# Random Keys Genetic Algorithms Scheduling and Rescheduling Systems for Common Production Systems

Elkin Rodriguez-Velasquez
*Old Dominion University*

# RANDOM KEYS GENETIC ALGORITHMS SCHEDULING AND RESCHEDULING SYSTEM FOR COMMON PRODUCTION SYSTEMS

by

Elkin Rodríguez-Velásquez

B.S. June 1997, Universidad Nacional de Colombia
M.Sc. June 2000, Universidad Nacional de Colombia

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

DOCTOR OF PHILOSOPHY

ENGINEERING MANAGEMENT

OLD DOMINION UNIVERSITY
May 2011

Approved by:

Ghaith Rabadi. (Director)

Ali Ardalan (Member)

Shannon Bowling (Member)

Resit Unal (Member)

## ABSTRACT

RANDOM KEYS GENETIC ALGORITHM SCHEDULING AND RESCHEDULING SYSTEM FOR COMMON PRODUCTION SYSTEMS

Elkin Rodríguez-Velásquez
Old Dominion University, 2011
Director: Dr. Ghaith Rabadi

The majority of scheduling research deals with problems in specific production environments with specific objective functions. However, in many cases, more than one problem type and/or objective function exists, resulting in the need for a more generic and flexible system to generate schedules. Furthermore, most of the published scheduling research focuses on creating an optimal or near optimal initial schedule during the planning phase. However, after production processes start, circumstances like machine breakdowns, urgent jobs, and other unplanned events may render the schedule suboptimal, obsolete or even infeasible resulting in a "rescheduling" problem, which is typically also addressed for a specific production environment, constraints, and objective functions.

This dissertation introduces a generic framework consisting of models and algorithms based on Random Keys Genetic Algorithms (RKGA) to handle both the scheduling and rescheduling problems in the most common production environments and for various types of objective functions. The Scheduling system produces predictive (initial) schedules for environments including single machines, flow shops, job shops and parallel machine production systems to optimize regular objective functions such as the Makespan and the Total Tardiness as well as non-regular objective functions such as the Total Earliness and Tardiness.

To deal with the rescheduling problem, and using as a basis the same RKGA, a reactive Rescheduling system capable of repairing initial schedules after the occurrence of unexpected events is introduced. The reactive Rescheduling system was designed not only to optimize regular and non-regular objective functions but also to minimize the instability, a very important aspect in rescheduling to avoid shop chaos due to disruptions. Minimizing both schedule inefficiency and instability, however, turns the problem into a multi-objective optimization problem, which is even more difficult to solve.

The computational experiments for the predictive model show that it is able to produce optimal or near optimal schedules to benchmark problems for different production environments and objective functions. Additional computational experiments conducted to test the reactive Rescheduling system under two types of unexpected events, machine breakdowns and the arrival of a rush job, show that the proposed framework and algorithms are robust in handling various problem types and computationally reasonable.

This dissertation is dedicated to my parents, who taught me the love of learning.

## ACKNOWLEDGMENTS

I would first like to express my appreciation for my advisor, Dr. Ghaith Rabadi. I was fortunate to find that he not only had a shared interest in the field of scheduling, but was an excellent guide. I thank him for always being there to talk with me about my research and also for his support, patience, and words of motivation at those crucial moments.

I would also like to thank the other members of my doctoral committee. Dr. Resit Unal, Dr. Shannon Bowling, and Dr.Ali Ardalan provided me with their kind advice.

I am thankful for the support of the Engineering Management and Systems Engineering Department.

I am also grateful for the support of the Universidad Nacional de Colombia, which was a great aid to me throughout my doctoral studies.

I thank my friend, Dr. Alexander Correa Espinal, for finding the time to offer me his statistical advice in the experimental design phase, despite being occupied with a variety of activities. I would also like to express my gratitude to the Bedoya-Correa family, who helped me take the first step in this journey at Old Dominion University. My gratitude goes out to the Padilla-Parra family, for the invaluable friendship and advice they offered me from the very first moment. Finally, I am very grateful to the Carvajalino-Palacio family. They brought me the warmth and happiness of Colombia in both pleasant and tough times.

