* and that the model is significant since the p-value is very small.

The factors that were determined to be significant due to a relatively large t-Stat and a small p-value are bolded in Table 118 and explained in Table 119; these factors are *Number of Jobs*, *Number of Machines, Repair Duration, Idle Time* and *Breakdown*, and the interactions between *Repair Duration* and *Breakdown*, *Idle Time* and *Breakdown*, *Number of Machines* and *Breakdown*, and *Number of Jobs* and *Breakdown*.

Table 116. *Shifted Jobs* Regression Results for *Robust System w/o Learning*

Regression Statistics	
R Square	0.858
Adjusted R Square	0.773
Standard Error	3.28318
Observations	73

Table 117. *Shifted Jobs* ANOVA Test for *Robust System w/o Learning*

ANOVA

	df	*SS*	*MS*	*F*	*Significance F (p-value)*
Regression	27	2933.01	108.63	10.08	0.000
Residual	45	485.07	10.78		
Total	72	3418.07			

Table 118. *Shifted Jobs* Effect Test for *Robust System w/o Learning*

Predictor	Coefficients	SE Coef	t Stat	P-value
Constant	3.891	1.534	2.54	0.015
A	-0.0502	0.4964	-0.1	0.92
B	3.4753	0.4995	6.96	**0**
C	-2.1223	0.5095	-4.17	**0**
D	-1.1446	0.5043	-2.27	**0.028**
E	-1.6311	0.4911	-3.32	**0.002**
F	-4.5305	0.5126	-8.84	**0**
AB	-0.0535	0.6185	-0.09	0.931
AC	-0.0844	0.6398	-0.13	0.896
AD	-0.008	0.6727	-0.01	0.991
AE	-0.9125	0.6373	-1.43	0.159
AF	0.4475	0.6526	0.69	0.496
BC	-1.1854	0.6577	-1.8	0.078
BD	-1.244	0.647	-1.92	0.061
BE	-0.2874	0.647	-0.44	0.659
BF	-1.675	0.6657	-2.52	**0.015**
CD	0.4626	0.6717	0.69	0.495
CE	0.3526	0.6638	0.53	0.598
CF	2.4411	0.6921	3.53	**0.001**
DE	0.5797	0.646	0.9	0.374
DF	2.0618	0.7083	2.91	**0.006**
EF	1.3537	0.6484	2.09	**0.042**
AA	-0.2858	0.8828	-0.32	0.748
BB	-1.3525	0.8491	-1.59	0.118
CC	2.8488	0.8571	3.32	**0.002**
DD	-0.285	0.8558	-0.33	0.741
EE	-0.1199	0.9176	-0.13	0.897
FF	2.9012	0.8684	3.34	**0.002**

Table 119. Factors' Effects on *Shifted Jobs* in the case of *Robust System w/o Learning*

Shifted Jobs Effects' Diagnostic for Robust System w/o Learning		
Factor/ Interaction	**Effect**	**Cause of Effect**
B	**+**	A higher number of jobs logically indicates a higher number of shifts between the machines
C	**-**	When there are more machines, the jobs on each machine will be less, i.e. fewer jobs will be shifted.
D	**-**	A larger repair duration leads to longer but fewer delays as no more than one breakdown can occur until the repair finishes
E	**-**	The higher the idle time the easier it will be to fix the schedule by just shifting the jobs on the same machine; i.e. less jobs will be shifted to another machine
F	**-**	When the time between breakdowns is larger, less delay will occur, hence, less shifting is required.
BF	**-**	BF effect is negative because F effect is stronger than B. B and F interact because the higher the number of jobs, the more they will be hit by a breakdown.
CF	**+**	C and F interact because more machines lead to fewer breakdowns on each machine as no more than one breakdown can occur at a time over the machines.
DF	**+**	D and F interact because if the repair durations and breakdown rate are higher, the delays will be more frequent and longer; i.e. $Cmax_R$ will increase.
EF	**+**	E and F interact because a higher idle time will absorb more frequent breakdowns, and vice versa

Shifted Jobs in the Robust System with Learning

The *Robust System with Learning* regression statistics are reported in Table 120, ANOVA test in Table 121, and Effect test in Table 122. The results indicate the success of the regression in predicting the values of *Shifted Jobs* and that the model is significant since the p-value is very small.

The factors that were determined to be significant due to a relatively large t-Stat and a small p-value are bolded in Table 122; these factors are *Number of Jobs* and *Breakdown*, and the interactions between *Repair Duration* and *Processing Time Range*, and *Number of Machines* and *Idle Time*. The effects of B and F are explained in Table 119.

Processing Time Range and *Repair Duration* interact because the repair duration depends on the processing time. If the latter increases, the repair time will increase too.

Number of Machines and *Idle Time* interact because the larger the number of machines, the smaller the number of jobs on each machine, hence, the smaller the idle time on each machine.

Table 120. *Shifted Jobs* Regression Results for *Robust System with Learning*

Regression Statistics	
R Square	0.808
Adjusted R Square	0.693
Standard Error	1.82045
Observations	73

Table 121. *Shifted Jobs* ANOVA Test for *Robust System with Learning*

ANOVA

	df	*SS*	*MS*	*F*	*Significance F (p-value)*
Regression	27	628.148	23.265	7.02	0.000
Residual	45	149.131	3.314		
Total	72	777.279			

Table 122. *Shifted Jobs* Effect Test for *Robust System with Learning*

Predictor	Coefficients	SE Coef	t Stat	P-value
Constant	2.1224	0.8507	2.49	0.016
A	-0.1905	0.2753	-0.69	0.492
B	2.036	0.277	7.35	**0**
C	-0.2405	0.2825	-0.85	0.399
D	-0.0989	0.2796	-0.35	0.725
E	-0.1247	0.2723	-0.46	0.649
F	-2.2	0.2842	-7.74	**0**
AB	-0.1548	0.3429	-0.45	0.654
AC	0.4019	0.3548	1.13	0.263
AD	0.902	0.373	2.42	**0.02**
AE	-0.1265	0.3534	-0.36	0.722
AF	0.4664	0.3619	1.29	0.204
BC	0.1875	0.3647	0.51	0.61
BD	-0.5196	0.3588	-1.45	0.154
BE	-0.042	0.3588	-0.12	0.907
BF	-0.6985	0.3691	-1.89	0.065
CD	-0.4441	0.3724	-1.19	0.239
CE	-0.8139	0.368	-2.21	**0.032**
CF	0.2897	0.3837	0.75	0.454
DE	-0.5489	0.3582	-1.53	0.132
DF	0.3717	0.3927	0.95	0.349
EF	-0.1357	0.3595	-0.38	0.708
AA	0.6357	0.4895	1.3	0.201
BB	-0.6806	0.4708	-1.45	0.155
CC	0.6272	0.4752	1.32	0.194
DD	0.1195	0.4745	0.25	0.802
EE	-0.1705	0.5088	-0.34	0.739
FF	2.2159	0.4815	4.6	**0**

Match-up Statistical Analysis

In this section, the significance of the factors and their interactions is determined for each of the two systems in the case of the *Match-up Time* performance measure. This analysis will follow the same approach used earlier.

Match-up in the Robust System w/o Learning

The *Robust System w/o Learning* regression statistics are reported in Table 123, ANOVA test in Table 124, and Effect test in Table 125. The results indicate the success of the regression in predicting the values of *Match-up* and that the model is significant since the p-value is very small.

The factors that were determined to be significant due to a relatively large t-Stat and a small p-value are bolded in Table 125 and explained in Table 126; these factors are *Processing Time Range*, *Number of Jobs*, *Number of Machines, Idle Time,* and *Breakdown*, and the interactions between *Number of Machines* and *Processing Time Range*, *Number of Jobs* and *Number of Machines*, and *Number of Machines* and *Breakdown.*

Table 123. *Match-up* Regression Results for *Robust System w/o Learning*

Regression Statistics	
R Square	0.802
Adjusted R Square	0.683
Standard Error	1811.28
Observations	73

Table 124. *Match-up* ANOVA Test for *Robust System w/o Learning*

ANOVA

	df	*SS*	*MS*	*F*	*Significance F (p-value)*
Regression	27	597666764	22135806	6.75	0.000
Residual	45	147632323	3280718		
Total	72	745299087			

Table 125. *Match-up* Effect Test for *Robust System w/o Learning*

Predictor	Coefficients	SE Coef	t Stat	P-value
Constant	829.8	846.4	0.98	0.332
A	602.9	273.9	2.2	**0.033**
B	1133.9	275.6	4.11	**0**
C	-1957	281.1	-6.96	**0**
D	-25.6	278.2	-0.09	0.927
E	-610.3	270.9	-2.25	**0.029**
F	-1093.2	282.8	-3.87	**0**
AB	503.9	341.2	1.48	0.147
AC	-853.3	353	-2.42	**0.02**
AD	-106.7	371.1	-0.29	0.775
AE	-251	351.6	-0.71	0.479
AF	-292.8	360	-0.81	0.42
BC	-1579.9	362.9	-4.35	**0**
BD	-66.8	356.9	-0.19	0.852
BE	-339.7	357	-0.95	0.346
BF	-656.5	367.3	-1.79	0.081
CD	463.4	370.6	1.25	0.218
CE	630.3	366.2	1.72	0.092
CF	1739.4	381.8	4.56	**0**
DE	310.7	356.4	0.87	0.388
DF	296.1	390.7	0.76	0.453
EF	579.7	357.7	1.62	0.112
AA	-507.4	487	-1.04	0.303
BB	-20.2	468.4	-0.04	0.966
CC	1726.1	472.8	3.65	**0.001**
DD	-1042.6	472.1	-2.21	**0.032**
EE	386.3	506.2	0.76	0.449
FF	410	479.1	0.86	0.397

Table 126. Factors' Effects on *Match-up Time* in the case of *Robust System w/o Learning*

Match-up Effects' Diagnostic for Robust System w/o Learning		
Factor/ Interaction	**Effect**	**Cause of Effect**
A	**+**	It is logical that the larger the processing time, the larger the match-up time will be.
B	**+**	When the number of jobs increases, more jobs will be shifted, i.e. longer time to match.
C	**-**	When there are more machines, the jobs on each machine will be less, i.e. time to match will be less.
E	**-**	It is easier to match-up with the initial schedule when the idle time is larger as it will absorb the shifting of the jobs better.
F	**-**	When the time between breakdowns is larger, less delay will occur, hence, it is easier to match-up.
AC	**-**	A (processing) and C (number of machines) interact in the case of Match-up time because for example a larger processing time with a small number of machines will increase the match-up dramatically; on the other hand, a smaller processing time with a large number of machines will decrease the match-up time.
BC	**-**	BC effect is negative because C effect is stronger than B (number of jobs). It is obvious that B and C interact as the number of jobs on each machine depends on both of them.
CF	**+**	C and F (breakdown) interact because more machines lead to fewer breakdowns on each machine as no more than one breakdown can occur at a time over the machines.

Match-up in the Robust System with Learning

The *Robust System with Learning* regression statistics are reported in Table 127, ANOVA test in Table 128, and Effect test in Table 129. The results indicate the success of the regression in predicting the values of *Match-up* and that the model is significant since the p-value is very small.

The factors that were determined to be significant due to a relatively large t-Stat and a small p-value are bolded in Table 129; these factors are *Processing Time Range*, *Number of Jobs*, *Number of Machines,* and *Breakdown*, and the interactions between *Number of Jobs* and

Number of Machines, and *Number of Machines* and *Breakdown.* The factor effects are explained in Table 126.

Processing Time Range and *Number of Jobs* interact in the case of *Match-up Time* because for example a large processing time with a high number of jobs will lead to a large *Match-up Time*, and vice versa.

Table 127. *Match-up* Regression Results for *Robust System with Learning*

Regression Statistics	
R Square	0.816
Adjusted R Square	0.706
Standard Error	844.630
Observations	73

Table 128. *Match-up* ANOVA Test for *Robust System with Learning*

ANOVA

	df	*SS*	*MS*	*F*	*Significance F (p-value)*
Regression	27	142671841	5284142	7.41	0.000
Residual	45	32102993	713400		
Total	72	174774834			

Table 129. *Match-up* Effect Test for *Robust System w/o Learning*

Predictor	Coefficients	SE Coef	t Stat	P-value
Constant	249.3	394.7	0.63	0.531
A	403.3	127.7	3.16	**0.003**
B	615.4	128.5	4.79	**0**
C	-1029.3	131.1	-7.85	**0**
D	160.1	129.7	1.23	0.224
E	-183.2	126.3	-1.45	0.154
F	-492.8	131.9	-3.74	**0.001**
AB	329.8	159.1	2.07	**0.044**
AC	-458.9	164.6	-2.79	**0.008**
AD	140.4	173.1	0.81	0.421
AE	-136.8	164	-0.83	0.408
AF	-150.3	167.9	-0.9	0.375
BC	-710	169.2	-4.2	**0**
BD	-28.1	166.5	-0.17	0.867
BE	-96.7	166.5	-0.58	0.564
BF	-287.6	171.3	-1.68	0.1
CD	-91.9	172.8	-0.53	0.597
CE	76.7	170.8	0.45	0.655
CF	712.3	178	4	**0**
DE	-8.2	166.2	-0.05	0.961
DF	13	182.2	0.07	0.943
EF	48.8	166.8	0.29	0.771
AA	11.5	227.1	0.05	0.96
BB	-152.9	218.4	-0.7	0.488
CC	861.6	220.5	3.91	**0**
DD	-322	220.2	-1.46	0.151
EE	230.8	236.1	0.98	0.334
FF	304.1	223.4	1.36	0.18

Repair and Rescheduling Rule Comparisons

Following the analysis of factors and interaction significance, this section will compare the systems to the rules described in chapter 6 based on each performance measure as well as an overall performance. The Eigenvalue Normalization Procedure explained in chapter 6 (Equation 7) will also be used here to attain a unique measure for the four performance measures. Conclusions are drawn regarding dominance among the rules.

Cmax Difference Comparison

Following the normalization of the performance measures, the *Cmax Difference* performance of the four rules and the two systems is presented in Table 130. The boxplot of the rules is also shown in Figure 26. It is visually noticeable that *Robust with Learning* performed the best, followed by *Robust w/o Learning*, *CR*, and *PR*, then *FJR*, and finally *RSR* that had the worst performance; however, this can not be validated unless tests are undertaken to determine that the differences are statistically significant. It is obvious though that no tests are needed for RSR and *Robust with Learning* as the boxplot indicates clearly that their performances are significantly far from the rest.

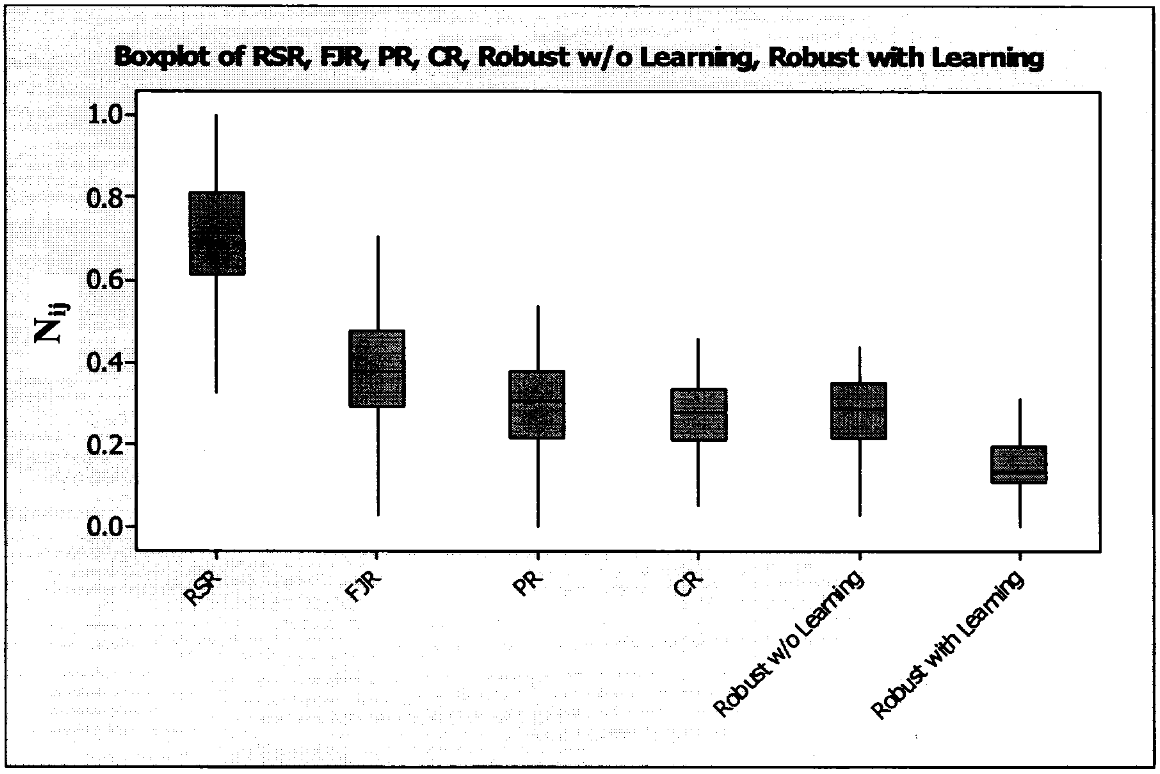

Figure 26. *Cmax Difference* Boxplot for the Rules and Systems

Table 130. *Cmax Difference* Performance among the rules and systems

	Cmax Difference					
Run	RSR	FJR	PR	CR	ROBUST w/o Learning	Robust with Learning
1	0.650911	0.368985	0.366392	0.316008	0.348239	0.291187
2	0.725332	0.35202	0.357177	0.336798	0.329012	0.026799
3	0.702077	0.478835	0.223957	0.340851	0.244333	0.227532
4	0.518137	0.4341	0.429091	0.429091	0.372324	0.190336
5	0.584696	0.426692	0.295155	0.263875	0.206127	0.526146
6	0.401019	0.340548	0.442394	0.442394	0.440802	0.370783
7	0.540974	0.263551	0.476662	0.430005	0.432527	0.196718
8	0.860452	0.286211	0.387681	0.105566	0.103746	0.074169
9	0.554789	0.502922	0.310274	0.39641	0.39641	0.169493
10	0.577695	0.36763	0.421639	0.440025	0.393204	0.071449
11	0.699039	0.582386	0.216641	0.261372	0.234183	0.045609
12	0.556085	0.534664	0.293037	0.293037	0.412994	0.250195
13	0.73604	0.313401	0.419926	0.276383	0.293778	0.144894
14	0.673579	0.356712	0.412686	0.250457	0.391814	0.180253
15	0.723169	0.396576	0.329925	0.331036	0.294377	0.121084
16	0.622262	0.704856	0.173037	0.239188	0.135664	0.102029
17	0.881628	0.221633	0.25053	0.25053	0.219052	0.009804
18	0.701506	0.264019	0.442373	0.225691	0.404656	0.166749
19	0.517531	0.421502	0.497797	0.382034	0.35827	0.179969
20	0.771481	0.326696	0.344877	0.269004	0.273787	0.178395
21	0.70447	0.506037	0.292833	0.240817	0.253018	0.199717
22	0.767491	0.26564	0.427752	0.259688	0.242018	0.177231
23	0.653916	0.407114	0.34727	0.322808	0.333292	0.266022
24	0.848799	0.349725	0.20953	0.211394	0.257505	0.048314
25	0.725576	0.364189	0.302557	0.378196	0.260535	0.196102
26	0.710361	0.492264	0.211136	0.291955	0.30665	0.17092
27	0.820512	0.318406	0.283379	0.294952	0.212554	0.113568
28	0.822603	0.342308	0.218575	0.23264	0.295552	0.130005
29	0.713181	0.54565	0.190234	0.204325	0.209805	0.267736
30	0.911031	0.116492	0.385216	0.059848	0	0.066928
31	0.99646	0.056645	0.014777	0.049257	0.025121	0.024136
32	0.538943	0.447787	0.424932	0.375005	0.418586	0.112324
33	0.920355	0.197807	0.200619	0.214877	0.108644	0.124709
34	0.41899	0.575493	0.336777	0.387294	0.241687	0.414038
35	0.999595	0.028458	0	0	0	0
36	0.327591	0.326306	0.295474	0.295474	0.378978	0.684087
37	0.998672	0.042134	0	0	0.028553	0.008009
38	0.60255	0.397249	0.378018	0.393697	0.372873	0.205423
39	0.744925	0.505173	0.180835	0.198919	0.318504	0.127168
40	0.833701	0.282329	0.272365	0.272365	0.272365	0.051816
41	0.938459	0.198134	0.167075	0.167075	0.152884	0.028917
42	0.80412	0.481635	0.203648	0.193701	0.162813	0.126167
43	0.769369	0.413305	0.241582	0.264663	0.302208	0.132332
44	0.733137	0.235509	0.535632	0.213378	0.237421	0.135076
45	0.918996	0.14929	0.17487	0.199519	0.213471	0.131152
46	0.817168	0.321691	0.244658	0.307517	0.249588	0.109695
47	0.794408	0.301251	0.306208	0.289546	0.288501	0.13165
48	0.688221	0.474444	0.361505	0.271255	0.280834	0.134619
49	0.723321	0.329109	0.384735	0.328623	0.316584	0.110703
50	0.711238	0.545149	0.26052	0.198236	0.271905	0.125907
51	0.805261	0.41034	0.208169	0.160196	0.302401	0.150772
52	0.489988	0.450774	0.449529	0.456998	0.365997	0.108803
53	0.662054	0.595236	0.196882	0.168829	0.206573	0.312155
54	0.543066	0.456222	0.370859	0.402169	0.385245	0.221923
55	0.635023	0.481455	0.319587	0.320521	0.353828	0.186771
56	0.730286	0.491173	0.309465	0.277958	0.198541	0.113945
57	0.712742	0.347572	0.361981	0.322697	0.330169	0.164367
58	0.909823	0.231518	0.19767	0.148929	0.22204	0.089809
59	0.53994	0.599587	0.317947	0.379151	0.269192	0.177905
60	0.853986	0.356535	0.256366	0.118845	0.246179	0.056027
61	0.685769	0.400032	0.304106	0.3337	0.376561	0.155114
62	0.529625	0.448145	0.431947	0.313652	0.400369	0.270948
63	0.736327	0.380664	0.282003	0.283885	0.31964	0.225011
64	0.791806	0.269905	0.383252	0.248387	0.254765	0.163435
65	0.571173	0.407905	0.499458	0.334493	0.367273	0.105557
66	0.956958	0.20185	0.026146	0.194006	0.047586	0.053862
67	0.66534	0.403985	0.348606	0.351395	0.348208	0.166933
68	0.72867	0.273539	0.380042	0.380042	0.29959	0.124893
69	0.761384	0.538749	0.172685	0.175183	0.200514	0.171258
70	0.60435	0.584205	0.307669	0.31264	0.29171	0.126364
71	0.824831	0.395705	0.201066	0.254198	0.212849	0.112798
72	0.652225	0.385516	0.332174	0.398851	0.312777	0.242341
73	0.715817	0.33417	0.329787	0.343665	0.287057	0.258205

The first test is the One-Way ANOVA, which will determine if there is significant difference between the means of the 6 alternatives. The ANOVA results are shown in Table 131.