I would like to show my absolute appreciation for my friend, Professor William Alvarez Bermudez. William, things started to happen after you explained the four friends' problem to me during my undergrad years. Thanks for sharing your knowledge, for your unconditional friendship and for encouraging me to pursue this goal. This Ph.D. belongs to you too.

I am heartily thankful to have Luz Bibiana by my side. Despite being thousands of miles away from each other, she has been with me all the time, and is a great source of motivation and inspiration.

# TABLE OF CONTENTS

## LIST OF TABLES

Table Page

## LIST OF FIGURES

# 1 INTRODUCTION

Scheduling theory is concerned with the allocation of a set of limited resources over time to process a set of jobs (Baker, 1974). Scheduling in manufacturing and production environments varies to include most commonly single machines, parallel machines, flow shops, job shops and their combinations. Depending on the nature of the business, scheduling problems may have different objective functions and processing conditions or restrictions. Over the past few decades, it has been shown that the majority of scheduling problems with various objective functions are explosively combinatorial in nature (Pinedo, 2008).

Most of the published scheduling research focuses on creating an optimal or near optimal initial schedule during the planning phase (see, for example, Muth and Thompson (1963), Conway, Maxwell and Miller (1967), Baker (1974), French (1982)). However, after production processes start, different circumstances like machine breakdowns, material delays, urgent jobs, and other unplanned events may render the schedule suboptimal, obsolete or even infeasible. In such cases, the scheduling problem turns into a "rescheduling" problem. In spite of their importance, rescheduling problems have not been broadly studied in the literature as much as scheduling problems, nor have they been adequately implemented in practice due to the difficulty of dealing with unexpected events (MacCarthy and Liu (1993), Mehta and Uzsoy (1999), and Arnaout and Rabadi (2007 and 2008)).

Rescheduling strategies may be divided into three main categories: Online Scheduling, Robust Scheduling, and Reactive Scheduling (Mehta and Uzsoy (1999), Arnaout (2006)). In Online Scheduling, there is no initial schedule to adhere to; instead, decisions are made locally, using dispatching (heuristic) rules to select the next job to process when an event disrupts the schedule. The main weakness of this approach is that quality of the schedule is typically poor, and it does not allow for resource planning (Mehta and Uzsoy (1999)). Robust Scheduling, on the other hand, anticipates unexpected events and develops an initial (or predictive) schedule with built-in flexibility (e.g., Esswein, Billaut, and Strusevich (2005)) or redundancy (e.g., Herroelen and Leus (2005)) to account for future events. There are two main drawbacks to this approach. First, it is very difficult to anticipate the type and timing for an event to occur, and, second, it is quite unlikely for schedulers in practice to voluntarily insert idle time in the schedule or keep too many redundant resources idle in anticipation of events that may take place. Instead, they usually deal with events as they occur. This leaves Reactive Scheduling as the most viable and practical option to deal with rescheduling problems.

In Reactive Scheduling, a schedule is created in response to interruptive events and three strategies are commonly applied (Abumaizar and Svestka, 1997): Total Rescheduling, Right-shift Rescheduling and Affected Operations Rescheduling. Total Rescheduling creates a totally new schedule for the operations that have been interrupted and for those that have not been started yet. Right-shift Rescheduling delays the start of all operations in the schedule by the time required to make it feasible. Affected Operations Rescheduling takes into account that not all operations may be affected by an event, so it delays only the ones that are affected by the event (either interrupted or that have been delayed due to delay in their preceding operations). These rescheduling methods, however, have been studied under specific problems with specific objective functions and constraints. There is a clear lack of research in the literature for a dependable reactive rescheduling system that can effectively repair schedules in a generic fashion regardless what the production environment is or what the objectives and constraints are. Most industries currently resort to manual or semi manual rescheduling when unexpected events occur. Most research and software scheduling systems focus on creating a good initial plan or schedule and few worked on or included rescheduling aspects.