Table 131. One-Way ANOVA for *Cmax Difference*

Anova: Single Factor

SUMMARY

Groups	*Count*	*Sum*	*Average*	*Variance*
RSR	73	52.063	0.713192	0.020635
FJR	73	27.40544	0.375417	0.018262
PR	73	21.93136	0.30043	0.013141
CR	73	20.07717	0.27503	0.010126
ROBUST w/o Learning	73	19.89701	0.272562	0.010589
Robust with Learning	73	11.83726	0.162154	0.012348

ANOVA

Source of Variation	*SS*	*df*	*MS*	*F*	*P-value*	*F crit*
Between Groups	13.27976	5	2.655953	187.2541	9.7E-106	2.23488
Within Groups	6.127351	432	0.014184			
Total	19.40711	437				

As the p-value in Table 131 is less than 0.05, we can reject the hypothesis that all the means are equal, i.e. there is a significant difference between the performances of the rules.

It was previously determined in Chapter 6 that the difference between *PR* and *CR* is not statistically significant (Table 84) and that both rules outperformed *FJR* (Table 83), and the latter outperformed *RSR*. Following this, a t test is conducted for *Robust w/o Learning – CR* (Table 132). Even though the *Robust w/o Learning* mean is smaller than *CR* mean (indicating that *Robust w/o Learning* performed better), this difference is not statistically significant as the 95% Confidence Interval overlaps with zero. Moreover, the p-value is greater than 0.05.

Table 132. t test for *Robust w/o Learning – CR* in the case of *Cmax Difference*

Two-sample T for Robust w/o Learning vs CR

	N	Mean	StDev	SE
Robust w/o Learn	73	0.273	0.103	0.012
CR	73	0.275	0.101	0.012

Difference = mu (Robust w/o Learning) - mu (CR)
Estimate for difference: -0.002468
95% CI for difference: (-0.035766, 0.030830)
T-Test of difference = 0 (vs not =): T-Value = -0.15 P-Value = 0.884 DF = 143

The next t test is for *Robust w/o Learning – Robust with Learning* (Table 133). As p-value is less than 0.05, we conclude that *Robust with Learning* outperformed *Robust w/o Learning* and the difference is statistically significant.

Table 133. t test for *Robust w/o Learning – Robust with Learning* in the case of *Cmax Difference*

Two-sample T for Robust w/o Learning vs Robust with Learning

	N	Mean	StDev	SE Mean
Robust w/o Learn	73	0.273	0.103	0.012
Robust with Lear	73	0.162	0.111	0.013

Difference = mu (Robust w/o Learning) - mu (Robust with Learning)
Estimate for difference: 0.110408
95% CI for difference: (0.075369, 0.145446)
T-Test of difference = 0 (vs not =): T-Value = 6.23 P-Value = 0.000 DF = 143

Based on previous tests, we conclude that for the *Cmax Difference*, the best performance was achieved by *Robust with Learning*, then *Robust w/o Learning*, *CR* and *PR*, followed by *FJR*, then finally *RSR* that had the worst performance.

CPU Comparison

The *CPU* performance is presented in Table 134. The boxplot of the rules and systems is also shown in Figure 26. It is known from chapter 6 that *RSR* performed the best (Table 87), followed by *FJR*, then *PR* and *CR* (Table 88). The ANOVA results shown in Table 135 indicate that the means are not equal.

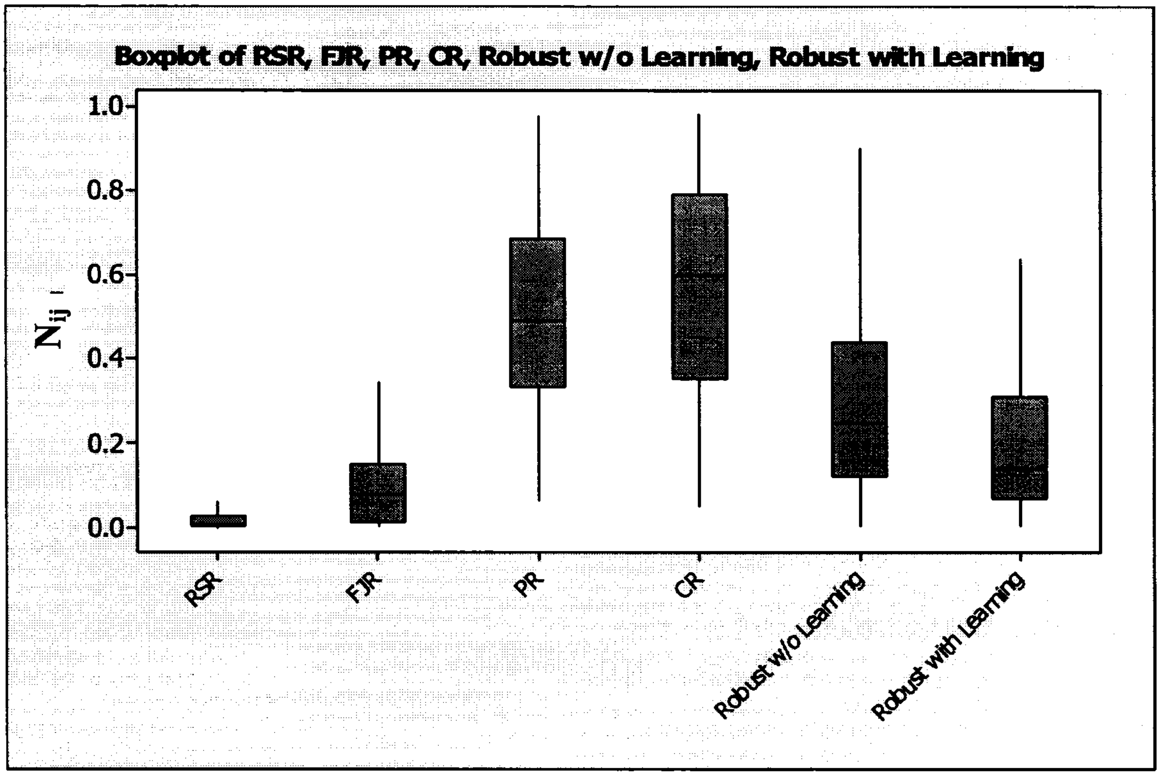

Figure 27. *CPU* Boxplot of the rules and systems

Table 134. *CPU* Performance among the rules and systems

Run	CPU Time RSR	FJR	PR	CR	Robust w/o Learning	Robust with Learning
1	0.01288	0.03221	0.663454	0.373595	0.334948	0.553952
2	0.00173	0.17887	0.697195	0.347068	0.601171	0.007688
3	0.00984	0.09531	0.588129	0.794112	0.118671	0.015064
4	0.43783	0.39139	0.670006	0.252082	0.325053	0.192378
5	0.03814	0.18346	0.733848	0.579245	0.179339	0.242211
6	0.05594	0.93234	0.063399	0.096964	0.063399	0.331914
7	0.01782	0.04411	0.688876	0.710495	0.065148	0.118903
8	0.01462	0.00595	0.66106	0.744987	0.076217	0.043995
9	0.01215	0.01215	0.164077	0.577308	0.783923	0.158
10	0.00094	0.0073	0.625893	0.193049	0.753328	0.058597
11	0.00697	0.00152	0.162805	0.977631	0.125504	0.04393
12	0.00369	0.7897	0.409611	0.110706	0.439132	0.059043
13	0.00107	0.09798	0.77531	0.599573	0.135813	0.106583
14	0.01231	0.01477	0.153885	0.437035	0.670941	0.578609
15	0.00146	0.843	0.115121	0.223684	0.106377	0.463398
16	0.00286	0.03379	0.209477	0.936148	0.25158	0.123711
17	0.01345	0.5072	0.308593	0.779394	0.110777	0.166165
18	0.05893	0.71582	0.125449	0.148884	0.193688	0.639307
19	0.14453	0.34487	0.628688	0.05247	0.398296	0.550937
20	0.00301	0.17285	0.384873	0.362654	0.802132	0.216915
21	0.00402	0.0813	0.263785	0.947936	0.058184	0.147726
22	0.00365	0.02729	0.868119	0.480299	0.09526	0.076486
23	0.00242	0.00519	0.482799	0.117672	0.33744	0.799474
24	0.15046	0.00692	0.375297	0.605318	0.658932	0.189378
25	0.60928	0.02101	0.357163	0.577764	0.388678	0.126058
26	0.04585	0.00663	0.67167	0.65289	0.138643	0.31816
27	0.02581	0.01251	0.539699	0.835073	0.0961	0.036335
28	0.00886	0.01772	0.751557	0.422289	0.441484	0.248058
29	0.01041	0.00911	0.458734	0.855002	0.194555	0.143151
30	0.0128	0.01359	0.266783	0.963544	0.005512	0.005549
31	0.00595	0.01106	0.307488	0.950285	0.035961	0.03098
32	0.00329	0.01644	0.813091	0.39298	0.422577	0.074814
33	0.00959	0.23877	0.304615	0.913215	0.02775	0.12399
34	0.01819	0.08353	0.642579	0.662427	0.22329	0.301855
35	0.00986	0.06371	0.469988	0.87896	0.043808	0.021579
36	0.00607	0.013	0.385806	0.112707	0.728263	0.554867
37	0.01253	0.04042	0.404537	0.911394	0.034023	0.052548
38	0.00131	0.09654	0.71174	0.429256	0.546598	0.032803
39	0.00164	0.09725	0.58418	0.764231	0.24182	0.08214
40	0.01382	0.06909	0.483645	0.400734	0.497463	0.594193
41	0.00178	0.10866	0.555744	0.508838	0.501119	0.411464
42	0.00277	0.11377	0.634039	0.507786	0.398182	0.410669
43	0.00133	0.11264	0.396485	0.864793	0.249128	0.142056
44	0.00131	0.12229	0.866192	0.41449	0.237454	0.081043
45	0.00639	0.01038	0.395153	0.455024	0.522879	0.602708
46	0.00058	0.14953	0.567544	0.78417	0.109036	0.169482
47	0.00223	0.26466	0.485166	0.768997	0.093867	0.307242
48	0.00538	0.11812	0.491823	0.721785	0.462358	0.096853
49	0.00119	0.07886	0.844254	0.440301	0.287967	0.064996
50	0.00467	0.01493	0.362094	0.924929	0.111872	0.025188
51	0.0022	0.0127	0.898347	0.320542	0.159294	0.254332
52	0.00029	0.09116	0.97394	0.069686	0.180255	0.076042
53	0.02576	0.1035	0.697089	0.661632	0.200173	0.157685
54	0.00135	0.13643	0.786845	0.210726	0.528166	0.197218
55	0.0883	0.27579	0.391911	0.826158	0.238292	0.15241
56	0.00206	0.03791	0.758571	0.612571	0.20537	0.075556
57	0.01141	0.14504	0.773216	0.449624	0.404049	0.124709
58	0.05058	0.00728	0.69068	0.232654	0.343114	0.590335
59	0.03172	0.07216	0.621366	0.668344	0.215032	0.338773
60	0.00075	0.00089	0.402283	0.903939	0.128776	0.066913
61	0.03146	0.13632	0.679505	0.635463	0.255863	0.222307
62	0.0179	0.15539	0.883044	0.253171	0.358805	0.054126
63	0.03044	0.08187	0.661876	0.656631	0.332812	0.111187
64	0.01106	0.04527	0.542132	0.780543	0.291244	0.099249
65	0.01758	0.07619	0.117743	0.923474	0.355463	0.029417
66	0.01144	0.0136	0.401298	0.907166	0.104392	0.069253
67	0.14737	0.53354	0.235256	0.652503	0.447431	0.11097
68	0.35569	0.62576	0.289825	0.599411	0.1515	0.125152
69	0.006	0.15834	0.506414	0.460569	0.335728	0.627375
70	0.0339	0.00458	0.197421	0.240936	0.900074	0.302773
71	0.00667	0.01523	0.403365	0.905525	0.123764	0.041559
72	0.01581	0.16227	0.247096	0.285556	0.821896	0.394089
73	0.03688	0.112	0.222634	0.133853	0.861851	0.419316

Table 135. One-Way ANOVA for *CPU Time*

Anova: Single Factor

SUMMARY

Groups	Count	Sum	Average	Variance
RSR	73	2.770214	0.037948	0.009458
FJR	73	10.43274	0.142914	0.042783
PR	73	36.95338	0.506211	0.051319
CR	73	40.95285	0.560998	0.074862
Robust w/o Learning	73	22.66898	0.310534	0.05257
Robust with Learning	73	15.58389	0.213478	0.037854

ANOVA

Source of Variation	SS	df	MS	F	P-value	F crit
Between Groups	15.43636	5	3.087272	68.90051	7.08E-53	2.23488
Within Groups	19.35692	432	0.044808			
Total	34.79328	437				

The t test for *PR - Robust w/o Learning* shown in Table 136 indicates that *Robust w/o Learning* outperformed *PR* and the difference is statistically significant. Furthermore, a t test for *Robust with Learning - Robust w/o Learning* shown in Table 137 indicates that *Robust with Learning* outperformed *Robust w/o Learning.*

Table 136. t test for *PR - Robust w/o Learning* in the case of *CPU Time*

Two-sample T for PR vs Robust w/o Learning

	N	Mean	StDev	SE Mean
PR	73	0.506	0.227	0.027
Robust w/o Learn	73	0.311	0.229	0.027

Difference = mu (PR) - mu (Robust w/o Learning)
Estimate for difference: 0.195677
95% CI for difference: (0.121107, 0.270246)
T-Test of difference = 0 (vs not =): T-Value = 5.19 P-Value = 0.000 DF = 143

Table 137. t test for *Robust with Learning - Robust w/o Learning* in the case of *CPU*

Two-sample T for Robust with Learning vs Robust w/o Learning

	N	Mean	StDev	SE Mean
Robust with Lear	73	0.213	0.195	0.023
Robust w/o Learn	73	0.311	0.229	0.027

Difference = mu (Robust with Learning) - mu (Robust w/o Learning)
Estimate for difference: -0.097056
95% CI for difference: (-0.166638, -0.027474)
T-Test of difference = 0 (vs not =): T-Value = -2.76 P-Value = 0.007 DF = 140

Finally, a t test was carried out for *Robust with Learning – FJR* in Table 138, which proved that *FJR* outperformed *Robust with Learning* and the difference is statistically significant.

Table 138. t test for *Robust with Learning – FJR* in the case of *CPU Time*

Two-sample T for Robust with Learning vs FJR

	N	Mean	StDev	SE Mean
Robust with Lear	73	0.213	0.195	0.023
FJR	73	0.143	0.207	0.024

Difference = mu (Robust with Learning) - mu (FJR)
Estimate for difference: 0.070564
95% CI for difference: (0.004867, 0.136261)
T-Test of difference = 0 (vs not =): T-Value = 2.12 P-Value = 0.035 DF = 143

Based on the previous tests, we conclude that for the *CPU Time*, the best performance was achieved by *RSR*, followed by *FJR*, then *Robust with Learning*, then *Robust w/o Learning*, and finally *PR* and *CR* that had the worst performance. This conclusion was expected as both *RSR* and *FJR* are heuristics that do not involve MIP solutions.

Match-up Comparison

The *Match-up* performance of the rules and systems is presented in Table 139. The boxplot is also shown in Figure 27. It is known from Chapter 6 that *FJR* performed the best between the 4 rules (Table 91), followed by *RSR*, then *PR* (Table 92), and finally *CR* (Table 93). The ANOVA results shown in Table 140 indicate that there is a significant difference between the performances of the rules.

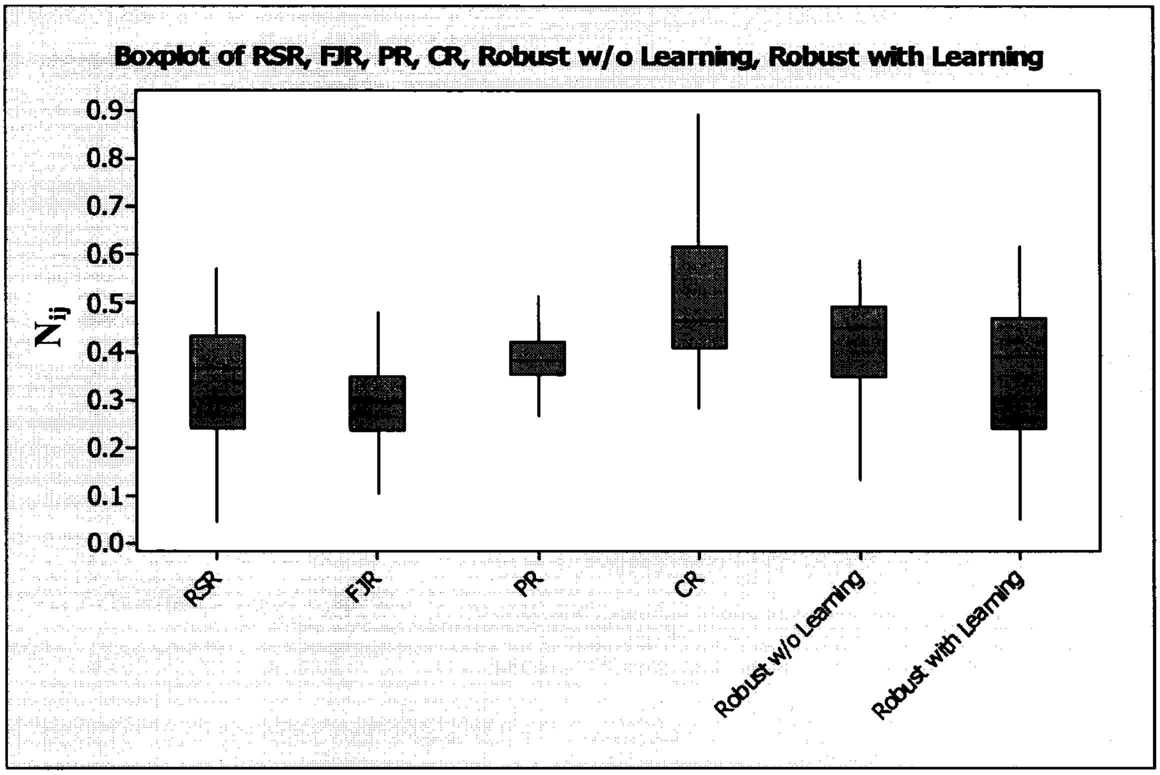

Figure 28. *Match-up* Boxplot for the rules and systems

Table 139. *Match-up* Performance among the rules and systems

	Match-Up Time					
Run	RSR	FJR	PR	CR	Robust w/o Learning	Robust with Learning
1	0.354241	0.307584	0.345619	0.460488	0.512724	0.430718
2	0.562401	0.279382	0.42368	0.593182	0.268162	0.048686
3	0.428822	0.241125	0.333921	0.470756	0.408191	0.508169
4	0.398521	0.288615	0.4028	0.4028	0.576443	0.317961
5	0.337217	0.238775	0.391674	0.401249	0.441644	0.565519
6	0.125621	0.258033	0.468534	0.468534	0.499091	0.479059
7	0.234909	0.190864	0.419655	0.425772	0.560356	0.486824
8	0.073761	0.176932	0.417059	0.859388	0.168022	0.150124
9	0.225741	0.480213	0.36529	0.435064	0.435064	0.454355
10	0.489898	0.236468	0.350289	0.647235	0.366457	0.167859
11	0.357672	0.351365	0.37569	0.452721	0.474343	0.421323
12	0.262525	0.424673	0.341669	0.341669	0.540493	0.474475
13	0.330283	0.212235	0.415199	0.777353	0.225165	0.136019
14	0.220157	0.365415	0.574223	0.360876	0.447123	0.397644
15	0.467017	0.247533	0.340095	0.420035	0.505585	0.415828
16	0.329043	0.274677	0.215425	0.468519	0.513213	0.535702
17	0.238568	0.451456	0.461396	0.461396	0.471337	0.302227
18	0.2221	0.137292	0.177375	0.732493	0.450934	0.40059
19	0.366837	0.232675	0.316756	0.49174	0.525552	0.439256
20	0.363236	0.322112	0.420159	0.64552	0.332103	0.246538
21	0.322724	0.258021	0.277945	0.547999	0.480338	0.4701
22	0.18958	0.164254	0.357614	0.865115	0.179129	0.169354
23	0.280917	0.3636	0.400197	0.393419	0.484573	0.489013
24	0.322599	0.395553	0.445578	0.462734	0.346169	0.454942
25	0.353546	0.359342	0.359342	0.341954	0.504238	0.495544
26	0.391279	0.30416	0.364486	0.51446	0.491037	0.340221
27	0.405319	0.212405	0.393554	0.739091	0.232161	0.188578
28	0.513242	0.382588	0.413905	0.428865	0.364436	0.319614
29	0.41232	0.348677	0.348133	0.315495	0.466171	0.519969
30	0.100848	0.123547	0.362672	0.891088	0.15705	0.155988
31	0.212675	0.271696	0.381833	0.776278	0.249601	0.26504
32	0.44552	0.299263	0.363286	0.540776	0.460452	0.274811
33	0.26767	0.223896	0.453974	0.725463	0.255307	0.283994
34	0.228752	0.347647	0.399794	0.40049	0.58961	0.398682
35	0.051267	0.356691	0.513598	0.690701	0.270625	0.236758
36	0.044022	0.378589	0.378589	0.378589	0.484241	0.577568
37	0.193848	0.180826	0.367847	0.840431	0.198634	0.220532
38	0.435603	0.269728	0.353505	0.576326	0.428405	0.311205
39	0.268349	0.283536	0.341379	0.49774	0.527718	0.452575
40	0.069356	0.439252	0.462371	0.462371	0.462371	0.401107
41	0.402662	0.40816	0.443266	0.443266	0.43946	0.291845
42	0.533495	0.224994	0.312512	0.334392	0.433214	0.517304
43	0.406494	0.275141	0.299292	0.522465	0.523342	0.350183
44	0.161926	0.104935	0.40375	0.880274	0.130762	0.088166
45	0.478395	0.309818	0.36677	0.419165	0.412331	0.441491
46	0.258402	0.330663	0.472631	0.614598	0.369474	0.293715
47	0.571699	0.307192	0.39236	0.514284	0.349716	0.195083
48	0.390966	0.292577	0.350348	0.507803	0.501653	0.359556
49	0.476153	0.237025	0.434236	0.684358	0.230131	0.085033
50	0.363019	0.254314	0.351386	0.424391	0.529084	0.469076
51	0.507712	0.212049	0.296187	0.323971	0.486743	0.517358
52	0.424072	0.272255	0.418443	0.567305	0.444955	0.22611
53	0.392229	0.286365	0.359226	0.381312	0.473467	0.515305
54	0.333688	0.294714	0.368625	0.441605	0.503016	0.466767
55	0.376223	0.344578	0.346922	0.346922	0.465297	0.531517
56	0.456561	0.302462	0.401568	0.411261	0.451813	0.406869
57	0.533806	0.263503	0.414489	0.614703	0.272036	0.148163
58	0.400519	0.402138	0.458372	0.485478	0.373818	0.303869
59	0.285902	0.347144	0.40077	0.436944	0.468359	0.476292
60	0.417339	0.215422	0.382898	0.382223	0.494999	0.491622
61	0.53576	0.370911	0.358032	0.370911	0.471366	0.295699
62	0.309474	0.312776	0.370276	0.426979	0.531048	0.452735
63	0.385167	0.359908	0.468058	0.513832	0.423233	0.244714
64	0.395714	0.224197	0.430879	0.718938	0.237643	0.184779
65	0.2454	0.195248	0.264765	0.83279	0.328454	0.173595
66	0.168622	0.262105	0.501521	0.74431	0.23766	0.202152
67	0.415138	0.40322	0.438974	0.442946	0.422289	0.312804
68	0.237404	0.324725	0.281065	0.281065	0.545757	0.618342
69	0.450864	0.238787	0.405408	0.40935	0.47146	0.430702
70	0.478569	0.334743	0.361391	0.429288	0.429653	0.399282
71	0.410182	0.318701	0.402461	0.617617	0.361339	0.23704
72	0.469506	0.322186	0.372491	0.39884	0.480285	0.383749
73	0.5256	0.293232	0.381692	0.392266	0.430177	0.391338

Table 140. One-Way ANOVA for *Match-up Time*

Anova: Single Factor

SUMMARY

Groups	*Count*	*Sum*	*Average*	*Variance*
RSR	73	25.09667	0.34379	0.016737
FJR	73	21.32688	0.292149	0.006023
PR	73	28.00718	0.38366	0.004064
CR	73	38.02203	0.52085	0.024933
Robust w/o Learning	73	30.0783	0.412032	0.012922
Robust with Learning	73	25.90672	0.354887	0.018727

ANOVA

Source of Variation	*SS*	*df*	*MS*	*F*	*P-value*	*F crit*
Between Groups	2.22014	5	0.444028	31.94217	1.06E-27	2.23488
Within Groups	6.00523	432	0.013901			
Total	8.22537	437				

The following t tests were carried out to determine superiority: *Robust with Learning – FJR* (Table 141), *Robust with Learning – RSR* (Table 142), *Robust with Learning – PR* (Table 143), *Robust w/o Learning – PR* (Table 144), and *Robust w/o Learning – CR* (Table 145).

Table 141. t test for *Robust with Learning – FJR* in the case of *Match-up Time*

Two-sample T for Robust with Learning vs FJR

	N	Mean	StDev	SE Mean
Robust with Lear	73	0.355	0.137	0.016
FJR	73	0.2921	0.0776	0.0091

Difference = mu (Robust with Learning) - mu (FJR)
Estimate for difference: 0.062738
95% CI for difference: (0.026258, 0.099217)
T-Test of difference = 0 (vs not =): T-Value = 3.41 P-Value = 0.001 DF = 113

Table 142. t test for *Robust with Learning – RSR* in the case of *Match-up Time*

Two-sample T for Robust with Learning vs RSR

	N	Mean	StDev	SE Mean
Robust with Lear	73	0.355	0.137	0.016
RSR	73	0.344	0.129	0.015

Difference = mu (Robust with Learning) - mu (RSR)
Estimate for difference: 0.011097
95% CI for difference: (-0.032471, 0.054665)
T-Test of difference = 0 (vs not =): T-Value = 0.50 P-Value = 0.615 DF = 143

Table 143. t test for *Robust with Learning – PR* in the case of *Match-up Time*

Two-sample T for Robust with Learning vs PR

	N	Mean	StDev	SE Mean
Robust with Lear	73	0.355	0.137	0.016
PR	73	0.3837	0.0638	0.0075

Difference = mu (Robust with Learning) - mu (PR)
Estimate for difference: -0.028773
95% CI for difference: (-0.063825, 0.006278)
T-Test of difference = 0 (vs not =): T-Value = -1.63 P-Value = 0.107 DF = 101

Table 144. t test for *Robust w/o Learning – PR* in the case of *Match-up Time*

Two-sample T for PR vs Robust w/o Learning

	N	Mean	StDev	SE Mean
PR	73	0.3837	0.0638	0.0075
Robust w/o Learn	73	0.412	0.114	0.013

Difference = mu (PR) - mu (Robust w/o Learning)
Estimate for difference: -0.028372
95% CI for difference: (-0.058593, 0.001849)
T-Test of difference = 0 (vs not =): T-Value = -1.86 P-Value = 0.065 DF = 113

Table 145. t test for *Robust w/o Learning – CR* in the case of *Match-up Time*

Two-sample T for Robust with Learning vs PR

	N	Mean	StDev	SE Mean
Robust with Lear	73	0.355	0.137	0.016
PR	73	0.3837	0.0638	0.0075

Difference = mu (Robust with Learning) - mu (PR)
Estimate for difference: -0.028773
95% CI for difference: (-0.063825, 0.006278)
T-Test of difference = 0 (vs not =): T-Value = -1.63 P-Value = 0.107 DF = 101

Following the above tests, we conclude that for the *Match-up Time*, the best performance was achieved by *FJR*, followed by *RSR* and *Robust with Learning*, then *PR* and *Robust w/o Learning*, and finally *CR* that had the worst performance.

Shifted Jobs Comparison

The *Shifted Jobs* performance of the rules and systems is presented in Table 146. The boxplot is shown in Figure 28. It is known from chapter 6 that *RSR* performed the best between the four rules as the number of shifted jobs in this rule is always zero (no shifting allowed), followed by *FJR*, then *CR*, and finally *PR*.

It is clear from Figure 28 that the two systems perform worse than *FJR* but better than *CR*. Next, a t test is carried out for *Robust w/o Learning - Robust with Learning* in Table 147. The results indicated that *Robust with Learning* outperformed *Robust w/o Learning* and the difference is statistically significant.

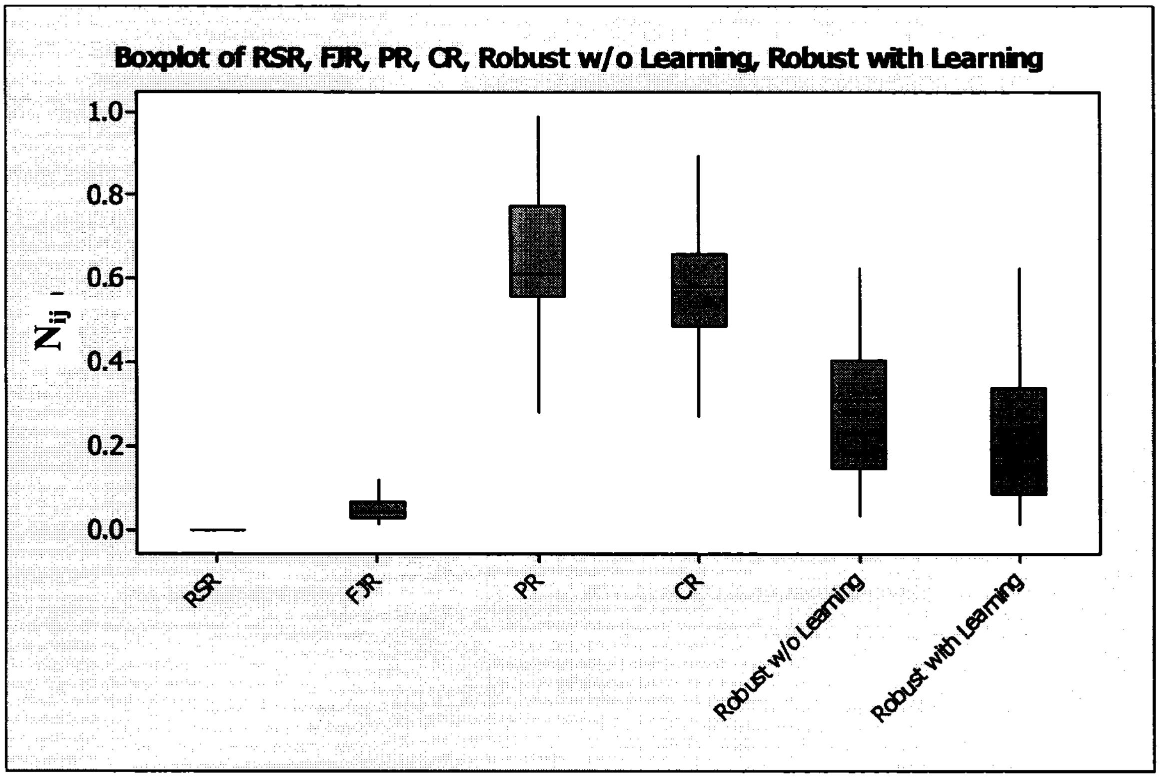

Figure 29. *Shifted Jobs* Boxplot for the rules and systems

Table 147. t test for *Robust w/o Learning - Robust with Learning* in the case of *Shifted Jobs*

Two-sample T for Robust w/o Learning vs Robust with Learning

	N	Mean	StDev	SE Mean
Robust w/o Learn	73	0.291	0.159	0.019
Robust with Lear	73	0.229	0.152	0.018

Difference = mu (Robust w/o Learning) - mu (Robust with Learning)
Estimate for difference: 0.061561
95% CI for difference: (0.010623, 0.112499)
T-Test of difference = 0 (vs not =): T-Value = 2.39 P-Value = 0.018 DF = 143

Table 146. *Shifted Jobs* Performance for the rules and systems

Run	RSR	FJR	PR	CR	Robust w/o Learning	Robust with Learning
1	0	0.074157	0.815722	0.488759	0.208987	0.215728
2	0	0.032739	0.922807	0.344835	0.167999	0.01484
3	0	0.026998	0.773955	0.548968	0.302382	0.086395
4	0	0.026698	0.640741	0.640741	0.347068	0.240278
5	0	0.016867	0.699998	0.548191	0.261445	0.3753
6	0	0.05255	0.504481	0.504481	0.504481	0.483461
7	0	0.042486	0.590554	0.616046	0.403616	0.327142
8	0	0.039945	0.787849	0.611921	0.042948	0.037509
9	0	0.064123	0.280539	0.512986	0.512986	0.625202
10	0	0.012099	0.983424	0.161582	0.078801	0.020297
11	0	0.030429	0.55989	0.66842	0.382389	0.304288
12	0	0.164122	0.369274	0.369274	0.623664	0.558015
13	0	0.036905	0.948209	0.302932	0.078911	0.039285
14	0	0.114216	0.585356	0.513971	0.456863	0.414032
15	0	0.057042	0.670993	0.618455	0.322737	0.24468
16	0	0.042032	0.720974	0.481622	0.326919	0.373622
17	0	0.132283	0.555157	0.555157	0.503111	0.33613
18	0	0.019011	0.986822	0.15485	0.032836	0.027652
19	0	0.025032	0.971171	0.175225	0.151366	0.050847
20	0	0.137751	0.769051	0.583836	0.182717	0.123833
21	0	0.039574	0.489273	0.837726	0.191187	0.143904
22	0	0.044583	0.913249	0.396528	0.061754	0.054223
23	0	0.044646	0.607181	0.521461	0.510746	0.310734
24	0	0.115374	0.584057	0.579518	0.360875	0.423668
25	0	0.101358	0.60815	0.594635	0.36489	0.36489
26	0	0.024198	0.647469	0.681755	0.326372	0.094285
27	0	0.034052	0.750245	0.640699	0.142956	0.070996
28	0	0.116445	0.545348	0.615215	0.456074	0.320222
29	0	0.025414	0.600001	0.537017	0.441989	0.394476
30	0	0.028828	0.753665	0.654731	0.033742	0.036691
31	0	0.074958	0.499312	0.856096	0.07542	0.080499
32	0	0.038329	0.939709	0.269903	0.182384	0.096782
33	0	0.059581	0.671608	0.717452	0.092019	0.148952
34	0	0.026561	0.61909	0.573157	0.413392	0.341498
35	0	0.06128	0.519013	0.845484	0.090588	0.061813
36	0	0.047057	0.376455	0.376455	0.602328	0.592917
37	0	0.038735	0.443977	0.890915	0.062432	0.061293
38	0	0.029762	0.941548	0.276512	0.165312	0.093884
39	0	0.065161	0.675877	0.662012	0.204496	0.242622
40	0	0.041739	0.542602	0.542602	0.542602	0.339126
41	0	0.107287	0.546246	0.546246	0.510266	0.36242
42	0	0.045235	0.569959	0.712449	0.303074	0.271409
43	0	0.038301	0.789797	0.565861	0.205403	0.111195
44	0	0.030076	0.940211	0.335133	0.045042	0.027498
45	0	0.112845	0.522433	0.597663	0.427141	0.417946
46	0	0.053995	0.559379	0.761452	0.262612	0.188164
47	0	0.094552	0.777096	0.533823	0.27873	0.156601
48	0	0.035949	0.653618	0.608331	0.37443	0.247441
49	0	0.038295	0.923744	0.369398	0.088946	0.029395
50	0	0.030673	0.557477	0.716976	0.308261	0.281423
51	0	0.029133	0.737787	0.429708	0.305894	0.42024
52	0	0.016346	0.967624	0.201339	0.131568	0.074755
53	0	0.029436	0.590554	0.597913	0.355068	0.40842
54	0	0.081165	0.594376	0.634334	0.354628	0.334649
55	0	0.029039	0.609817	0.560036	0.448029	0.336021
56	0	0.021724	0.66188	0.612424	0.332789	0.275013
57	0	0.032154	0.951135	0.272948	0.124942	0.064792
58	0	0.097934	0.599936	0.665225	0.378678	0.211102
59	0	0.032449	0.550752	0.657745	0.379738	0.344658
60	0	0.018174	0.602661	0.665181	0.324957	0.297332
61	0	0.165383	0.551276	0.551276	0.54025	0.270125
62	0	0.097193	0.696549	0.556834	0.392821	0.202485
63	0	0.04742	0.573192	0.703652	0.371056	0.190773
64	0	0.042073	0.946991	0.303276	0.086165	0.045147
65	0	0.022562	0.985485	0.160677	0.047266	0.016068
66	0	0.044183	0.663481	0.735882	0.092094	0.088523
67	0	0.066978	0.585832	0.583153	0.462593	0.313456
68	0	0.127891	0.596824	0.596824	0.477459	0.208036
69	0	0.028263	0.697418	0.603461	0.295619	0.247495
70	0	0.055566	0.562609	0.684855	0.3459	0.302836
71	0	0.041371	0.507826	0.818164	0.237386	0.121061
72	0	0.01913	0.587566	0.638124	0.403098	0.29105
73	0	0.04932	0.580874	0.643894	0.306877	0.389076

Based on the previous tests, we conclude that for the *Shifted Jobs*, the best performance was achieved by *RSR*, followed by *FJR*, then *Robust with Learning*, then *Robust w/o Learning*, then *CR*, and finally *PR* that had the worst performance.

Overall Performance Comparison

The *overall performance* including all the performance measures is presented in Table 148. The boxplot is shown in Figure 29. It is known from chapter 6 that *FJR* performed the best among the four rules (Table 98), followed by *RSR*, then *PR* and *CR* (Table 99). The ANOVA results shown in Table 149 indicate that there is a significant difference between the performances of the rules and systems.

Table 149. One-Way ANOVA for the *Overall Performance*

Anova: Single Factor

SUMMARY

Groups	*Count*	*Sum*	*Average*	*Variance*
RSR	73	79.92988	1.09493	0.042709
FJR	73	63.15228	0.8651	0.077417
PR	73	135.9991	1.863002	0.154734
CR	73	139.1484	1.906143	0.150971
Robust w/o Learning	73	93.85687	1.285711	0.200583
Robust with Learning	73	70.04649	0.959541	0.205709

ANOVA

Source of Variation	*SS*	*df*	*MS*	*F*	*P-value*	*F crit*
Between Groups	74.9431	5	14.98862	108.0749	7.83E-74	2.23488
Within Groups	59.91292	432	0.138687			
Total	134.856	437				

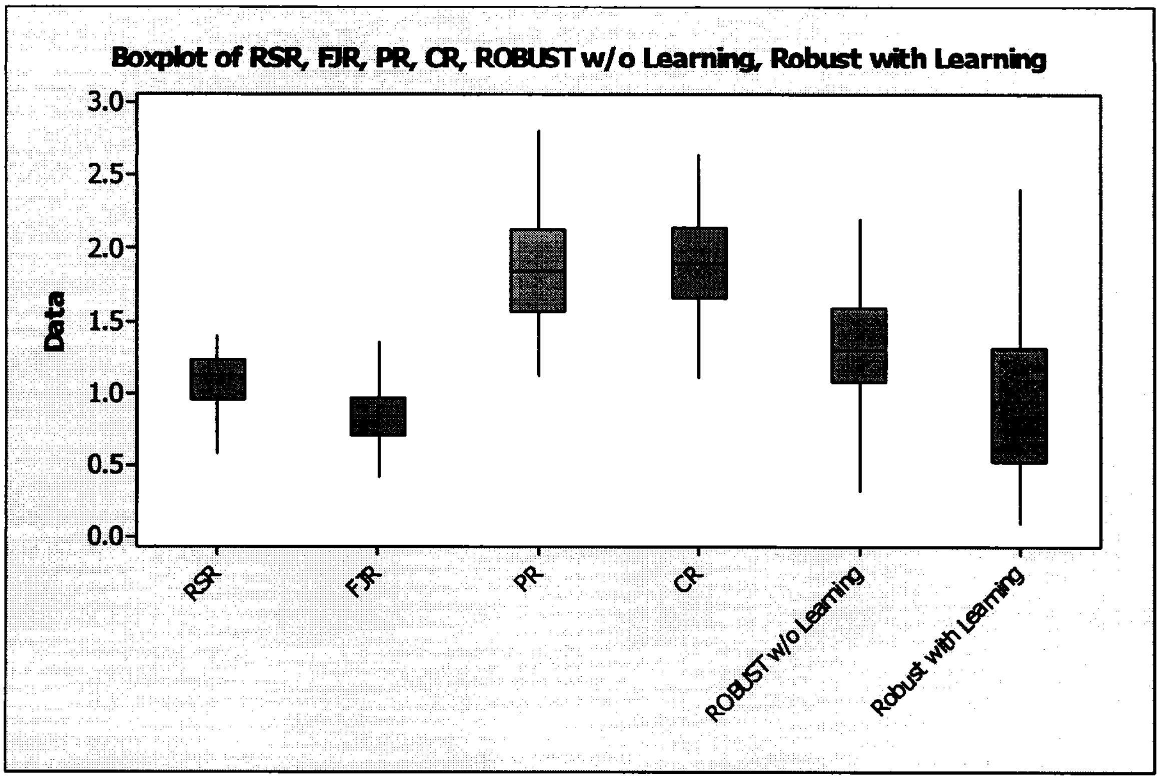

Figure 30. *Overall Performance* Boxplot for the rules and systems

The following t tests were carried out to determine superiority: *Robust with Learning – FJR* (Table 150), *Robust with Learning – RSR* (Table 151), *Robust w/o Learning – PR* (Table 152), and *Robust w/o Learning – RSR* (Table 153).

Based on these tests, we conclude that for the *Overall Performance*, the best performance was achieved by *FJR* and *Robust with Learning*, followed by *RSR*, then *Robust w/o Learning*, and finally *PR* and *CR*.

Table 148. *Overall Performance* among the rules and systems

Run	Overall Performance					
	RSR	FJR	PR	CR	Robust w/o Learning	Robust with Learning
1	1.018034	0.782932	2.191187	1.63885	1.404897	1.491585
2	1.289459	0.84301	2.400859	1.621883	1.366345	0.098013
3	1.140737	0.842265	1.919961	2.154687	1.073577	0.83716
4	1.354484	1.140803	2.142639	1.724714	1.620887	0.940953
5	0.960048	0.865796	2.120676	1.792561	1.088556	1.709177
6	0.582581	1.583474	1.478808	1.512373	1.507774	1.665218
7	0.793703	0.541015	2.175748	2.182318	1.461647	1.129586
8	0.948836	0.509037	2.253649	2.321862	0.390932	0.305797
9	0.792683	1.059411	1.12018	1.921768	2.128383	1.407049
10	1.068532	0.623498	2.381245	1.441892	1.59179	0.318202
11	1.063683	0.965704	1.315026	2.360143	1.216419	0.81515
12	0.8223	1.913159	1.413591	1.114686	2.016282	1.341729
13	1.067398	0.660517	2.558644	1.956241	0.733667	0.426782
14	0.906046	0.851116	1.72615	1.562339	1.966741	1.570539
15	1.191643	1.544156	1.456134	1.59321	1.229077	1.244989
16	0.954164	1.055353	1.318913	2.125477	1.227377	1.135064
17	1.133647	1.312572	1.575676	2.046477	1.304277	0.814327
18	0.982539	1.136139	1.732019	1.261919	1.082115	1.234298
19	1.0289	1.024081	2.414412	1.101469	1.433485	1.221008
20	1.137729	0.959413	1.91896	1.861014	1.590738	0.765681
21	1.03121	0.884935	1.323836	2.574478	0.982727	0.961447
22	0.960721	0.501768	2.566735	2.00163	0.578161	0.477294
23	0.937255	0.82055	1.837447	1.35536	1.666052	1.865242
24	1.321863	0.867569	1.614462	1.858964	1.62348	1.116302
25	1.6884	0.845898	1.627212	1.89255	1.51834	1.182593
26	1.147486	0.82725	1.894762	2.14106	1.262702	0.923586
27	1.251638	0.577372	1.966878	2.509816	0.683771	0.409477
28	1.344705	0.859059	1.929386	1.69901	1.557546	1.0179
29	1.135912	0.928851	1.597101	1.911839	1.312521	1.325332
30	1.024676	0.282458	1.768336	2.569211	0.196304	0.265156
31	1.215088	0.414354	1.20341	2.631915	0.386102	0.400655
32	0.987752	0.801822	2.541018	1.578665	1.483999	0.558731
33	1.197611	0.720057	1.630816	2.571008	0.48392	0.681645
34	0.665936	1.033228	1.99824	2.023367	1.467979	1.456072
35	1.060721	0.510142	1.502599	2.415144	0.405021	0.32015
36	0.377682	0.764956	1.436324	1.163225	2.19381	2.409438
37	1.20505	0.302118	1.216361	2.64274	0.323643	0.342382
38	1.039465	0.793274	2.384811	1.67579	1.513188	0.643315
39	1.014917	0.951124	1.782271	2.122903	1.292538	0.904506
40	0.916875	0.832412	1.760983	1.678072	1.774801	1.386241
41	1.342902	0.822236	1.712332	1.665426	1.60373	1.094646
42	1.34039	0.86563	1.720158	1.748328	1.297283	1.325549
43	1.177189	0.839384	1.727155	2.217782	1.280081	0.735766
44	0.896372	0.492811	2.745785	1.843276	0.65068	0.331783
45	1.403778	0.582332	1.459225	1.671372	1.575822	1.593297
46	1.076149	0.855875	1.844212	2.467736	0.990709	0.761057
47	1.368337	0.967651	1.96083	2.10665	1.010814	0.790576
48	1.084568	0.92109	1.857294	2.109174	1.619275	0.838469
49	1.200669	0.683292	2.586969	1.822679	0.923628	0.290127
50	1.078924	0.845062	1.531477	2.264531	1.221122	0.901593
51	1.315172	0.664227	2.14049	1.234417	1.254332	1.342703
52	0.914345	0.830534	2.809536	1.295328	1.122776	0.485709
53	1.080042	1.014536	1.843751	1.809686	1.235281	1.393565
54	0.878105	0.968532	2.120705	1.688834	1.771054	1.220557
55	1.099547	1.130861	1.668237	2.053637	1.505446	1.20672
56	1.188911	0.853268	2.131484	1.914213	1.188513	0.871383
57	1.257956	0.788273	2.500821	1.659972	1.131197	0.50203
58	1.36092	0.738872	1.946658	1.532287	1.31765	1.195114
59	0.857564	1.051342	1.890836	2.142185	1.332321	1.337628
60	1.272075	0.591026	1.644208	2.070188	1.194911	0.911894
61	1.252988	1.072646	1.892919	1.89135	1.64404	0.943245
62	0.856996	1.013509	2.381815	1.550637	1.683043	0.980294
63	1.151937	0.86986	1.98513	2.158	1.44674	0.771685
64	1.198583	0.581441	2.303254	2.051144	0.869817	0.49261
65	0.834148	0.701901	1.867451	2.251434	1.098456	0.324637
66	1.137022	0.521737	1.592447	2.581365	0.481732	0.41379
67	1.227846	1.407726	1.608668	2.029998	1.68052	0.904162
68	1.321769	1.351914	1.547756	1.857342	1.474305	1.076423
69	1.218243	0.964142	1.781924	1.648562	1.303323	1.476831
70	1.116814	0.979094	1.429089	1.667719	1.967338	1.131255
71	1.241682	0.771004	1.514718	2.595505	0.935338	0.51246
72	1.137536	0.889104	1.539326	1.721372	2.018056	1.311229
73	1.278295	0.788721	1.514987	1.513678	1.885963	1.457935

Table 150. t test for *Robust with Learning – FJR* in the case of *Overall Performance*

Two-sample T for Robust with Learning vs FJR

	N	Mean	StDev	SE Mean
Robust with Lear	73	0.96	0.454	0.053
FJR	73	0.865	0.278	0.033

Difference = mu (Robust with Learning) - mu (FJR)
Estimate for difference: 0.094441
95% CI for difference: (-0.028874, 0.217756)
T-Test of difference = 0 (vs not =): T-Value = 1.52 P-Value = 0.132 DF = 119

Table 151. t test for *Robust with Learning – RSR* in the case of *Overall Performance*

Two-sample T for Robust with Learning vs RSR

	N	Mean	StDev	SE Mean
Robust with Lear	73	0.96	0.454	0.053
RSR	73	1.095	0.207	0.024

Difference = mu (Robust with Learning) - mu (RSR)
Estimate for difference: -0.135389
95% CI for difference: (-0.251124, -0.019654)
T-Test of difference = 0 (vs not =): T-Value = -2.32 P-Value = 0.022 DF = 100

Table 152. t test for *Robust w/o Learning – PR* in the case of *Overall Performance*

Two-sample T for ROBUST w/o Learning vs PR

	N	Mean	StDev	SE Mean
ROBUST w/o Learn	73	1.286	0.448	0.052
PR	73	1.863	0.393	0.046

Difference = mu (ROBUST w/o Learning) - mu (PR)
Estimate for difference: -0.577291
95% CI for difference: (-0.715215, -0.439367)
T-Test of difference = 0 (vs not =): T-Value = -8.27 P-Value = 0.000 DF = 141

Table 153. t test for *Robust w/o Learning – RSR* in the case of *Overall Performance*

Two-sample T for ROBUST w/o Learning vs RSR

	N	Mean	StDev	SE Mean
ROBUST w/o Learn	73	1.286	0.448	0.052
RSR	73	1.095	0.207	0.024

Difference = mu (ROBUST w/o Learning) - mu (RSR)
Estimate for difference: 0.190781
95% CI for difference: (0.076259, 0.305302)
T-Test of difference = 0 (vs not =): T-Value = 3.30 P-Value = 0.001 DF = 101

Computational Tests Summary

In this chapter, a robust reactive scheduling system has been introduced for the unrelated parallel machine problem. The system with and without the learning capability was compared to the rules introduced in chapter 6 and evaluated based on four performance measures: *Cmax Difference*, *CPU Time*, *Match-up Time*, and *Shifted Jobs*. Extensive computational tests indicated the following conclusions about the system:

Robust Scheduling System w/o Learning

The *Robust w/o Learning* ranked 2nd among the 6 alternatives (4 rules and 2 systems) in the case of *Cmax Difference* (after *Robust with Learning* and tied with *CR* and *PR*), and 4th for *CPU Time*, *Shifted Jobs*, *Match-up Time*, and *Overall Performance* (after *RSR*, *FJR*, and *Robust with Learning*).

The following was determined from the DoE factor analyses of *Robust w/o Learning* performance:

- *Cmax Difference* improves when the number of machines increases.
- *CPU Time* improves when the time between breakdowns, the idle time, and the number of machines increase and the number of jobs decreases.
- *Match-up Time* decreases when the number of machines, idle time, and time between breakdowns increase and the number of jobs and processing time range decreases.
- *Shifted Jobs* declines when the number of jobs decreases and the number of machines, repair duration, idle time, and the time between breakdowns increase.

Robust Scheduling System with Learning

Robust with Learning ranked 1st among the 6 alternatives in the case of *Cmax Difference* and *Overall Performance* (tied with *FJR*), 2nd for *Match-up Time* (after *FJR* and tied with *RSR*), and 3rd for *CPU Time* and *Shifted Jobs* (after *RSR* and *FJR*).

The following was determined from the experimental design factor analyses of *Robust with Learning* performance:

- *Cmax Difference* improves when the processing time range decreases.
- *CPU Time* improves when the number of jobs decreases.
- *Match-up Time* decreases when the number of machines and time between breakdowns increase and the number of jobs and processing time range decrease.

- *Shifted Jobs* declines when the number of jobs decreases and the time between breakdowns increases.

Furthermore, the average usage of each of the three rules (*RSR*, *FJR*, and *PR*) incorporated in both *Robust with Learning* and *Robust w/o Learning* was recorded for all problem replications (14892 replicates). The results indicated the following: *FJR* usage is almost the same in both systems, *RSR* usage is 37.31% higher in *Robust with Learning*, and *PR* usage is 12.82% less in *Robust with Learning*. This observation is extremely important because it explains the reason why the CPU time is smaller in *Robust with Learning*, as the latter utilizes more the simple heuristic *RSR* and less the *PR* rule which requires a high CPU time.

Finally, as the superiority of each of the 6 alternatives depend strongly on which performance measure is being evaluated, Table 154 below summarizes the ranks of the alternatives for all possible combinations of the four performance measures addressed in this dissertation (15 alternatives). All necessary ANOVA and t tests were carried out to make sure that the reported results are statistically significant. Note that the alternatives are ranked between 1 and 6, where 1 indicates the best performance and 6 the worst one.

Table 154. Ranks of the Rules and Systems for all combinations of Performance Measures (1 = Best, 6 = Worst)

Performance Measures				**Repair Rules and Systems**					
Cmax Difference	***CPU Time***	***Match-up Time***	***Shifted Jobs***	***RSR***	***FJR***	***PR***	***CR***	***Robust w/o Learning***	***Robust with Learning***
●				6	5	2	2	2	1
	●			1	2	6	6	4	3
		●		2	1	4	6	4	2
			●	1	2	6	5	4	3
●	●			4	2	6	6	2	1
●		●		6	2	2	5	2	1
●			●	4	1	6	5	3	1
	●	●		1	1	5	6	4	3
	●		●	1	2	6	6	4	3
		●	●	1	1	6	6	4	3
●	●	●		4	1	5	6	3	1
●	●		●	3	1	6	6	4	1
●		●	●	3	1	6	6	3	1
	●	●	●	1	2	6	6	4	3
●	●	●	●	3	1	6	6	4	1

CHAPTER VIII

GENERALIZABILITY, CONCLUSIONS, AND FUTURE RESEARCH

In this chapter, conclusions are summarized based on the results of the computational study performed in previous chapters. What makes the problem addressed in this research unique is that up to our knowledge, no published work was found on the generation of predictable schedules in parallel machine environments. Furthermore, most of the literature that addressed schedule repair and rescheduling strategies were designed for either a flow shop or a job shop, which require different recovery rules than the one necessary for a parallel machine environment. The research gap extends to an absence of publications tackling schedule repair and rescheduling strategies for unrelated parallel machines. Finally, no previous literature was found on designing a robust scheduling system that combines schedule repair, rescheduling, system response, and learning in a parallel scheduling environment.

This chapter includes three sections. The first section lists the contributions of this research and its generalizability. In the second section, conclusions on the performance of the repair and rescheduling rules and the robust systems are presented. Finally, future research is discussed in the third section.

RESEARCH CONTRIBUTIONS AND GENERALIZABILITY

Research Contributions

The main contributions of this research are the following:

1. New and improved heuristics (*FJR* and *PR*) for scheduling repair and rescheduling in unrelated parallel machine environments.
2. An analysis of six repair and rescheduling alternatives with four performance measures, and a comparison study that allows readers to choose the rule that will optimize the performance measure(s) they desire.
3. An idle time insertion rule (*MCFJI*) equipped with a learning parameter that guarantees robust predictable schedules.
4. A robust predictable-reactive scheduling construct, which will react according to an event driven policy and attempt to overcome the perturbations using schedule repair as long as possible, otherwise it will use complete rescheduling.

Research Generalizability

Even though the developed rules and systems in this research were only tested on unrelated parallel machines subjects to breakdowns, they can be generalized to the following:

1. The environment must be a parallel machine one; however, it does not have to be the unrelated machines (which is the hardest case), i.e. the rules can be applied also to uniform and identical machines. The rules were developed for the parallel machine

environment independent of which generalization of the problem is used. As can be seen from the previous chapters, the rationale of all the rules is to shift the jobs upon a disruption either on the same machine or to another machine, i.e. the only requirement is to have parallel machines.

2. The machine breakdowns can be replaced by almost any other disruption type causing a delay, such as new job arrivals, absenteeism, the closing of a processing unit, etc… For example, in the case of a new job arrival, the latter's time of arrival will be considered as the start of a breakdown, and its processing time as the delay of a breakdown. Following this, any of the rules or systems can be implemented to repair the schedule.
3. The approach followed in this research can be adapted to environments other than parallel machines. The rules will have to change or be modified; however, the system architecture can still be utilized.

RESEARCH CONCLUSIONS

Based on the results discussed in previous chapters, the following conclusions can be drawn:

1. *Robust with Learning* ranked 1st among the 6 alternatives (4 rules and 2 systems) in the case of *Cmax Difference* and *Overall Performance* (tied with *FJR*), 2nd for *Match-up Time* (after *FJR* and tied with *RSR*), and 3rd for *CPU Time* and *Shifted Jobs* (after *RSR* and *FJR*). Moreover, the processing time range had a significant effect on *Robust with Learning* in the case of *Cmax Difference*, as the latter improves when the processing range decreases. Furthermore, *CPU Time* improves when the number of jobs decreases; *Match-up Time* decreases when the number of machines and time between breakdowns increase and the number of jobs and processing time range decreases, and *Shifted Jobs* declines when the number of jobs decreases and the time between breakdowns increases.
2. *Robust w/o Learning* ranked 2nd among the 6 alternatives in the case of *Cmax Difference* (after *Robust with Learning* and tied with *CR* and *PR*), and 4th for *CPU Time*, *Shifted Jobs*, *Match-up Time*, and *Overall Performance* (after *RSR*, *FJR*, and *Robust with Learning*). Furthermore, in the case of *Robust w/o Learning*, *Cmax Difference* improves when the number of machines increases, *CPU Time* decreases when the time between breakdowns, the idle time, and the number of machines increase and the number of jobs decreases; *Match-up Time* decreases when the number of machines, idle time, and time between breakdowns increase and the number of jobs and processing time range decreases, and *Shifted Jobs* declines when the number of jobs decreases and the number of machines, repair duration, idle time, and the time between breakdowns increase.

3. *FJR* ranked 5th among the rules in the case of *Cmax Difference* (after *Robust with Learning*, *Robust w/o Learning*, *CR* and *PR*), 2nd for *CPU Time* and *Shifted Jobs* (after *RSR*), and was the best in the case of *Match-up Time* and *Overall Performance* (tied with *Robust with Learning*). In addition, *FJR* performance is impacted as follows: *Cmax Difference* improves when the number of machines increases, *CPU Time* improves when the time between breakdowns and the number of machines increase and the number of jobs decreases, *Match-up Time* decreases when the number of machines, idle time, and time between breakdowns increase and the number of jobs decreases, and *Shifted Jobs* declines when the number of jobs and repair durations decrease and the number of machines and the time between breakdowns increase.
4. *PR* ranked 2nd among the rules in the case of *Cmax Difference* (after *Robust with Learning* and tied with *Robust w/o Learning* and *CR*), 4th for *Match-up Time* (after *FJR*, *Robust with Learning*, and *RSR* and tied with *Robust w/o Learning*), and was the worst in the case of *CPU Time* (tied with *CR*), *Shifted Jobs*, and *Overall Performance* (tied with *CR*). In addition, *PR* performance is impacted as follows: *Cmax Difference* improves when the number of machines increases, *CPU Time* improves when the time between breakdowns increases and the number of jobs decreases, *Match-up Time* decreases when the number of machines and time between breakdowns increase and the number of jobs decreases, and *Shifted Jobs* declines when the number of jobs and repair durations decrease and the number of machines and the time between breakdowns increase.
5. *RSR* had the worst *Cmax Difference* performance among the 6 alternatives, the best *CPU* and *Shifted Jobs* performances, the second best *Match-up Time* (after *FJR* and tied with *Robust with Learning*), and ranked 3rd for *overall performance* (after *FJR* and *Robust*

with Learning). Recall that *RSR* performed the best in the case of *Shifted Jobs* because it does not shift jobs between the machines. Moreover, *RSR* was the finest in *CPU Time* as it is a simple heuristic with a computational complexity of O(mn) at the most. In addition, *RSR* performance is impacted as follows: *Cmax Difference* improves when the number of machines increases, *CPU Time* improves when the time between breakdowns increases and the number of jobs decreases, and *Match-up Time* decreases when the number of machines, idle time, and time between breakdowns increase and the number of jobs decreases. *Shifted Jobs* is always zero when using *RSR*.

6. *CR* ranked 2nd among the rules in the case of *Cmax Difference* (after *Robust with Learning* and tied with *PR* and *Robust w/o Learning*), 5th for *Shifted Jobs* (after *RSR*, *FJR*, *Robust with Learning*, and *Robust w/o Learning*), and was the worst in the case of *CPU Time* (tied with *PR*), *Match-up Time*, and *Overall Performance* (tied with *PR*). In addition, *CR* performance is impacted as follows: *Cmax Difference* improves when the number of machines increases, *CPU Time* improves when the time between breakdowns increases and the number of jobs and machines decreases, *Match-up Time* decreases when the number of machines and time between breakdowns increase and the number of jobs decreases, and *Shifted Jobs* declines when the number of jobs and repair durations decrease and the number of machines and the time between breakdowns increase.
7. A new idle time insertion rule, *CFJI*, was introduced and compared to the traditional initial schedule where no idle time is built-in, and to Mehta's rule *OSMH*. *CFJI* outperformed the other rules; however, as the problem size increased, it overestimated the idle time needed for insertion.

8. The learning parameter was successful in predicting the realized schedule and was determined to be an essential addition to the robust system. In fact, *MCFJI* (which is *CFJI* with the learning parameter) performed much better than *CFJI* alone. Furthermore, *Robust with Learning* outperformed *Robust w/o Learning* and delivered the finest performances for almost all performance measure combinations.

FUTURE RESEARCH

In this research, repair and rescheduling rules and systems were developed and compared for the unrelated parallel machine problem. This dissertation is innovative in the sense that no previous work was found on rescheduling in unrelated parallel machine environments. The extensions listed below can be considered in future research:

1. Extending this dissertation results to unrelated parallel machine environments with machine eligibility restrictions. Scheduling in the presence of machine eligibility restrictions when not all machines can process all the jobs is a practical problem into which there has been little research (Centeno and Armacost, 2004).
2. Extending the problem to include sequence dependent setup times. This will increase the problem's complexity and the proposed rules will need to be modified to account for this extension.
3. Extending the results to identical and uniform parallel machine environments to verify if the rules would dominance hold.
4. Extending the problem to different environments other than the parallel one. Such environments include the flow shop and job shop problems where more work has been done on schedule repair and rescheduling. This extension can also be beneficial to compare the rules and systems developed in this dissertation to existing ones for the flow shop or job shop problems.
5. Altering the proposed rules and systems to be able to absorb more than one overlapping event (disruption). Such extension can be a great addition to the current literature and

will provide the rules with the ability to handle a broader variety of problems, such as for example the case where several jobs can arrive after time 0.

6. Modifying and testing the rules for different quality measures such as tardiness, earliness, or weighted tardiness and earliness.
7. In the case of *PR*, we are dealing with bicriteria optimization problem (minimizing *Shifted Jobs* and *Cmax Difference*). The hierarchical approach followed by Alagoz and Azizoglu (2003) was used in this research, i.e. minimizing the less important measure (*Shifted Jobs*) subject to the constraint that the more important measure (*Cmax Difference*) is kept at its optimum. An extension to the *PR* rule is to investigate the simultaneous approach for bicriteria problems, i.e. generation of efficient schedules or optimization of a weighted combination of the two performance measures.
8. The learning parameter used with *CFJI* proved to be effective in predicting the realized schedule $Cmax_R$ and has aided the robust system in reaching superior performance measures. However, as the literature and findings on machine learning are almost abundant, it is worthy to investigate other intelligent parameters. The fields that can be explored are brain models, adaptive control theory, artificial intelligence, and evolutionary models.

REFERENCES

Abumaizar, R. J., and Svestka, J. A. (1997) Rescheduling job shops under random disruptions. *International Journal of Production Research*, V. 35, pp. 2065-2082.

Adachi, T., Talavage, J.J., and Moodie, C.L. (1989) A rule based control method for a multi-loop production system. *Artificial Intelligence in Engineering,* V. 4, pp. 115-125.

Ahn, W., and Brewer, W.F. (1993) *Psychological studies of explanation-based learning.* Kluwer Academic Publishers, Boston.

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

Alagoz, O., Azizoglu, M. (2003) Rescheduling of identical parallel machines under machine eligibility constraints. *European Journal of Operational Research*, V. 149, pp. 523-532.

Algorithms and Theory of Computation Handbook, CRC Press LLC (1999) *Lower Bound.* From Dictionary of Algorithms and Data Structures [online], Paul E. Black, ed., U.S. National Institute of Standards and Technology. Available: http://www.nist.gov/dads/HTML/lowerbound.html, last accessed: 10/24/2006.

Al-Salem, A. (2004) Scheduling to minimize makespan on unrelated parallel machines with sequence dependent setup times. *Engineering Journal of the University of Qatar*, V.17, pp. 177-187.

Anderson, E.J., Potts, C.N. (2004) Online Scheduling of a Single Machine to Minimize Total Weighted Completion Time. *Mathematics of Operations Research*, V. 29, pp. 686-697.

Anderson, J.R. (1991) *The place of cognitive architecture in rational analysis*. In: Architectures for Intelligence, Hillsdale, NJ, pp. 1-24.

Anderson, T.W. (1958) *An Introduction to Multivariate Statistical Analysis*. John Wiley, New York.

Arnaout, J-P, Rabadi, G., and Mun, J.H. (2006) A Dynamic Heuristic for the Stochastic Unrelated Parallel Machine Scheduling Problem. *International Journal of Operations Research*, V. 3, pp. 136-143.

Arnaout, J-P (2005) Predictable Scheduling of Unrelated Parallel Machines subject to Breakdowns. *Proceedings of the 26th National Conference of the American Society for Engineering Management,* Virginia Beach: 464-469.

Arnaout, J-P, and Rabadi, G. (2005) Minimizing the total weighted completion time on unrelated parallel machines with stochastic times. *Proceedings of the 2005 Winter Simulation Conference*, Orlando: 2141-2147.

Aytug, H., Bhattacharyya, S., Koehler, G.J., and Snowdon, J. (1994) A Review of Machine Learning in Scheduling. *IEEE Transactions on Engineering Management*, V. 41, pp. 165-171.

Azizoglu, M., and Alagoz, O. (2005) Parallel-machine rescheduling with machine disruptions. *IIE Transactions*, V. 37, pp. 1113-1118.

Bean, J.C., Birge, J.R., Mittenehal, J., and Noon, C.E. (1991) Match-up scheduling with multiple resources, release dates and disruption. *Operations Research*, V. 39, pp. 470-483.

Beasley, J.E. (2006) OR Notes: Simulation. Available: http://people.brunel.ac.uk/~mastjjb/jeb/or/sim.html, last accessed : 10/24/2006.

Bollinger, J., and Duffie, N. (1988) *Computer Control of Machines and Processes*. Reading, Addison-Wesley, MA.

Brignell, John (2006) *What is the Normal Distribution and what is so normal about it?* Available: http://www.numberwatch.co.uk, last accessed: 10/24/2006.

Bruno, J.L., Downey, P.J., and Frederickson, G.N. (1981) Sequencing tasks with exponential service times to minimize the expected flow time or makespan. *Journal of the ACM*, V. 28, pp. 100-113.

Centeno, G., and Armacost, R. (2004) Minimizing makespan on parallel machines with release time and machine eligibility restrictions. *International Journal of Production Research*, V. 42, pp. 1243-1256.

Chong, C.S., Sivakumar, A.I., and Gay, R. (2003) Simulation-based scheduling for dynamic discrete manufacturing. *Proceedings of the 2003 Winter Simulation Conference*, pp. 1465- 1473.

Church, L. K., and Uzsoy, R. (1992) Analysis of periodic and event-driven rescheduling policies in dynamic shops. *International Journal of Computer Integrated Manufacturing*, V. 5, pp. 153-163.

Costa, A.M., Vargas, P.A., Von Zuben, F.J., França, P.M. (2002) Makespan minimization on parallel processors: an immune based approach. *Proceedings of the Special Sessions on Artificial Immune Systems in the 2002 Congress on Evolutionary Computation, IEEE World Congress on Computational Intelligence*, Honolulu, Hawaii: 115-123.

Cowling, P., and Johansson, M. (2002) Using real time information for effective dynamic scheduling. *European Journal of Operational Research*, V. 139, pp. 230-244.

Cowling, P.I., Ouelhadj, D., and Petrovic, S. (2003) A multi-agent architecture for dynamic scheduling of steel hot rolling. *Journal of Intelligent Manufacturing*, V. 14, pp. 457-470.

Crama, Y. (2005) *Advanced Operations Research*. HEC Management School, University of Liège.

Davenport, A.,Gefflot, C., and Beck, J. (2001) Slack-based techniques for robust schedules. *Proceedings of the 6th European Conference on Planning (ECP-2001)*, pp. 7–18. Available: http://tidel.mie.utoronto.ca/pubs/uncertainty-ecp.ps.gz, last accessed: 10/24/2006.

Davis, E., and Jaffe, J. (1981) Algorithms for Scheduling Tasks on Unrelated Processors. *Journal of the Association for Computing Machinery*, V. 28, pp. 721-736.

Davis, R., and Smith, R. (1983) Negotiation as a metaphor for distributed problem solving. *Artificial Intelligence,* V. 20, pp. 63-109.

Derbyshire, J. (2004) *Prime Obsession: Berhhard Riemann and the Greatest Unsolved Problem in Mathematics*. Plume.

Dietterich, T.G. (1989) Machine Learning. *Annual review of computer science*, V. 3. Palo Alto, CA: Annual Reviews, Inc.

Dorn, J., Kerr, R., and Thalhammer, G. (1993) Reactive Scheduling in a Fuzzy-Temporal Framework. *Proceedings of the IFIP International Workshop on Knowledge-Based Reactive Scheduling*, Athens, Greece.

Feldmann, S., Sgall, J., and Teng, S.-H. (1991) Dynamic scheduling on parallel machines. *Proceedings of IEEE*, San Juan, Puerto Rico: 111-120.

Fisher, R.A. (1960). *The Design of Experiments*. Hafner Publishing Company, New York.

Fox, M.S., and Smith, S.F. (1984) ISIS---A knowledge-based system for factory scheduling. *Expert Systems*, V. 1, pp. 25-44.

Garner, B.J., and Ridley, G.J. (1994) Application of neural network process models in reactive scheduling. Knowledge based reactive scheduling. *IFIP Trans. B (Appl. Tech.)*, V. B-15, pp. 19-28.

Garey, M.R., and Johnson, D.S. (1979) *Computers and Intractability: A Guide to the Theory of NP-Completeness*. Freeman, San Francisco.

Ghirardi, M., and Potts, C.N. (2005) Makespan minimization for unrelated parallel machines: a recovering beam search approach. *European Journal of Operational Research*, V. 165, pp. 457-467.

Gluck, M., and Rumelhart, D. (1989) *Neuroscience and Connectionist Theory*. The Developments in Connectionist Theory. Erlbaum Associates, Hillsdale, NJ.

Graves, S. C. (1981) A review of production scheduling. *Operations Research,* V. 29, pp. 646-675.

Grigoriev, A., Sviridenko, M., and Uetz, M. (2005) Unrelated Parallel Machine Scheduling with Resource Dependent Processing Times. *Proceedings of the 11th Conference on Integer Programming and Combinatorial Optimization*, Lecture Notes in Computer Science 3509, 2005, pp. 182-195.

Guinet, A. (1991) Textile Production Systems: a succession of Non-identical Parallel Processor Shops. *Journal of Operational Research Society*, V. 42, pp. 655-671.

Guo, B., Nonaka, Y. (1999) Rescheduling and optimization of schedules considering machine failures. *International Journal of Production Economics*, V. 60-61, pp. 503-513.

Herroelen, W., and Leus, R. (2004) Robust and reactive scheduling: a review and classification of procedures. *International Journal of Production Research*, V. 42, pp. 1599-1620.

Hoogeveen, J.A., Schuurman, P., and Woeginger, G.J. (2001) Non-approximability results for scheduling problems with minsum criteria. *INFORMS journal on computing*, V. 13, pp. 157-168.

Jozefowska, J., Mika, M., Roycki, R., Waligora, G., and Wglarz, J. W. (1998) Local search meta-heuristics for discrete-continuous scheduling problems. *European Journal of Operational Research*, V. 107, pp. 354-370.

Kizilisik, Ö. (1999) *Predictive and Reactive Scheduling*, Department of Industrial Engineering, Bilkent University.

Kouikoglou, V.S., and Phillis, Y.A. (1997) Review of a fast simulation method for the analysis of queuing networks. *Applied Stochastic Models and Data Analysis*, V. 13, pp. 73-83.

Langley, P. (1996) *Elements of Machine Learning*. Morgan Kaufman Publishers, Inc. San Francisco, CA.

Langley, P., and Carbonell, J.G. (1984) Approaches to Machine Learning. *Journal of the American Society for Information Science*, V. 35, pp. 306-316.

Langley, P., and Carbonell, J.G. (1987) Language acquisition and machine learning. *In: Mechanisms of language acquisition*. Hillsdale, NJ.

Law, A., and Kelton, D. (2000) *Simulation Modeling and Analysis*. McGraw-Hill, 3rd edition, 2000.

Lawler, E.L., and Labetoulle, J. (1978) On preemptive scheduling of unrelated parallel processors by linear programming. *Journal of the Association for Computing Machinery*, V. 25, pp. 612-619.

Lee, I. (2001) Artificial intelligence search methods for multi-machine two-stage scheduling with due date penalty, inventory, and machining costs. *Computers and Operations Research*, V. 28, pp. 835-852.

Lee, K. (1989) *Automatic speech recognition: The development of the Sphinx system*. Kluwer Academic Publishers, Boston.

Leon, V. J., Wu, S. D. and Storer, R. H. (1994) Robustness measures and robust scheduling for job shops. IIE Transactions, V. 26, pp. 32-41.

Leung, J. (2004). *Handbook of Scheduling: Algorithms, Models, and Performance Analysis*. Chapman & Hall, New York.

Liaw, C., Lin, Y., Cheng, C., and Chen, M. (2003) Scheduling unrelated parallel machines to minimize total weighted Tardiness. *Computers and Operations Research*, V. 30, pp. 1777-1789.

Low, C.Y., 2005. Simulated annealing heuristic for flow shop scheduling problems with unrelated parallel machines. *Computers and Operations Research*, V. 32, pp. 2013-2026.

MacCarthy, B. L., and Liu, J. (1993). Addressing the gap in scheduling research: a review of optimization and heuristic methods in production scheduling. *International Journal of Production Research*, V. 31, pp. 59-79.

Martello, S., Soumis, F., and Toth, P. (1997) Exact and approximation algorithms for makespan minimization on unrelated parallel machines. *Discrete Applied Mathematics*, V. 75, pp. 169-188.

Mehta, S. V., and Uzsoy, R. (1998) Predictable scheduling of a Job Shop subject to breakdowns. *IEEE Transactions on Robotics and Automation*, V. 14, pp.365-378.

Mehta, S. V. and Uzsoy, R. (1999) Predictable scheduling of a single machine subject to breakdowns. *International Journal of Computer Integrated Manufacturing*, V. 12, pp.15-38.

Mendez, C., and Cerda, J. (2004) An MILP framework for batch reactive scheduling with limited discrete resources. *Computers and Chemical Engineering*, V. 28, pp.1059-1068.

Michalski, R.S., Carbonell, J.G., and Mitchell, T.M. (1983) *Machine Learning: An Artificial Intelligence Approach*. Morgan Kaufman Publishers, Inc. Los Altos, CA.

Mitchell, T. (1997) Does Machine Learning really Work? *Artificial Intelligence Magazine*, V.18, pp. 11-20.

Miyashita, K. (1995) Case based knowledge acquisition for schedule optimization. *Artificial Intelligence in Engineering*, V. 9, pp. 277-287.

Mokotoff, E., and Chretienne, P. (2002) A cutting plane algorithm for the unrelated parallel machine scheduling problem. *European Journal of Operational Research,* V. 141, pp. 515-525.

Mosheiov, G., and Sidney, J. (2003) Scheduling with general job-dependent learning curves. *European Journal of Operational Research*, V.147, pp. 665-670.

Nilsson, N. (1996) *Introduction to Machine Learning*. An early draft of a proposed textbook. Available: http://ai.stanford.edu/people/nilsson/mlbook.html, last accessed: 10/24/2006.

NIST/SEMATECH e-Handbook of Statistical Methods. Available: http://www.itl.nist.gov/div898/handbook/, last accessed: 10/24/2006.

Nof, S.Y., and Grant, F.H. (1991) Adaptive/predictive scheduling; review and a general framework. *Production Planning and Control*, V. 2, pp. 298-312.

O'Donovan, R., Uzsoy, R., and McKay, K. N. (1999) Predictable scheduling of a single machine with breakdowns and sensitive jobs. *International Journal of Production Research*, V. 37, pp. 4217-4233.

O'Kane, J. F. (2000) A knowledge-based system for reactive scheduling decision-making in FMS. *Journal of Intelligent Manufacturing*, V. 11, pp. 461-474.

Ouelhadj, D. (2003) *A Multi-Agent system for the integrated dynamic scheduling of steel production*, Doctoral Dissertation, The School of Computer Science and Informations Technology, University of Nottingham, Nottingham, UK.

Ow, P.S., Smith, S.F., and Howie, R. (1988) A cooperative scheduling system. *Proceedings of the 1988 Expert Systems and Intelligent Manufacturing,* pp.70-89.

Pinedo, M. (2002) *Scheduling theory, algorithms and systems*. Second edition, Prentice Hall.

Pomerleau, D.A. (1989) *ALVINN: An autonomous land vehicle in a neural network*. Carnegie Mellon University, Pittsburgh, PA.

Rabadi, G., Moraga, R., and Al-Salem, A. (2006) Heuristics for the Unrelated Parallel Machine Scheduling Problem with Setup Times. *Journal of Intelligent Manufacturing*, V. 17, pp. 85-97.

Raheja, A.S., and Subramaniam, V. (2002) Reactive recovery of job shop schedules - a review. *International Journal of Advanced Manufacturing Technology*, V. 19, pp. 756-763.

Reeves, C. R. (1995) *Modern heuristic techniques for combinatorial problems*. John Wiley & Sons, McGraw-Hill International (UK) Limited.

Ross, P. (1996) *Taguchi Techniques for Quality Engineering*, New York: McGraw Hill.

Rowland, T., and Weisstein, E. W. (2006a) *Upper Bound*. From *MathWorld*--A Wolfram Web Resource. Available: http://mathworld.wolfram.com/UpperBound.html, last accessed: 10/24/2006.

Rowland, T., and Weisstein, E. W. (2006b) *Lower Bound*. From *MathWorld*--A Wolfram Web Resource. Available: http://mathworld.wolfram.com/LowerBound.html, last accessed: 10/24/2006.

Sabuncuoglu, I., and Bayiz, M. (2000). Analysis of reactive scheduling problems in a job shop environment. *European Journal of Operational Research*, V. 126, pp. 567-586.

Sauer, J., and Bruns, R. (1997) Knowledge-based scheduling systems in industry and medicine. *Expert IEEE*, V. 12, pp. 24-31.

Schmidt, G. (1994) How to apply fuzzy logic to reactive production scheduling. Knowledge based reactive scheduling. *IFIP Trans. B (Appl. Tech.)*, V. B-15, pp. 57-66.

Schrage, L. (2001) *Optimization Modeling with LINGO*. Lindo Systems, Inc. Chicago, Illinois.

Sejnowski, T., Koch, C., and Churchland, P. (1988) Computational Neuroscience. *Science*, V. 241, pp. 1299-1306.

Selfridge, O. (1993) The Gardens of Learning. *Artificial Intelligence Magazine*, pp. 36-48.

Shafaei, R., and Brunn, P. (1999) Workshop scheduling using practical (inaccurate) data Part 2: an investigation of the robustness of scheduling rules in a dynamic and stochastic environment. *International Journal of Production Research*, V. 37, pp. 4105-4117.

Shavlik, J.W., and Dietterich, T.G., eds. (1990) *Readings in Machine Learning*. Morgan Kaufnab Publishers, Inc. San Mateo, California.

Shaw, M.J., Park, S.C., and Raman, N. (1990) *Intelligent Scheduling with Machine Leaning Capabilities: The Induction of Scheduling Knowledge*. Carnegie Mellon University - The Robotics Institute-Technical Report.

Shukla, C. S. and Chen, F. F. (1996) The state of the art in intelligent real-time FMS control: a comprehensive survey. *Journal of Intelligent Manufacturing*, V. 7, pp. 441-455.

Subramaniam, V., Raheja, A.S., and Rama Bhupal Reddy, K. (2005) Reactive repair tool for job shop schedules. *International Journal of Production Research*, V. 43, pp. 1-23.

Sun, J. and Xue, D. (2001) A dynamic reactive scheduling mechanism for responding to changes of production orders and manufacturing resources. *Computers in Industry*, V. 46, pp.189-207.

Sundararaghavan, P. S., Kunnathur, A. S., and Viswanathan, I. (1997) Minimizing Makespan in Parallel Flowshops. *The Journal of the Operational Research Society*, V. 48, pp. 834-842.

Suresh, V. and Chaudhuri, D. (1993) Dynamic scheduling a survey of research. *International Journal of Production Economics*, V. 32, pp. 53-63.

Taguchi, G. (1993). *Taguchi Methods: Design of Experiments*, American Supplier Institute, Inc., Michigan.

Tesauro, G. (1995) Temporal difference learning and TD-gammon. *Communications of the ACM*, V. 38, pp. 58-68.

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

Vieira, G. E., Hermann, J. W. and Lin, E. (2003) Rescheduling manufacturing systems: a framework of strategies, policies and methods. *Journal of Scheduling*, V. 6, pp. 36-92.

Vredeveld, T., and Hurkens, C. (2002) Experimental Comparison of Approximation Algorithms for Scheduling Unrelated Parallel Machines. *INFORMS Journal on Computing*, V. 14, pp. 175-189.

Weng, M.X., Lu, J., and Ren, H. (2001). Unrelated parallel machine scheduling with setup consideration and a total weighted completion time objective. *International Journal of Production Economics*, V. 70, pp. 215-226.

Williams, H. P., and Brailsford, S. C. (1996) *Computational logic and integer programming.* In Beasley, J. E., editor, Advances in Linear and Integer Programming, pages 249–281. Clarendon Press, Oxford, UK.

Williams, H.P. (1999) *Model Building in Mathematical Programming*, Fourth Edition, John Wiley and Sons, LTD.

Wu, S. D., Storer, R. H. and Chang, P. C. (1991) A rescheduling procedure for manufacturing systems under random disruptions. *Proceedings of Joint USA/German Conference on New Directions for Operations Research in Manufacturing*, pp. 292-306.

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

Yih, Y., and Thesen, A. (1991) Semi-Markov decision models for real time scheduling. *International Journal of Production Research,* V. 29, pp. 2331-2346.

Youssef, H., Sait, S. M., and Adiche, H. (2001) Evolutionary algorithms, simulated annealing and tabu search: a comparative study. *Engineering Applications of Artificial Intelligence*, V. 14 (2), pp. 167-181.

APPENDIX A: ROBUST SYSTEM IMPLEMENTATION CODE IN VISUAL C++

This is the C++ Main Function that implements the Robust System (with or w/o Learning). The program calculates the 95% confidence intervals for each of the Performance measures and the averages of usages of each of the rules

```
#ifndef for
#define for if (0) {} else for
#endif

# include <iostream>
# include <conio.h>
# include <fstream>
# include <time.h>
# include <string>
# include <math.h>
# include <windows.h>
# include "lingd90.h"
using namespace std;

// Global Inputs needed for the problem
const int nom=8,noj=100,Mincrease=1,iteration =45,Teta=1;
float beta1=0.1,beta2=0.2, alpha=0.8*(1);
int maxpro=150,minpro=1;
ifstream fin;
ofstream fout;

// Functions Prototypes
void inputdata (double[][500]);
void LINGO1 (double[][500],double[][500]);
void sort(double[][500],double[][500], int[][500], int[]);
void assign(double[][500],int[],int[][500], int[],float[][500],float[][500],float[][500],float&, float&,
        int&,int&,int&,int&,float&);
int jobposit (int[],int[],int,float[],float[],float, double[][500],int[][500]);
int jobposup (float[],int[],int[],int,float[],float[], double[][500],int[][500]);
void RepairRule1 (int,float[][500], float[][500], float[], int[],int[],double[][500], int[][500], float&,
        int&, float[]);
void RepairRule2 (float[],float[],int,float[][500],float[][500],float[],int[],int[],double[][500],
        int[][500],float&, int&, float[],int&);
void RepairRule5 (float[],float[],int,float[][500],float[][500],float[],int[],int[],double[][500],
        int[][500],float&,int&,float[]);
void LINGO2 (double[][500],double[10][500],double[10][500],double[],int,double&);
void LINGO3 (double [][500],double[10][500],double[10][500],double [],int ,double&,double& );
void LINGO4 (double [][500],double [10][500],double [10][500],double [],int ,double& ,double );
```

```
// Main Function
void main()
{
int countIter=0;
float CmaxDiffCount=0,MatchCount=0,Tidle=0;
double CPUtime=0,CPUCount=0,JobsCounter=0,RSRcounter=0,FJRcounter=0,PRcounter=0;
float Record[10][1000]={0};
float varCmax=0,varCPU=0,varMatch=0,varSJobs=0,avgCmax=0,avgSJobs=0,avgCPU=0,
        avgMatch=0,avgTidle=0,avgRSR=0,avgFJR=0,avgPR=0;

fout.open("results.txt");

if(!fout)
{
    cout<<"Output Failure"<<endl
      <<"Press any Key"<<endl;
    getch();
    return;
}// end of error check

while (countIter < iteration)          //Run the appropriate iterations number
{
clock_t starti=0,endi=0;
double data[10][500]={0},Xdecisions[10][500]={0};
int place[10][500]={0}, number[nom]={0},jobposi[nom]={0};
float idles[10][500]={0}, start[10][500]={0},finito[10][500]={0};
float matchcounter=0,CmaxDifference=0;
int jobsc=0,RSRC=0,FJRC=0,PRC=0;

inputdata(data);

LINGO1(data,Xdecisions);

sort(data,Xdecisions,place,number);

starti=clock();                        //Record the start of CPU time

assign(data,number,place,jobposi,idles,start,finito,matchcounter,CmaxDifference,jobsc,RSRC,FJRC,
        PRC,Tidle);

endi=clock();                          //Record the end of CPU time
CPUtime=double ((endi-starti)/double(CLOCKS_PER_SEC));

countIter=countIter +1;

Record[1][countIter]=(CmaxDifference);    //record cmax
Record[2][countIter]=CPUtime;             //record the CPU
Record[3][countIter]=matchcounter;        //record the matching time
Record[4][countIter]=jobsc;               //record the shifted jobs
Record[5][countIter]=RSRC;                //record the use of RSR
Record[6][countIter]=FJRC;                //record the use of FJR
```

```
Record[7][countIter]=PRC;                    //record the use of PR
Record[8][countIter]=Tidle;                  //record the Tidle
} //end of while loop

for(int i=1;i<(countIter+1);i++)             //Get the averages
{
        avgCmax=avgCmax + Record[1][i];      //AVG Cmax
        avgCPU=avgCPU + Record[2][i];        //AVG CPU
        avgMatch=avgMatch + Record[3][i];    //AVG Match
        avgSJobs=avgSJobs+Record[4][i];      //AVG SJobs
        avgRSR=avgRSR+Record[5][i];          //AVG RSR
        avgFJR=avgFJR+Record[6][i];          //AVG FJR
        avgPR=avgPR+Record[7][i];            //AVG PR
        avgTidle=avgTidle+Record[8][i];      //AVG Tidle
}

avgCmax=avgCmax/countIter;
avgCPU=avgCPU/countIter;
avgMatch=avgMatch/countIter;
avgSJobs=avgSJobs/countIter;
avgRSR=avgRSR/countIter;
avgFJR=avgFJR/countIter;
avgPR=avgPR/countIter;
avgTidle=avgTidle/countIter;

for(int i=1;i<(countIter+1);i++)             //Get the variance for each rule
{
        varCmax=varCmax+(pow((avgCmax-Record[1][i]),2));
        varCPU=varCPU+(pow((avgCPU-Record[2][i]),2));
        varMatch=varMatch+(pow((avgMatch-Record[3][i]),2));
        varSJobs=varSJobs+(pow((avgSJobs-Record[4][i]),2));
}

varCmax=(varCmax/(countIter-1));
varCPU=(varCPU/(countIter-1));
varMatch=(varMatch/(countIter-1));
varSJobs=(varSJobs/(countIter-1));

varCmax=2.009 * pow((varCmax/countIter),0.5);
varCPU=2.009 * pow((varCPU/countIter),0.5);
varMatch=2.009 * pow((varMatch/countIter),0.5);
varSJobs=2.009 * pow((varSJobs/countIter),0.5);

cout<<"Required iterations "<<countIter<<endl;
cout<<"Cmax average is "<<avgCmax<<" and the LCI is "<<avgCmax - varCmax<<" and UCI
        "<<avgCmax + varCmax<<endl;
cout<<"CPU average is "<<avgCPU<<" and the LCI is "<<avgCPU - varCPU<<" and the UCI
        "<<avgCPU + varCPU<<endl;
cout<<"Match average is "<<avgMatch<<" and the LCI is "<<avgMatch - varMatch<<" and the UCI
        "<<avgMatch + varMatch<<endl;
```

```
cout<<"SJobs average is "<<avgSJobs<<" and the LCI is "<<avgSJobs - varSJobs<<" and the UCI
        "<<avgSJobs + varSJobs<<endl;
cout<<"RSR average is "<<avgRSR<<endl;
cout<<"FJR average is "<<avgFJR<<endl;
cout<<"PR average is "<<avgPR<<endl;

alpha=1+((nom*(avgCmax))/avgTidle);                    //Determine the learning parameter

fout.close();

getch();

}// End of main function
```

inputdata Function will input the processing time of jobs on the unrelated parallel machines.
The processing time will be randomly generated from a uniform distribution
between minpro and maxpro

```
void inputdata (double datas[][500])
{
        for(int i=1;i<(nom+1);i++)                                //input jobs processing time
        {
                for(int j=1;j<(noj+1);j++)
                {
                        datas[i][j] = rand() % (maxpro - minpro) +minpro ;
                }
        }
        for(int i=1;i<(nom+1);i++)                                //Display jobs processing time
        {
                for(int j=1;j<(noj+1);j++)
                {
                        cout<<datas[i][j]<<" " ;
                }
                cout<<endl;
        }
} // The end of input data
```

Below is the C++ LINGO1 function that interfaces with LINGO. Information is passed to LINGO so the latter can generate the optimal initial schedule. The LINGO file that contains MIP[1] is called LINGO1.Lng (Appendix E)

```
void LINGO1 (double datak[][500],double X[10][500])
{
char cScript[256];                                          //LINGO interface
double dObjective, dStatus=-1.;
double dnoj[]= {noj};
double dnom[] = {nom};
double dX[1000]={0};
int nError=-1, nPointersNow;
int index = 0,nM=1;

// create the LINGO environment object
        pLSenvLINGO pLINGO;
  pLINGO = LScreateEnvLng();
  if ( !pLINGO)
  {
    printf( "Can"t create LINGO environment!\n");
    goto FinalExit;
  }
  // Open LINGO's log file
  nError = LSopenLogFileLng( pLINGO, "LINGO.log");
  if ( nError) goto ErrorExit;

  // Pass memory transfer pointers to LINGO

  // @POINTER(1)
  nError = LSsetPointerLng( pLINGO, dnoj, &nPointersNow);
  if ( nError) goto ErrorExit;

  // @POINTER(2)
  nError = LSsetPointerLng( pLINGO, dnom, &nPointersNow);
  if ( nError) goto ErrorExit;

  // @POINTER(3)
 double datas2[1000];
    for ( int i = 0; i<(nom); i++)
      for ( int j = 0; j < (noj); j++)
        datas2[ i * noj + j] = datak[i+1][j+1];
          nError = LSsetPointerLng( pLINGO, datas2, &nPointersNow);
          if ( nError) goto ErrorExit;

  // @POINTER(4)
  nError = LSsetPointerLng( pLINGO, &dObjective, &nPointersNow);
  if ( nError) goto ErrorExit;
```

```
// @POINTER(5)
nError = LSsetPointerLng( pLINGO, &dStatus, &nPointersNow);
if ( nError) goto ErrorExit;

// @POINTER(6)
      nError = LSsetPointerLng( pLINGO, dX, &nPointersNow);
      if ( nError) goto ErrorExit;

// Here is the script we want LINGO to run
strcpy( cScript, "SET ECHOIN 1 \n TAKE LINGO1.Lng \n GO \n QUIT \n");

// Run the script
nError = LSexecuteScriptLng( pLINGO, cScript);
if ( nError) goto ErrorExit;

// Close the log file
LScloseLogFileLng( pLINGO);

// Any problems?
if ( nError || dStatus != LS_STATUS_GLOBAL_LNG)
{
  // Had a problem
  printf( "Unable to solve!");
}

// Output the decision variables
for ( int i = 1; i<(nom+1); i++)
{
   for ( int j = 1; j < (noj+1); j++)
            {
                    X[i][j]=dX[index];
                    index++;
            }
}
for (int i=1;i<nom+1;i++)
{
        cout<<"the decisions on machine "<<i<<" are: "<<endl;
        for(int j=1;j<noj+1;j++)
                cout<<" "<<X[i][j];
}
        cout<<"the objective is "<<dObjective<<" and status is "<<dStatus<<endl;

              goto NormalExit;
ErrorExit:
  printf("LINGO Error Code: %d\n", nError);

NormalExit:
  LSdeleteEnvLng( pLINGO);

FinalExit: ;
}
```

Below is the C++ Sort function. It will assign the jobs to the machines according to the Optimal solution obtained via LINGO1.Lng (i.e. MIP[1])

```
void sort(double thedata[][500],double XI[][500], int places[][500], int numbers[])
{

for (int i=1; i<10;i++)
{
        numbers[i]=1;
}

for (int i=1; i< (nom+1); i++)
{
        for (int j=1; j <(noj+1); j++)
        {
                if(XI[i][j] == 1)
                {
                        places[i][numbers[i]]=j;
                        numbers[i]=numbers[i] +1;
                }
        }
}

for (int i=1; i< (nom+1); i++)                    //Get the accurate number of jobs on each machine
{
        numbers[i]=numbers[i]-1;
}

for (int i=1; i< (nom+1); i++)
{
        for (int j=1; j <numbers[i]+1; j++)
        {
        cout<<" "<<places[i][j];
        }
        cout<<endl;
}
}
```

The Assign Function executes the initial optimal schedule subject to breakdowns, then applies the appropriate repair and rescheduling rules to repair the schedule

```
void assign(double dataw[][500],int thenumbers[],int theplaces[][500], int jobpos[],float idle[][500],
        float S[][500], float F[][500], float& matchcount,float& CmaxDiff,int& jobct,int& RSR, int&
        FJR, int& PR,float& Totalidle)
{
int l=1,state=0,imakespan=0,Mtotal[nom]={0};
float pmakespan=0,rmakespan=0,residle=0,Rmatchcount=0,Pmatchcount=0,Fmatchcount=0;
float procomp[nom]={0},lamda[nom]={0},Mexpected[nom]={0},tidle[nom]={0};
double r,rk;
float comp[nom]={0},left1=0,left2=0,breakdown=0,repair[nom]={0},repairs=0,findpos[nom]={0};

// This part will calculate the expected processing time on each machine
for (int i=1; i<(nom +1); i++)
{
        for(int j=1;j<(thenumbers[i]+1);j++)
        {
                Mtotal[i] = Mtotal[i] + dataw[i][theplaces[i][j]];
        }
}

for (int i=1; i<(nom +1); i++)
{
        Mexpected[i]=(float(Mtotal[i])/float(thenumbers[i] ));
        cout<<"Mexpected of machine "<<i<<" is "<<Mexpected[i]<<endl;
}

// This part will calculate the objective function CmaxSi
for(int i=1;i<(nom +1);i++)
{
        if(Mtotal[i] > imakespan)
        {
                imakespan = Mtotal[i];
        }
}

// This part will calculate the repair time and lamda for each machine
for(int i=1; i<(nom +1);i++)
{
        lamda[i]=((((1/((-Teta*Mexpected[i])*log(0.1)))+(1/((-Teta*Mexpected[i])*log(0.2)))+
                (1/((-Teta*Mexpected[i])*log(0.3)))+(1/((-Teta*Mexpected[i])*log(0.4)))+
                (1/((-Teta*Mexpected[i])*log(0.5)))+(1/((-Teta*Mexpected[i])*log(0.6)))+
                (1/((-Teta*Mexpected[i])*log(0.7)))+(1/((-Teta*Mexpected[i])*log(0.8)))+
                (1/((-Teta*Mexpected[i])*log(0.9))))/9));
        repair[i]= (beta1*Mexpected[i])+(((beta2*Mexpected[i])-(beta1*Mexpected[i]))*(0.5));
}
```

```
        // This part will add idle time to jobs and calculate CmaxP using CFJI rule (Chapter 4)
for(int i=1; i<(nom +1); i++)
{
        for(int j=1;j<thenumbers[i]+1;j++)
        {
                idle[i][j]= alpha * repair[i] * lamda[i] * dataw[i][theplaces[i][j]]*
                                                        (1-(float(j)/float(thenumbers[i])));
                tidle[i]=tidle[i]+idle[i][j];
                S[i][j]=F[i][j-1]+idle[i][j-1];                 // Start time of job j on machine i
                F[i][j]=S[i][j] + dataw[i][theplaces[i][j]];    // Finish time of job j on machine i
        }
}

for (int i=1; i<(nom +1); i++)
{
        residle=residle+tidle[i];
        for(int j=1;j<(thenumbers[i]+1);j++)
        {
                procomp[i] = procomp[i] + dataw[i][theplaces[i][j]] + idle[i][j];
        }
}

Totalidle = residle;

for(int i=1;i<(nom +1);i++)
{
        if(procomp[i] > pmakespan)
        {
                pmakespan = procomp[i];
        }
}

// This part will generate the events (breakdowns) and calculate the realized schedule makespan
int j=1,re=1,nM=1,nM2=1, machine=0;
int rm[noj]={0};
bool karen=false,hobbi=true;
float residue1=0,RepairF[nom]={0};
float location[nom]={0},findposition[nom]={0},finish[nom]={0},finish2=0,Match[nom][500]={0};
float matching[10][500]={0};

for (int i=1; i <(nom +1); i++)        //Finish of jobs
{
        finish[i]=F[i][thenumbers[i]];
}

while (hobbi)
{
        // Generate breakdowns
        r =((double)rand()/((double)(RAND_MAX)+(double)(1)));
        breakdown =(-Teta*Mexpected[1])* log(r);
        cout<<endl<<"breakdown is: "<<breakdown<<endl;
```

```
rk = (rand() % nom) + 1;          //Determine on which machine the breakdown will occur
machine=rk;

//Determine the location of the breakdown on each machine
for (int i=1; i<(nom +1); i++)
{
        location[i]=location[i]+breakdown;
}

karen=false;

for (int i=1; i<(nom +1); i++)              //Exit the while loop if all jobs are processed
{
        if(location[i] < finish[i])
        {
                karen=true;
        }
}
if(karen == false)
{
        break;
        hobbi=false;
}

if(location[machine] < finish[machine])
{
        if (RepairF[machine] > location[machine])        //Ensure Breakdown after repair
        {
                for(int z=1;z<(nom +1);z++)
                {                                        //Assume the breakdown did not occur
                        location[z]=location[z]-breakdown;
                }
                continue;
        }                                                //Determine the repair time
        r =((double)rand()/((double)(RAND_MAX)+(double)(1)));
        repairs= (beta1* Mexpected[machine])+ (((beta2*Mexpected[machine])-
                                        (beta1*Mexpected[machine]))*r);
        RepairF[machine]=location[machine] + repairs;
        residle=residle-repairs;

        //Determine the job position on the machine upon the breakdown
        jobpos[machine]=jobposit(thenumbers,jobpos,machine,location,findposition,repairs,
                                                        dataw,theplaces);
        if (S[machine][jobpos[machine]] < RepairF[machine])
        {
                state=0;
                if(residle>0)   //Still able to apply RSR and FJR
                {
                        RepairRule1 (machine,S,F,RepairF,thenumbers,jobpos,dataw,
                                        theplaces,Rmatchcount,state,finish);
```

```
                        if(state == 1)
                        {
                                RSR=RSR+1;
                                matchcount=matchcount+Rmatchcount;
                                Rmatchcount=0;
                        }
                        else
                        {
                        RepairRule2 (location,findposition,machine,S,F,RepairF,
                                        thenumbers,jobpos,dataw,theplaces,
                                                Fmatchcount,jobct,finish,state);
                        if(state==1)
                        {
                                FJR=FJR +1;
                                matchcount=matchcount+Fmatchcount;
                                Fmatchcount=0;
                        }
                }
        }
        if(state!=1)
        {
                RepairRule5 (location,findposition,machine,S,F,RepairF,thenumbers,
                        jobpos,dataw,theplaces,Pmatchcount,jobct,finish);
                PR=PR+1;
                matchcount=matchcount+Pmatchcount;
                Pmatchcount=0;
        }
    }
}
}       //End of while

for(int i=1;i<(nom +1);i++)
{
        if(F[i][thenumbers[i]] > rmakespan)
        {
                rmakespan = F[i][thenumbers[i]];
        }
}
} //End of Function
```

jobposit function will calculate the job position on down machine when the breakdown occurs

```
int jobposit (int num[],int jobp[],int mtype,float locat[],float findpos[],float rep, double datak[][500],
                                                                                    int jplaces[][500])
{

int status=0;

while (jobp[mtype] < (num[mtype]+1) && status==0)
{
        findpos[mtype] = findpos[mtype] + datak[mtype][jplaces[mtype][jobp[mtype]]];

        if (findpos[mtype] == locat[mtype])
        {
                jobp[mtype]=jobp[mtype] + 1;
                status=1;
        }
        else if (findpos[mtype] > locat[mtype])
        {
                jobp[mtype] = jobp[mtype];
                status=1;
        }
        if(jobp[mtype] > num[mtype])     //in case of the last job
        {
                jobp[mtype]=jobp[mtype]+1;
        }
        jobp[mtype]=jobp[mtype] +1;
}
jobp[mtype]=jobp[mtype] - 1;
findpos[mtype]=locat[mtype] + rep;
cout<<"Job   "<<jobp[mtype]<<endl;
return jobp[mtype];

}   //End of Function
```

jobposup function will calculate the job position on the up machines when the breakdown occurs

```
int jobposup (float trackpos[],int nemra[],int jobpup[],int matype,float locate[],float findpose[],
                                                      double dataz[][500],int jplac[][500])
{

int status=0,jobpos=0;

trackpos[matype]=findpose[matype];
jobpos=jobpup[matype];

while ((jobpos < nemra[matype]+1) && status==0)
{
        trackpos[matype] = trackpos[matype]+ dataz[matype][jplac[matype][jobpos]];

        if (trackpos[matype] == locate[matype])
        {
                jobpos=jobpos + 1;
                status=1;
        }
        else if (trackpos[matype] > locate[matype])
        {
                jobpos = jobpos +1;
                status=1;
        }
        jobpos=jobpos +1;
        }
        jobpos=jobpos - 1;
        return jobpos;

}   //End of Function
```

This is the *RSR* rule when implemented as the first rule in the Robust System

```
void RepairRule1 (int mach,float SI[][500], float FI[][500], float ReF[], int thenumero[],int jobpi[],
                double datam[][500], int joplaces[][500], float& matchc, int& status1, float finisia[])
{

float awal[10][500]={0}, ekher[10][500]={0},petit=0,finitio[10][500]={0};
int index=0,matchsignal=0;

status1=0;
for(int i=1; i<(nom +1); i++)     // Use temporary S and F arrays so the original won't b modified
{
        for(int j=1;j<(thenumero[i]+1);j++)
        {
                awal[i][j]=SI[i][j];
                ekher[i][j]=FI[i][j];
                finitio[i][j]=FI[i][j];
        }
}

int k = 0, lecmax=0;
awal[mach][jobpi[mach]]=ReF[mach];                //Shift 1 job to the right
ekher[mach][jobpi[mach]]= awal[mach][jobpi[mach]] + datam[mach][joplaces[mach][jobpi[mach]]];

if(ekher[mach][jobpi[mach]] <= awal[mach][(jobpi[mach] +1)])            //RSR Successful
{
        status1=1;
        for(int i=1; i<(nom +1); i++)                           //Reupdate the start and finish of the jobs
        {
                for(int j=1;j<(thenumero[i]+1);j++)
                {
                        SI[i][j] =awal[i][j];
                        FI[i][j]= ekher[i][j];
                }
        }
}
else        //Shift 2 jobs to the right
{
        awal[mach][jobpi[mach]+1]=ekher[mach][jobpi[mach]];
        ekher[mach][jobpi[mach]+1]= awal[mach][jobpi[mach]+1] +
                                        datam[mach][joplaces[mach][jobpi[mach]+1]];
}
                                                                        //RSR Successful
if((status1==0) && (ekher[mach][jobpi[mach]+1] <= awal[mach][(jobpi[mach] +2)]))
{
        status1=1;
```

```
        for(int i=1; i<(nom +1); i++)                    //Reupdate the start and finish of the jobs
        {
                for(int j=1;j<(thenumero[i]+1);j++)
                {
                        SI[i][j] =awal[i][j];
                        FI[i][j]= ekher[i][j];
                }
        }
}
} //End of function
```

This is *FJR* when implemented as the second rule in the Robust System

```
void RepairRule2 (float locati[],float findposi[], int machi,float SE[][500], float FE[][500],
        float RepF[], int lenumero[],int jobsp[], double datap[][500], int jplas[][500], float& mathc,
                int& jobcount, float fini[],int& status2)
{

float awal[10][500]={0}, ekher[10][500]={0},awil[10][500]={0}, ekhir[10][500]={0},
        track[nom]={0},path[nom]={0},wpath[nom]={0},residle[nom]={0},compi[nom]={0};
int joblocat[nom]={0}, ma7al[10][500]={0}, ma7il[10][500]={0};
int fitsignal=0,k=0,states=0,petitindex=0,jindex=0;
bool hobbi=true, karen=false;
float petit=0,makespani=0;

for(int i=1; i<(nom +1); i++)            //Use temporary S and F arrays so the original won't b modified
{
        for(int j=1;j<(lenumero[i]+1);j++)
        {
                awal[i][j]=SE[i][j];
                awil[i][j]=SE[i][j];
                ekher[i][j]=FE[i][j];
                ekhir[i][j]=FE[i][j];
                ma7al[i][j]=jplas[i][j];
                ma7il[i][j]=jplas[i][j];
        }
}

for(int i=1; i<(nom +1); i++)            //Get the jobs "on the right" locations on each machine
{
        joblocat[i]=jobposup (track,lenumero,jobsp,i,locati,findposi,datap,jplas);
        if(i == machi)                   //for the down machine, locate the job after the down job
        {
                joblocat[i] = jobsp[i]+1;
                cout<<"jobloc "<<joblocat[i]<<endl;
                cout<<"machi "<<machi<<endl;
                track[i]=RepF[i];        //Because it can only start once the repair finishes
        }
}

jindex=jplas[machi][jobsp[machi]];

for(int i=1; i<(nom +1); i++)            //assume the down job on each machine to see which one is
                                         //more appropriate
{
        path[i]=track[i]+datap[i][jindex];
        wpath[i]=path[i];                //Use it in case we couldn't fit the job on any machine
}
```

```
while(hobbi)
{
        karen=false;
        for(int i=1;i<(nom +1); i++)    //Check if we still have jobs to shift in order to fit the down job
        {
                if((lenumero[i]) >= (joblocat[i]+k))
                {
                        karen=true;
                }
                else
                {
                        path[i]=1000000;  //assigned a large number so this path is not chosen
                }
        }
        if(karen==false)
        {
                hobbi=false;
                break;
        }


        for(int i=1; i<(nom +1); i++)
        {                                       //check if the job can be fitted on any or all the machines
                if(path[i] <= SE[i][joblocat[i]+k])
                {
                        residle[i]=SE[i][joblocat[i]+k] - path[i];
                        fitsignal=1;
                }
        }       //in case the job has been fitted on a machine, check where it'll b most economical
        if(fitsignal == 1)
        {
                status2=1;
                petit=0;        //Locate the machine where the job can be processed with minimal cost
                for(int i=1; i<(nom +1); i++)
                {
                        if (residle[i] > petit)
                        {
                                petit = residle[i];
                                petitindex = i;
                        }
                }

                if(petitindex != machi)                 //Update the number of shifted jobs
                {
                        jobcount=jobcount+1;
                }
                                                                //Update the match-up time
                mathc=mathc+(SE[petitindex][joblocat[petitindex]+k] - RepF[machi]);

                lenumero[petitindex]=lenumero[petitindex] +1;
```

```
                                        //shifts the job on recipient machine
for(int j=joblocat[petitindex]+1; j < lenumero[petitindex] +1; j++)
{
        awal[petitindex][j]=awil[petitindex][j-1];
        ekher[petitindex][j]=ekhir[petitindex][j-1];
        ma7al[petitindex][j]=ma7il[petitindex][j-1];
}
                                        //Start updating the recipient machine
awal[petitindex][joblocat[petitindex]] = track[petitindex];
ekher[petitindex][joblocat[petitindex]]=track[petitindex]+ datap[petitindex][jindex];
ma7al[petitindex][joblocat[petitindex]] = jindex;
if(k > 0)
{                                       //update the shifted jobs required for fitting
        for(int j=joblocat[petitindex]; j <(joblocat[petitindex] +k);j++)
        {
                awal[petitindex][j +1] = ekher[petitindex][j];
                ekher[petitindex][j +1] = awal[petitindex][j +1]+
                                datap[petitindex][jplas[petitindex][j+1]];
        }
}

for(int j=1; j<(lenumero[petitindex] +1); j++)
{
        SE[petitindex][j] = awal[petitindex][j];
        FE[petitindex][j] = ekher[petitindex][j];
        jplas[petitindex][j] = ma7al[petitindex][j];
}                                       //Finished updating the recipient machine

lenumero[machi]=lenumero[machi] -1;     //Start updating the giver machine

for(int j=jobsp[machi]; j <(lenumero[machi] +1);j++)
{
        awal[machi][j] = SE[machi][j+1];
        ekher[machi][j] = FE[machi][j+1];
        ma7al[machi][j] = jplas[machi][j+1];
}

for(int j=1; j<(lenumero[machi] +1); j++)
{
        SE[machi][j] = awal[machi][j];
        FE[machi][j] = ekher[machi][j];
        jplas[machi][j] = ma7al[machi][j];
}                                       //Finished updating the giver machine

hobbi=false;
}

else                    //Need to Shift more jobs in order to fit the down job
{
        k=k+1;
```

```
            for(int i=1; i<(nom +1); i++)      //update the tracking variable "path"
            {
                    path[i]=path[i]+datap[i][jplas[i][joblocat[i]+k]];
            }
        }

}       //End of while loop

} //End of the function
```

This is the C++ Function that implements *PR* rule as standalone or implemented in the Robust System

```
void RepairRule5 (float locatO[],float findpO[], int machO,float SO[][500], float FO[][500], float
        RepFO[], int lenumeroO[],int jobspO[], double datapO[][500], int jplasO[][500], float&
        mathcO, int& jobcount, float finiO[])
{

float awal[10][500]={0}, ekher[10][500]={0},track[nom]={0},residle[nom]={0},
        ES[nom] = {0},LF[nom] = {0};
float petit=0, makespan=0,LatestS=0;
int states=0, joblocat[nom]={0},ma7al[10][500]={0}, ResJobs[noj]={0},jindex=0, Njob[noj]={0},
        c[nom]={0};
int JobsNo = 0;
bool jiji=true,karen=true,lello=true;
double SPANS[nom]={0}, Xjobs[10][500]={0},Xnew[10][500]={0},Xnewer[10][500]={0},
        ProcJobs[10][500]={0}, status=10,ESt[nom]={0},object=0,statu=8;

for(int i=1; i<(nom +1); i++)           //Use temporary jplas arrays so the original won't be modified
{
        for(int j=1;j<(lenumeroO[i]+1);j++)
        {
                awal[i][j]=SO[i][j];
                ekher[i][j]=FO[i][j];
                ma7al[i][j]=jplasO[i][j];
        }
}

for(int i=1; i<(nom +1); i++)                   //Get the jobs locations on each machine
{
        joblocat[i]=jobposup (track,lenumeroO,jobspO,i,locatO,findpO,datapO,jplasO);
        if(i == machO)                          //for the down machine, locate the down job
        {
                joblocat[i] = jobspO[i];
        }
}

for(int i=1; i<(nom +1); i++)                   //Get the ES on each machine
{
        ES[i]= track[i];
        if(i == machO)                  //for the down machine, ES is just after the repair
        {
                ES[i] = RepFO[i];
        }
        ESt[i-1]=double(ES[i]);                         //Keep a double array for Lingo
}
```

```
int matchIncrease=0;                        //This is used to increase the match-up when it's not enough

while (jiji)
{
        jiji=true;      //reinitialize jiji
        matchIncrease = matchIncrease + (Mincrease * 1);        //Increment the match-up
                        //Check if the match increase has exceeded the nb of jobs on any machine
        for (int i=1; i<(nom +1); i++)
        {
                if((matchIncrease + joblocat[i]- 1) >= lenumeroO[i])
                {
                        lello=false;
                }
        }

        if(lello==false)                                //Apply complete rescheduling
        {
                for(int j=1; j<(JobsNo +1);j++)         //Reinitialize the arrays
                {
                        for(int i=1; i<(nom +1);i++)
                        {
                                ProcJobs[i][j]=0;
                                Xjobs[i][j]=0;
                        }
                }
                JobsNo = 0;             //This is the number of jobs that need to be rescheduled

                for(int i=1;i<(nom +1); i ++)
                {
                        for(int j=joblocat[i];j<(lenumeroO[i] +1); j++)
                        {
                                JobsNo = JobsNo +1;                     //Increment nb of jobs
                                ResJobs[JobsNo]=jplasO[i][ j];          //these are the jobs located
                                                                        //after the breakdown
                                        Xjobs[i][JobsNo]=1;
                        }
                }

                for(int i =1; i<(nom+1);i++)                    //Get the processing time array
                {
                        for(int j=1; j<(JobsNo +1);j++)
                        {
                                ProcJobs[i][j] = datapO[i][ResJobs[j]];
                        }
                }
                LINGO3 (ProcJobs,Xjobs,Xnew,ESt,JobsNo,status,object);
                if(status==0)                           //LINGO3 found an optimal solution
                {
                        LINGO4 (ProcJobs,Xjobs,Xnewer,ESt,JobsNo,statu,object);
```

```
if(statu==0)                        //we were able to min nb of shifted jobs
{
        for(int i=1;i<nom +1;i++)
        {
                for(int j=1;j<JobsNo+1;j++)
                {
                        Xnew[i][j]=Xnewer[i][j];
                }
        }
}

for (int i=1;i<nom+1;i++)
{
        cout<<"the decisions on machine "<<i<<" are: "<<endl;
        for(int j=1;j<JobsNo+1;j++)
                cout<<" "<<Xnew[i][j];
}
jiji=false;

for (int i=1; i< (nom+1); i++)        //Update the new places of the jobs
{
        for (int j=1; j <(JobsNo+1); j++)
        {
                if((Xnew[i][j] - Xjobs[i][j])<0)          //Machine i lost the
                                                          //job (joblocat[i]+j-1)
                {
                        lenumeroO[i]=lenumeroO[i]-1;
                }
                if((Xnew[i][j] - Xjobs[i][j]) > 0)   //Machine won the job
                                                     //(ResJobs[j])
                {
                        lenumeroO[i]=lenumeroO[i]+1;
                        jobcount=jobcount+1;        //update the shifted jobs
                }
        }
}

for(int i=1;i<nom +1;i++)
{
        for(int j=1;j<JobsNo+1;j++)
        {
                if(Xnew[i][j]==1)
                {
                        Njob[i]=Njob[i]+1;
                        jplasO[i][joblocat[i]+Njob[i]-1]=ResJobs[j];
                        if(Njob[i]==1)
                        {
                                SO[i][joblocat[i]+Njob[i]-1]=ES[i];
                        }
```

```
                                        else
                                        {
                                                SO[i][joblocat[i]+Njob[i]-1] =
                                                        FO[i][joblocat[i]+Njob[i]-2];
                                        }
                                        FO[i][joblocat[i]+Njob[i]-1]=
                                                SO[i][joblocat[i]+Njob[i]-1] +ProcJobs[i][j];
                                }
                        }
                }
        }
        makespan=0;
        for(int i=1;i<(nom +1);i++)
        {
                if(FO[i][lenumeroO[i]] > makespan)
                {
                        makespan = FO[i][lenumeroO[i]];
                }
        }
        mathcO = mathcO + (makespan - RepFO[machO]);        //Match-up time required

}                                                       //End of Complete rescheduling

if(jiji==true)
{
        for (int i=1; i<(nom +1); i++)                  //calculate the span on each machine
        {
                LF[i] = SO[i][joblocat[i]+ matchIncrease];
                if(joblocat[i] >= lenumeroO[i])
                {
                        LF[i]=FO[i][lenumeroO[i]];
                }
                SPANS[i-1] = double(LF[i] - ES[i]);
        }

        JobsNo = 0;                     //This is the number of jobs that need to be rescheduled

        for(int i=1;i<(nom +1); i ++)
        {
                for(int j=1;j<(matchIncrease +1); j++)
                {
                        JobsNo = JobsNo +1;             //Increment nb of jobs
                        ResJobs[JobsNo]=jplasO[i][joblocat[i]+ j - 1];      //these r the jobs located
                                                                            //after the breakdown
                        Xjobs[i][JobsNo]=1;
                        cout<<"these r the "<<ResJobs[JobsNo]<<" ";
                }
        }
```

```
for(int j=1; j<(JobsNo +1);j++)                      //Get the processing time array
{
        for(int i=1; i<(nom +1);i++)
        {
                ProcJobs[i][j] = datapO[i][ResJobs[j]];
        }
}

LINGO2 (ProcJobs,Xjobs,Xnew,SPANS,JobsNo,status); //send info to LINGO to try to find
                                                     //a solution
if(status ==0)
{
        for (int i=1;i<nom+1;i++)
        {
                cout<<"the decisions on machine "<<i<<" are: "<<endl;
                for(int j=1;j<JobsNo+1;j++)
                        cout<<" "<<Xnew[i][j];
        }
        jiji=false;
        for (int i=1; i< (nom+1); i++)               //Update the new places of the jobs
        {
                for (int j=1; j <(JobsNo+1); j++)
                {                                    //Machine i lost the job (joblocat[i]+ j - 1)
                        if((Xnew[i][j] - Xjobs[i][j])<0)
                        {
                                lenumeroO[i]=lenumeroO[i]-1;
                                for(int k=joblocat[i]+ j - 1; k<(lenumeroO[i]+1); k++)
                                {
                                        jplasO[i][k]=jplasO[i][k+1];
                                }
                        }
                        if((Xnew[i][j] - Xjobs[i][j]) > 0)  //Machine won the job (ResJobs[j])
                        {
                                for(int k=joblocat[i]+ j - 1; k<(lenumeroO[i]+1); k++)
                                {
                                        ma7al[i][k+1]=jplasO[i][k];
                                        jobcount=jobcount+1;        //update the shifted jobs
                                }
                                for(int k=joblocat[i]+ j - 1; k<(lenumeroO[i]+1); k++)
                                {
                                        jplasO[i][k+1]=ma7al[i][k+1];
                                }
                                jplasO[i][joblocat[i]+ j - 1]= ResJobs[j];
                                lenumeroO[i]=lenumeroO[i]+1;
                                Njob[i]=Njob[i]+1;
                        }                                    //Machine kept the same job
                        if((Xnew[i][j] == Xjobs[i][j]) && (Xjobs[i][j] == 1))
                        {               //Increment the nb. of jobs assigned to this machine
                                Njob[i]=Njob[i]+1;
                        }
                }
```

```
        }
        for (int i=1; i< (nom+1); i++)                    //Get the Start on all machines
        {
                SO[i][joblocat[i]] = ES[i];
                FO[i][joblocat[i]] = SO[i][joblocat[i]] + datapO[i][jplasO[i][joblocat[i]]];
        }
        for (int i=1; i< (nom+1); i++)             //Update the start and finish of the jobs
        {
                for(int j=1; j <(Njob[i]); j++)
                {
                        SO[i][joblocat[i]+j]=FO[i][joblocat[i]+j-1];
                        FO[i][joblocat[i]+j]=SO[i][joblocat[i]+j] +
                                datapO[i][jplasO[i][joblocat[i]+j]];
                }
        }

        for(int i=1;i<(nom+1);i++)
        {
                if(FO[i][joblocat[i]+Njob[i]-1] - ES[i] > LatestS)
                {
                        LatestS = FO[i][joblocat[i]+Njob[i]-1] - ES[i];
                }
        }
        mathcO=mathcO + (LatestS);                     //update the match-up time
    }   // End of If

    else
    {
        for(int j=1; j<(JobsNo +1);j++)                //Reinitialize the arrays
        {
                for(int i=1; i<(nom +1);i++)
                {
                        ProcJobs[i][j]=0;
                        Xjobs[i][j]=0;
                }
        }
        jiji=true;
    }
} //End of IF

} //End of while

} //End of function
```

This function will generate an optimal reschedule by interfacing with LINGO2.Lng (MIP[2])
in order to match with the initial schedule, with objective of minimizing # of jobs
that will be shifted to other machines

```
void LINGO2 (double processing[][500],double Xold[10][500],double Xnews[10][500],
        double SPAN[],int JobNo,double& stat)
{
char cScript[256];                                              //LINGO interface
double dObjective, dStatus=-1.;
double dnom[] = {nom};
double JobsNo[]={0};
double dX[1000]={0};
int nError=-1, nPointersNow;
int index = 0,nM=1;

// create the LINGO environment object
        pLSenvLINGO pLINGO;
  pLINGO = LScreateEnvLng();
  if ( !pLINGO)
  {
    printf( "Can"t create LINGO environment!\n");
    goto FinalExit;
  }

  // Open LINGO's log file
  nError = LSopenLogFileLng( pLINGO, "LINGO2.log");
  if ( nError) goto ErrorExit;

  // Pass memory transfer pointers to LINGO

  // @POINTER(1)
  JobsNo[0]=(double)JobNo;   //Assign the nb of jobs
  nError = LSsetPointerLng( pLINGO, JobsNo, &nPointersNow);
  if ( nError) goto ErrorExit;

  // @POINTER(2)
  nError = LSsetPointerLng( pLINGO, dnom, &nPointersNow);
  if ( nError) goto ErrorExit;

  // @POINTER(3)
  double datas3[1000];
    for ( int i = 0; i<(nom); i++)       //Transfer the "processing" double array to "datas3" single array
      for ( int j = 0; j < (JobNo); j++)
        datas3[ i * JobNo + j] = processing[i+1][j+1];
          nError = LSsetPointerLng( pLINGO, datas3, &nPointersNow);
          if ( nError) goto ErrorExit;
```

```
// @POINTER(4)
double datas4[1000];
for ( int i = 0; i<(nom); i++)        //Transfer the "Xold" double array to "datas4" single array
   for ( int j = 0; j < (JobNo); j++)
     datas4[ i * JobNo + j] = Xold[i+1][j+1];
nError = LSsetPointerLng( pLINGO, datas4, &nPointersNow);
if ( nError) goto ErrorExit;

// @POINTER(5)
nError = LSsetPointerLng( pLINGO, SPAN, &nPointersNow);
if ( nError) goto ErrorExit;

// @POINTER(6)
nError = LSsetPointerLng( pLINGO, &dObjective, &nPointersNow);
if ( nError) goto ErrorExit;

// @POINTER(7)
nError = LSsetPointerLng( pLINGO, &dStatus, &nPointersNow);
if ( nError) goto ErrorExit;

// @POINTER(8)
       nError = LSsetPointerLng( pLINGO, dX, &nPointersNow);
       if ( nError) goto ErrorExit;

// Here is the script we want LINGO to run
strcpy( cScript, "SET ECHOIN 1 \n TAKE LINGO2.Lng \n GO \n QUIT \n");

// Run the script
nError = LSexecuteScriptLng( pLINGO, cScript);
if ( nError) goto ErrorExit;

// Close the log file
LScloseLogFileLng( pLINGO);

// Any problems?
if ( nError || dStatus != LS_STATUS_GLOBAL_LNG)
{
  // Had a problem
  printf( "Unable to solve!");
}
stat=dStatus;

// Output the decision variables
for ( int i = 1; i<(nom+1); i++)
{
   for ( int j = 1; j < (JobNo+1); j++)
              {
                     Xnews[i][j]=dX[index];
                     index++;
              }
}
```

```
  for (int i=1;i<(nom+1);i++)
  {
          cout<<"the decisions on machine "<<i<<" are: "<<endl;
          for(int j=1;j<(JobNo+1);j++)
                  cout<<" "<<Xnews[i][j];
  }
          cout<<"the objective is "<<dObjective<<" and status is "<<dStatus<<endl;
                goto NormalExit;
ErrorExit:
  printf("LINGO Error Code: %d\n", nError);

NormalExit:
  LSdeleteEnvLng( pLINGO);

FinalExit: ;

}
```

This function will generate an optimal reschedule by interfacing with LINGO3.Lng (MIP[3]) with the objective of minimizing $Cmax_R$

```
void LINGO3 (double processingK[][500],double XoldK[10][500],double XnewsK[10][500],
        double ESK[],int JobNoK,double& statK,double& dobj)
{
char cScript[256];                                          //LINGO interface
double dObjective, dStatus=-1.;
double dnom[] = {nom};
double JobsNo[]={0};
double dX[1000]={0};
int nError=-1, nPointersNow;
int index = 0,nM=1;

// create the LINGO environment object
        pLSenvLINGO pLINGO;
  pLINGO = LScreateEnvLng();
  if ( !pLINGO)
  {
    printf( "Can"t create LINGO environment!\n");
    goto FinalExit;
  }

  // Open LINGO's log file
  nError = LSopenLogFileLng( pLINGO, "LINGO3.log");
  if ( nError) goto ErrorExit;

  // Pass memory transfer pointers to LINGO

  // @POINTER(1)
  JobsNo[0]=(double)JobNoK;   //Assign the nb of jobs
  nError = LSsetPointerLng( pLINGO, JobsNo, &nPointersNow);
  if ( nError) goto ErrorExit;

  // @POINTER(2)
  nError = LSsetPointerLng( pLINGO, dnom, &nPointersNow);
  if ( nError) goto ErrorExit;

  // @POINTER(3)
  double datas3[1000];

    for ( int i = 0; i<(nom); i++)      //Transfer the "processing" double array to "datas3" single array
      for ( int j = 0; j < (JobNoK); j++)
        datas3[ i * JobNoK + j] = processingK[i+1][j+1];
          nError = LSsetPointerLng( pLINGO, datas3, &nPointersNow);
          if ( nError) goto ErrorExit;
```

```
// @POINTER(4)
double datas4[1000];
for ( int i = 0; i<(nom); i++)        //Transfer the "Xold" double array to "datas4" single array
   for ( int j = 0; j < (JobNoK); j++)
      datas4[ i * JobNoK + j] = XoldK[i+1][j+1];
nError = LSsetPointerLng( pLINGO, datas4, &nPointersNow);
if ( nError) goto ErrorExit;

// @POINTER(5)
nError = LSsetPointerLng( pLINGO, ESK, &nPointersNow);
if ( nError) goto ErrorExit;

// @POINTER(6)
nError = LSsetPointerLng( pLINGO, &dObjective, &nPointersNow);
if ( nError) goto ErrorExit;

// @POINTER(7)
nError = LSsetPointerLng( pLINGO, &dStatus, &nPointersNow);
if ( nError) goto ErrorExit;

// @POINTER(8)
      nError = LSsetPointerLng( pLINGO, dX, &nPointersNow);
      if ( nError) goto ErrorExit;

// Here is the script we want LINGO to run
strcpy( cScript, "SET ECHOIN 1 \n TAKE LINGO3.Lng \n GO \n QUIT \n");

// Run the script
nError = LSexecuteScriptLng( pLINGO, cScript);
if ( nError) goto ErrorExit;

// Close the log file
LScloseLogFileLng( pLINGO);

// Any problems?
if ( nError || dStatus != LS_STATUS_GLOBAL_LNG)
{
  // Had a problem
  printf( "Unable to solve!");
}
statK=dStatus;

// Output the decision variables
for ( int i = 1; i<(nom+1); i++)
{
   for ( int j = 1; j < (JobNoK+1); j++)
            {
                    XnewsK[i][j]=dX[index];
                    index++;
            }
}
```

```
  for (int i=1;i<(nom+1);i++)
  {
          cout<<"the decisions on machine "<<i<<" are: "<<endl;
          for(int j=1;j<(JobNoK+1);j++)
                  cout<<" "<<XnewsK[i][j];
  }
          cout<<"the objective is "<<dObjective<<" and status is "<<dStatus<<endl;
          dobj=dObjective;
                goto NormalExit;
ErrorExit:
  printf("LINGO Error Code: %d\n", nError);

NormalExit:
  LSdeleteEnvLng( pLINGO);

FinalExit: ;

}
```

This function will generate an optimal reschedule by interfacing with LINGO4.Lng (MIP[4])with the objective of minimizing the number of shifted jobs while $Cmax_R$ is constrained to be at its optimum (the value obtained using LINGO3.Lng (MIP[3]))

```
void LINGO4 (double processingZ[][500],double XoldZ[10][500],double XnewsZ[10][500],
        double ESZ[],int JobNoZ,double& statZ,double dobjZ)
{
char cScript[256];                                                    //LINGO interface
double dObjective, dStatus=-1.;
double dnom[] = {nom};
double JobsNo[]={0},Objecta[]={0};
double dX[1000]={0};
int nError=-1, nPointersNow;
int index = 0,nM=1;

// create the LINGO environment object
        pLSenvLINGO pLINGO;
  pLINGO = LScreateEnvLng();
  if ( !pLINGO)
  {
    printf( "Can"t create LINGO environment!\n");
    goto FinalExit;
  }

  // Open LINGO's log file
  nError = LSopenLogFileLng( pLINGO, "LINGO4.log");
  if ( nError) goto ErrorExit;

  // Pass memory transfer pointers to LINGO

  // @POINTER(1)
  JobsNo[0]=(double)JobNoZ;    //Assign the nb of jobs
  nError = LSsetPointerLng( pLINGO, JobsNo, &nPointersNow);
  if ( nError) goto ErrorExit;

  // @POINTER(2)
  nError = LSsetPointerLng( pLINGO, dnom, &nPointersNow);
  if ( nError) goto ErrorExit;

  // @POINTER(3)
  double datas3[1000];

    for ( int i = 0; i<(nom); i++)      //Transfer the "processing" double array to "datas3" single array
      for ( int j = 0; j < (JobNoZ); j++)
        datas3[ i * JobNoZ + j] = processingZ[i+1][j+1];
          nError = LSsetPointerLng( pLINGO, datas3, &nPointersNow);
          if ( nError) goto ErrorExit;
```

```
// @POINTER(4)
double datas4[1000];
for ( int i = 0; i<(nom); i++)        //Transfer the "Xold" double array to "datas4" single array
   for ( int j = 0; j < (JobNoZ); j++)
      datas4[ i * JobNoZ + j] = XoldZ[i+1][j+1];
nError = LSsetPointerLng( pLINGO, datas4, &nPointersNow);
if ( nError) goto ErrorExit;

// @POINTER(5)
nError = LSsetPointerLng( pLINGO, ESZ, &nPointersNow);
if ( nError) goto ErrorExit;

// @POINTER(6)
Objecta[0]=dobjZ;
nError = LSsetPointerLng( pLINGO, Objecta, &nPointersNow);
if ( nError) goto ErrorExit;

// @POINTER(7)
nError = LSsetPointerLng( pLINGO, &dStatus, &nPointersNow);
if ( nError) goto ErrorExit;

// @POINTER(8)
      nError = LSsetPointerLng( pLINGO, dX, &nPointersNow);
      if ( nError) goto ErrorExit;

// Here is the script we want LINGO to run
strcpy( cScript, "SET ECHOIN 1 \n TAKE LINGO4.Lng \n GO \n QUIT \n");

// Run the script
nError = LSexecuteScriptLng( pLINGO, cScript);
if ( nError) goto ErrorExit;

// Close the log file
LScloseLogFileLng( pLINGO);

// Any problems?
if ( nError || dStatus != LS_STATUS_GLOBAL_LNG)
{              // Had a problem
  printf( "Unable to solve!");
}
statZ=dStatus;

// Output the decision variables
for ( int i = 1; i<(nom+1); i++)
{
   for ( int j = 1; j < (JobNoZ+1); j++)
            {
                    XnewsZ[i][j]=dX[index];
                    index++;
            }
 }
```

```
  for (int i=1;i<(nom+1);i++)
  {
          cout<<"the decisions on machine "<<i<<" are: "<<endl;
          for(int j=1;j<(JobNoZ+1);j++)
                  cout<<" "<<XnewsZ[i][j];
  }
          cout<<"status is "<<dStatus<<endl;
                goto NormalExit;
ErrorExit:
  printf("LINGO Error Code: %d\n", nError);

NormalExit:
  LSdeleteEnvLng( pLINGO);

FinalExit: ;

}
```

APPENDIX B: RSR IMPLEMENTATION CODE IN VISUAL C++

The main function used for Robust System (Appendix A) can be used for the *RSR* implementation (after deleting the unnecessary code lines; for example, the average usage of the rules)

inputdata, LINGO1, sort, jobposit, jobposup, and assign functions are described in Appendix A. The only change needed is for the assign function where only *RSR* should be applied.

The *RSR* rule function is shown below

```
void RepairRule1 (int mach,float SI[][500], float FI[][500], float ReF[], int thenumero[],int jobpi[],
        double datam[][500], int joplaces[][500], float& matchc, int& nschedule, float finisia[])
{

float awal[10][500]={0}, ekher[10][500]={0},petit=0,finitio[10][500]={0};
int index=0,matchsignal=0;

for(int i=1; i<(nom +1); i++)    //Use temporary S and F arrays so the original won't b modified
{
        for(int j=1;j<(thenumero[i]+1);j++)
        {
                awal[i][j]=SI[i][j];
                ekher[i][j]=FI[i][j];
                finitio[i][j]=FI[i][j];
        }
}

int k = 0, lecmax=0;
awal[mach][jobpi[mach]]=ReF[mach];
ekher[mach][jobpi[mach]]= awal[mach][jobpi[mach]] + datam[mach][joplaces[mach][jobpi[mach]]];

while((ekher[mach][jobpi[mach] + k] > awal[mach][(jobpi[mach] +k +1)]) && ((jobpi[mach]+k)
        <thenumero[mach]))
{
        if((jobpi[mach] + k+1) == thenumero[mach]) //if the next job is the last, we need to stop
        {
                awal[mach][(jobpi[mach] +k +1)] = ekher[mach][jobpi[mach] + k];
                ekher[mach][jobpi[mach] + k +1] = awal[mach][(jobpi[mach] +k +1)] +
                                        datam[mach][joplaces[mach][jobpi[mach] + k+1]];
                finisia[mach] = ekher[mach][jobpi[mach] + k +1];  //update the finish time of the
                                                                    //machine
                matchsignal=1;
                nschedule=1;
                break;
        }
        awal[mach][(jobpi[mach] +k +1)] = ekher[mach][jobpi[mach] + k];
        ekher[mach][jobpi[mach] + k +1] = awal[mach][(jobpi[mach] +k +1)] +
                                        datam[mach][joplaces[mach][jobpi[mach] + k+1]];
        k = k +1;
}
```

```
if(jobpi[mach]+k == thenumero[mach])                    //If this is the last job
{
        finisia[mach] = ekher[mach][jobpi[mach] + k];
}

for(int i=1; i<(nom +1); i++)                 //Reupdate the start and finish of the jobs
{
        for(int j=1;j<(thenumero[i]+1);j++)
        {
                SI[i][j] =awal[i][j];
                FI[i][j]= ekher[i][j];
        }
}

for(int i=1;i<(nom +1); i++)                          //get the makespan
{
        if(FI[i][thenumero[i]] > lecmax)
        {
                lecmax=FI[i][thenumero[i]];
        }
}

if(nschedule == 1)                            //in the case of the last job
{
        matchc = matchc + (lecmax - ReF[mach]);
        cout<<"match current "<<matchc<<" because of "<<lecmax - ReF[mach]<<endl;
}
else            //if not last job
{
        matchc = matchc + (awal[mach][jobpi[mach] + k+1] - ReF[mach]);
        cout<<"match current "<<matchc<<" because of "<<ekher[mach][jobpi[mach] + k] -
                                                                ReF[mach]<<endl;
}
} //End of function
```

APPENDIX C: FJR IMPLEMENTATION CODE IN VISUAL C++

The main function used for Robust System (Appendix A) can be used for the *FJR* implementation (after deleting the unnecessary code lines; for example, the average usage of the rules)

inputdata, LINGO1, sort, jobposit, jobposup, and assign functions are described in Appendix A. The only change needed is for the assign function where only *FJR* should be applied.

The *FJR* rule function is shown below

```
void RepairRule2 (float locati[],float findposi[], int machi,float SE[][500], float FE[][500], float
        RepF[], int lenumero[],int jobsp[], double datap[][500], int jplas[][500], float& mathc, int&
        jobcount, float fini[])
{

float awal[10][500]={0}, ekher[10][500]={0},awil[10][500]={0},
        ekhir[10][500]={0},track[nom]={0},path[nom]={0},wpath[nom]={0},residle[nom]={0},
        compi[nom]={0};
int joblocat[nom]={0}, ma7al[10][500]={0}, ma7il[10][500]={0};
int fitsignal=0,k=0,states=0,petitindex=0,jindex=0;
bool hobbi=true, karen=false;
float petit=0,makespani=0;

for(int i=1; i<(nom +1); i++)    // Use temporary S and F arrays so the original job-machine
                                 //assignment won't be modified
{
        for(int j=1;j<(lenumero[i]+1);j++)
        {
                awal[i][j]=SE[i][j];
                awil[i][j]=SE[i][j];
                ekher[i][j]=FE[i][j];
                ekhir[i][j]=FE[i][j];
                ma7al[i][j]=jplas[i][j];
                ma7il[i][j]=jplas[i][j];
        }
}

for(int i=1; i<(nom +1); i++)    //Get the jobs "on the right" locations on each machine
{
        joblocat[i]=jobposup (track,lenumero,jobsp,i,locati,findposi,datap,jplas);
        if(i == machi)            //for the down machine, locate the job after the down job
        {
                joblocat[i] = jobsp[i]+1;
                track[i]=RepF[i];  //Because it can only start once the repair finishes
        }
}

jindex=jplas[machi][jobsp[machi]];
```

```
for(int i=1; i<(nom +1); i++)     //assume the down job will be fitted on each machine to determine
                                  //which one is more appropriate
{
        path[i]=track[i]+datap[i][jindex];
        wpath[i]=path[i];         //Use it in case we can not fit the job on any machine
}

while(hobbi)
{
        karen=false;
        for(int i=1;i<(nom +1); i++)   //Check if we still have jobs to shift in order to fit the down job
        {
                if((lenumero[i]) >= (joblocat[i]+k))
                {
                        karen=true;
                }
                else
                {
                        path[i]=1000000;  //assigned a large number so this path is not chosen
                }
        }
        if(karen==false)
        {
                hobbi=false;
                break;
        }

        for(int i=1; i<(nom +1); i++)
        {
                if(path[i] <= SE[i][joblocat[i]+k])    //check if the job can be fitted on any or all the
                                                        //machines
                {
                        residle[i]=SE[i][joblocat[i]+k] - path[i];
                        fitsignal=1;
                }
        }
        if(fitsignal == 1)   //in case the job has been fitted on a machine, check where it'll be more
                             //economical
        {
                petit=0;
                for(int i=1; i<(nom +1); i++)        //Locate the machine where the job can be
                                                     //processed with minimal cost
                {
                        if (residle[i] > petit)
                        {
                                petit = residle[i];
                                petitindex = i;
                        }
                }
```

```
if(petitindex != machi)     //Update the number of shifted jobs
{
        jobcount=jobcount+1;
}
                                                    //Update the matchup time
mathc=mathc+(SE[petitindex][joblocat[petitindex]+k] - RepF[machi]);


lenumero[petitindex]=lenumero[petitindex] +1;

for(int j=joblocat[petitindex]+1; j < lenumero[petitindex] +1; j++)  //shift the jobs on
                                                                    //recipient machine
{
        awal[petitindex][j]=awil[petitindex][j-1];
        ekher[petitindex][j]=ekhir[petitindex][j-1];
        ma7al[petitindex][j]=ma7il[petitindex][j-1];
}
                                        //Start updating the recipient machine
awal[petitindex][joblocat[petitindex]] = track[petitindex];
ekher[petitindex][joblocat[petitindex]] = track[petitindex] +datap[petitindex][jindex];
ma7al[petitindex][joblocat[petitindex]] = jindex;
if(k > 0)
{                                       //update the shifted jobs required for fitting
        for(int j=joblocat[petitindex]; j <(joblocat[petitindex] +k);j++)
        {
                awal[petitindex][j +1] = ekher[petitindex][j];
                ekher[petitindex][j +1] = awal[petitindex][j +1]+
                        datap[petitindex][jplas[petitindex][j+1]];
        }
}

for(int j=1; j<(lenumero[petitindex] +1); j++)
{
        SE[petitindex][j] = awal[petitindex][j];
        FE[petitindex][j] = ekher[petitindex][j];
        jplas[petitindex][j] = ma7al[petitindex][j];
}                                       //Finished updating the recipient machine

lenumero[machi]=lenumero[machi] -1;     //Start updating the giver machine

for(int j=jobsp[machi]; j <(lenumero[machi] +1);j++)
{
        awal[machi][j] = SE[machi][j+1];
        ekher[machi][j] = FE[machi][j+1];
        ma7al[machi][j] = jplas[machi][j+1];
}

for(int j=1; j<(lenumero[machi] +1); j++)
{
        SE[machi][j] = awal[machi][j];
        FE[machi][j] = ekher[machi][j];
```

```
                jplas[machi][j] = ma7al[machi][j];
            }                                   //Finished updating the giver machine

            hobbi=false;
        }
        else                        //Need to Shift more jobs in order to fit the down job
        {
            k=k+1;
            for(int i=1; i<(nom +1); i++)           //update the tracking variable "path"
            {
                path[i]=path[i]+datap[i][jplas[i][joblocat[i]+k]];
            }
        }

    }       //End of while loop

    if(karen == false)    //i.e. we ran out of jobs and couldn't fit the down job on any machine
    {               //In this case, we will just fit it to the machine with the smallest path
        petit=1000000000;
        petitindex=0;

        for(int i=1; i<(nom +1); i++)     //Use temporary S and F arrays so the original won't be
                                          //modified
        {
            for(int j=1;j<(lenumero[i]+1);j++)
            {
                awal[i][j]=SE[i][j];
                ekher[i][j]=FE[i][j];
                ma7al[i][j]=jplas[i][j];
            }
        }
        for(int i=1; i<(nom +1); i++)              //update the tracking variable "wpath"
        {
            for(int j=joblocat[i];j<(lenumero[i]+1);j++)
            {
                wpath[i]=wpath[i]+datap[i][jplas[i][j]];
            }
        }
                        //Locate the machine where the job can be processed with minimal cost
        for(int i=1; i<(nom +1); i++)
        {
            if (wpath[i] < petit)
            {
                petit = wpath[i];
                petitindex = i;
                cout<<"chosen machine "<<petitindex<<endl;
            }
        }
```

```
if(petitindex != machi)                               //Update the number of shifted jobs
{
        jobcount=jobcount+1;
}


lenumero[petitindex]=lenumero[petitindex] +1;        //Start updating the recipient machine
                                                      //shifts the job on recipient machine
for(int j=lenumero[petitindex]; j > joblocat[petitindex]; j--)
{
        ma7al[petitindex][j]=ma7al[petitindex][j-1];
}

awal[petitindex][joblocat[petitindex]] = track[petitindex];
ekher[petitindex][joblocat[petitindex]] = track[petitindex] + datap[petitindex][jindex];
ma7al[petitindex][joblocat[petitindex]] = jindex;

for(int j=joblocat[petitindex];j<lenumero[petitindex];j++)
{
        awal[petitindex][j+1]=ekher[petitindex][j];
        ekher[petitindex][j+1]=awal[petitindex][j+1] +
                datap[petitindex][ma7al[petitindex][j+1]];
}

for(int j=1; j<(lenumero[petitindex] +1); j++)
{
        SE[petitindex][j] = awal[petitindex][j];
        FE[petitindex][j] = ekher[petitindex][j];
        jplas[petitindex][j] = ma7al[petitindex][j];
}                                                     //Finished updating the recipient machine

lenumero[machi]=lenumero[machi] -1;                   //Start updating the giver machine

ma7al[machi][jobsp[machi]] = jplas[machi][jobsp[machi]+1];
awal[machi][jobsp[machi]] = track[machi];
ekher[machi][jobsp[machi]] = awal[machi][jobsp[machi]]
        +datap[machi][ma7al[machi][jobsp[machi]]];

for(int j=jobsp[machi]+1; j <(lenumero[machi] +1);j++)
{
        ma7al[machi][j] = jplas[machi][j+1];
        awal[machi][j] = ekher[machi][j-1];
        ekher[machi][j] = awal[machi][j] +datap[machi][ma7al[machi][j]];
}
for(int j=1; j<(lenumero[machi] +1); j++)
{
        SE[machi][j] = awal[machi][j];
        FE[machi][j] = ekher[machi][j];
        jplas[machi][j] = ma7al[machi][j];
}                                                     //Finished updating the giver machine
```

```
makespani=0;

for(int i=1;i<(nom +1);i++)                //Get the new makespan
{
        if(FE[i][lenumero[i]] > makespani)
        {
                makespani = FE[i][lenumero[i]];
        }
}

mathc=mathc+(makespani - RepF[machi]);  //Update the matchup time
cout<<"the new makespan is "<<makespani<<endl;

} //End of If

}  //End of the function
```

APPENDIX D: CR IMPLEMENTATION CODE IN VISUAL C++

The main function used for Robust System (Appendix A) can be used for the *CR* implementation (after deleting the unnecessary code lines; for example, the average usage of the rules)

inputdata, LINGO1, LINGO2, LINGO3, LINGO4, sort, jobposit, jobposup, and assign functions are described in Appendix A. The only change needed is for the assign function where only *CR* should be applied.

The *CR* rule function is shown below.

```
void RepairRule5 (float locatO[],float findpO[], int machO,float SO[][500], float FO[][500], float
        RepFO[], int lenumeroO[],int jobspO[], double datapO[][500], int jplasO[][500], float&
        mathcO, int& jobcount, float finiO[])
{

float awal[10][500]={0}, ekher[10][500]={0},track[nom]={0},residle[nom]={0},ES[nom] = {0},
        LF[nom] = {0};
float petit=0, makespan=0,LatestS=0;
int states=0, joblocat[nom]={0}, ma7al[10][500]={0},ResJobs[noj]={0},jindex=0,Njob[noj]={0},
        c[nom]={0};
int JobsNo = 0;
bool jiji=true,karen=true,lello=true;
double SPANS[nom]={0}, Xjobs[10][500]={0},Xnew[10][500]={0},Xnewer[10][500]={0},
        ProcJobs[10][500]={0},status=10,ESt[nom]={0},object=0,statu=8;

for(int i=1; i<(nom +1); i++)            //Use temporary jplas arrays so the original won't b modified
{
        for(int j=1;j<(lenumeroO[i]+1);j++)
        {
                awal[i][j]=SO[i][j];
                ekher[i][j]=FO[i][j];
                ma7al[i][j]=jplasO[i][j];
        }
}

for(int i=1; i<(nom +1); i++)                          //Get the jobs locations on each machine
{
        joblocat[i]=jobposup (track,lenumeroO,jobspO,i,locatO,findpO,datapO,jplasO);
        if(i == machO)                                 //for the down machine, locate the down job
        {
                joblocat[i] = jobspO[i];
        }
}

for(int i=1; i<(nom +1); i++)                                  //Get the ES on each machine
{
        ES[i]= track[i];
```

```
        if(i == machO)                              //for the down machine, ES is just after the repair
        {
                ES[i] = RepFO[i];
        }
        ESt[i-1]=double(ES[i]);                                 //Keep a double array for Lingo
}

int matchIncrease=0;                     //This is used to increase the match-up when it's not enough

for(int i=1;i<(nom +1); i ++)
{
        for(int j=joblocat[i];j<(lenumeroO[i] +1); j++)
        {
                JobsNo = JobsNo +1;                                     //Increment nb of jobs
                ResJobs[JobsNo]=jplasO[i][ j];          //these r the jobs located after the breakdown
                Xjobs[i][JobsNo]=1;
        }
}

for(int i =1; i<(nom+1);i++)                                    //Get the processing time array
{
        for(int j=1; j<(JobsNo +1);j++)
        {
                ProcJobs[i][j] = datapO[i][ResJobs[j]];
        }
}
LINGO3 (ProcJobs,Xjobs,Xnew,ESt,JobsNo,status,object);
if(status==0)                                                   //LINGO3 found an optimal solution
{
        LINGO4 (ProcJobs,Xjobs,Xnewer,ESt,JobsNo,statu,object);
        if(statu==0)                                    //we were able to min nb of shifted jobs
        {
                for(int i=1;i<nom +1;i++)
                {
                        for(int j=1;j<JobsNo+1;j++)
                        {
                                Xnew[i][j]=Xnewer[i][j];
                        }
                }
        }

        for (int i=1;i<nom+1;i++)
        {
                cout<<"the decisions on machine "<<i<<" are: "<<endl;
                for(int j=1;j<JobsNo+1;j++)
                        cout<<" "<<Xnew[i][j];
        }
        jiji=false;
```

```
        for (int i=1; i< (nom+1); i++)                          //Update the new places of the jobs
        {
                for (int j=1; j <(JobsNo+1); j++)
                {
                        if((Xnew[i][j] - Xjobs[i][j])<0)     //Machine i lost the job (joblocat[i] + j - 1)
                        {
                                lenumeroO[i]=lenumeroO[i]-1;
                        }

                        if((Xnew[i][j] - Xjobs[i][j]) > 0)          //Machine won the job (ResJobs[j])
                        {
                                lenumeroO[i]=lenumeroO[i]+1;
                                jobcount=jobcount+1;                        //update the shifted jobs
                        }
                }
        }

        for(int i=1;i<nom +1;i++)
        {
                for(int j=1;j<JobsNo+1;j++)
                {
                        if(Xnew[i][j]==1)
                        {
                                Njob[i]=Njob[i]+1;
                                jplasO[i][joblocat[i]+Njob[i]-1]=ResJobs[j];
                                if(Njob[i]==1)
                                {
                                        SO[i][joblocat[i]+Njob[i]-1]=ES[i];
                                }
                                else
                                {
                                        SO[i][joblocat[i]+Njob[i]-1]=FO[i][joblocat[i]+Njob[i]-2];
                                }
                                FO[i][joblocat[i]+Njob[i]-1]=SO[i][joblocat[i]+Njob[i]-1] +
                                                                                ProcJobs[i][j];
                        }
                }
        }
}

for(int i=1;i<(nom +1);i++)
{
        if(FO[i][lenumeroO[i]] > makespan)
        {
                makespan = FO[i][lenumeroO[i]];
        }
}
mathcO = mathcO + (makespan - ES[machO]);   //Match-up time required

}   //End of Complete rescheduling
```

APPENDIX E: LINGO MODELS

This is LINGO1.Lng where MIP[1] is implemented. The objective function is to minimize the makespan in order to attain optimal initial schedules for the unrelated parallel machine problem

```
MODEL:

DATA:
N_O_J=@pointer( 1);
N_O_M=@pointer(2);

ENDDATA

SETS:
JOBS/1..N_O_J/;                               !DEFINE NUMBER OF JOBS;
MACHINES/1.. N_O_M/;                          !DEFINE NUMBER OF MACHINES;
LINKS(MACHINES,JOBS):PROCESSING,XI;
ENDSETS


[robj] MIN=C;  !OBJECTIVE FUNCTION;

@FOR(JOBS(J):                                 !FIRST CONSTRAINT;
      @SUM(MACHINES(I):(XI(I,J)))=1);

@FOR(MACHINES(I):                             !SECOND CONSTRAINT;
      @SUM(JOBS(J):XI(I,J)* PROCESSING(I,J)) < C);

@FOR (LINKS(MACHINES,JOBS):@BIN(XI));         !THIS FUNCTION WILL MAKE THE
                                              DECISION VARIABLES BINARY;



data:
  PROCESSING=@pointer(3);
  @pointer(4) = rObj;
  @pointer(5) = @status();
  @pointer (6) = XI;

enddata

END
```

This is LINGO2.Lng where MIP[2] is implemented. The objective function is to minimize the number of shifted jobs at a minimal match-up time in the *PR* rule

```
MODEL:

DATA:
N_O_J=@pointer( 1);
N_O_M=@pointer(2);

ENDDATA

SETS:
JOBS/1..N_O_J/;                              !DEFINE NUMBER OF JOBS;
MACHINES/1.. N_O_M/: SPAN;                   !DEFINE NUMBER OF MACHINES;
LINKS(MACHINES,JOBS):PROCESSING,XI,XO,Y;
ENDSETS

[robj] MIN=(@SUM(LINKS(I,J): Y(I,J)));       !OBJECTIVE FUNCTION;

@FOR(JOBS(J):            !Ensure that every job will be assigned to only 1 machine;
      @SUM(MACHINES(I):(XI(I,J)))=1);

@FOR(MACHINES(I):                            !SECOND CONSTRAINT;
      @SUM(JOBS(J):XI(I,J)* PROCESSING(I,J)) < SPAN(I));

@FOR (LINKS(I,J):
      XO(I,J) - XI(I,J)-Y(I,J) < 0);   !Constraint 1 for absolute value;

@FOR (LINKS(I,J):
      -XO(I,J) + XI(I,J)-Y(I,J) < 0);   !Constraint 2 for absolute value;

@FOR (LINKS(MACHINES,JOBS):@BIN(XI));        !THIS FUNCTION WILL MAKE THE
                                             DECISION VARIABLES BINARY;


data:
  PROCESSING=@pointer(3);
  XO=@pointer(4);
  SPAN=@pointer(5);
  @pointer(6) = rObj;
  @pointer(7) = @status();
  @pointer (8) = XI;

enddata

END
```

This is LINGO3.Lng where MIP[3] is implemented. The objective function is to minimize the makespan in *CR* rule

```
MODEL:

DATA:
N_O_J=@pointer( 1);
N_O_M=@pointer(2);

ENDDATA

SETS:
JOBS/1..N_O_J/;                                  !DEFINE NUMBER OF JOBS;
MACHINES/1.. N_O_M/: ES;                         !DEFINE NUMBER OF MACHINES;
LINKS(MACHINES,JOBS):PROCESSING,XI,XO,Y;
ENDSETS

[robj] MIN=C;                                    !OBJECTIVE FUNCTION;

@FOR(JOBS(J):               !Ensure that every job will be assigned to only 1 machine;
        @SUM(MACHINES(I):(XI(I,J)))=1);

@FOR(MACHINES(I):                                !SECOND CONSTRAINT;
        @SUM(JOBS(J):(XI(I,J)* PROCESSING(I,J))) +ES(I) < C );

@FOR (LINKS(MACHINES,JOBS):@BIN(XI));            !THIS FUNCTION WILL MAKE THE
                                                 DECISION VARIABLES BINARY;


data:
  PROCESSING=@pointer(3);
  XO=@pointer(4);
  ES=@pointer(5);
  @pointer(6) = rObj;
  @pointer(7) = @status();
  @pointer (8) = XI;

enddata

END
```

This is LINGO4.Lng where MIP[4] is implemented. The objective function is to minimize the number of shifted jobs, while the makespan is constrained to be at its optimum, i.e. the value obtained from MIP[3]

```
MODEL:

DATA:
N_O_J=@pointer( 1);
N_O_M=@pointer(2);

ENDDATA

SETS:
JOBS/1..N_O_J/;                              !DEFINE NUMBER OF JOBS;
MACHINES/1.. N_O_M/: ES;                     !DEFINE NUMBER OF MACHINES;
LINKS(MACHINES,JOBS):PROCESSING,XI,XO,Y;
ENDSETS

[robj] MIN=(@SUM(LINKS(I,J): Y(I,J)));       !OBJECTIVE FUNCTION;

@FOR(JOBS(J):                  !Ensure that every job will be assigned to only 1 machine;
        @SUM(MACHINES(I):(XI(I,J)))=1);

@FOR(MACHINES(I):                            !SECOND CONSTRAINT;
        @SUM(JOBS(J):(XI(I,J)* PROCESSING(I,J))+ES(I)) < jiji );

@FOR (LINKS(I,J):
        XO(I,J) - XI(I,J)-Y(I,J) < 0);       !Constraint 1 for absolute value;

@FOR (LINKS(I,J):
        -XO(I,J) + XI(I,J)-Y(I,J) < 0);      !Constraint 2 for absolute value;

@FOR (LINKS(MACHINES,JOBS):@BIN(XI));        !THIS FUNCTION WILL MAKE THE
                                             DECISION VARIABLES BINARY;


data:
  PROCESSING=@pointer(3);
  XO=@pointer(4);
  ES=@pointer(5);
  jiji=@pointer(6);
  @pointer(7) = @status();
  @pointer (8) = XI;

enddata

END
```