This dissertation attempts to close this research gap by introducing algorithms and methods based on a generic framework that are capable of repairing schedules in most common production environments and for most objective functions. The remainder of this document is organized as follows:

background is presented in Chapter 2 to introduce the topic of scheduling. General aspects about scheduling such as problem classification and some specific examples are presented in order to familiarize the reader with the field. The third chapter presents the literature review in the area of rescheduling, covering Online, Robust and Reactive scheduling. The conclusions of this chapter address the research gap. The research purpose is presented in Chapter 4, where the scope and the general and specific objectives are discussed. Chapter 5 introduces a procedure to reduce Earliness and Tardiness in diverse types of schedules. Chapter 6 presents the generic predictive scheduling model covering the Random Keys Genetic Algorithms approach and the theoretical principles upon which such a model is built to generate solutions for the different production environments and objective functions covered by this work.

In order to test the predictive model, computational experiments for different production environments, problem sizes and objective functions are presented in Chapter 7. In Chapter 8, the generic reactive model is presented to deal with different unexpected events and the quality and stability of the reactive solution. The predictive and the reactive model are connected in this chapter. The computational experiments to test the reactive model under different unexpected events are presented in Chapter 9. Finally, Chapter 10 discusses the research conclusions, contributions, and future research.

# 2 BACKGROUND

## 2.1 Field of Study

Scheduling theory is concerned with the allocation of a set of limited resources over time to process a set of jobs (Baker, 1974). Although scheduling problems may be present in long term planning, scheduling has generally been associated with the short term planning level, which is sometimes called operative planning (Sipper and Bulfin, (1997)). Over the last few decades, scheduling has become an area of knowledge for which there are well known textbooks used in academia and research, like the ones by Baker (1974), French (1982) and Pinedo (2008), scientific journals dedicated to it such as the *Journal of Scheduling*, published by Springer and the *International Journal of Planning and Scheduling* by Inderscience, in addition to many Operations Research and Industrial Engineering journals that publish Scheduling research.

## 2.2 Scheduling Problem Classification

Graham, Lawler, Lenstra, and Rinnooy Kan (1979) introduced a notation that is widely accepted in the literature, and is commonly called Graham's notation, for classifying scheduling problems. It consists of a triple $\alpha$ / $\beta$ / $\gamma$. The first symbol, $\alpha$, represents the environment of the shop. It can be a single machine (1), parallel machines (Pm), a flow shop (Fm), a job shop (Jm), an open shop (Om) or combinations of these where m is the number of machines. $\beta$ refers to specificities of the problem and $\gamma$ represents the objective function, which is typically a function of the jobs' completion times, i.e. the time at which each job is finished in the schedule. Depending on the instances of $\alpha$, $\beta$ and $\gamma$ under consideration, some problems can be solved by optimal or heuristic techniques. The next section discusses the differences between such techniques and the situations in which they can be used according to the problems' characteristics.

## 2.3 Problem Solving Techniques for Scheduling Problems

Most scheduling problems belong to a category for which it may not be possible to find optimal solutions for large problems, even with the best computational techniques developed so far (Baker and Trietsch, 2009); these problems are called NP-hard. For other problems, there are techniques that are able to find the solution in a reasonable time. These are usually called polynomial time algorithms and the problems they can solve, polynomial time problems (Pinedo, 2008).

Regarding the solution techniques, there are general purpose methods for solving combinatorial problems (General) or methods designed to solve specific scheduling problems (Specific). As for the optimality, there are methods that guarantee finding an optimal solution (Exact) and approximate methods (Approximate) that do not guarantee such solutions. Therefore, not all methods may be used with every problem. While NP-hard problems may be solved only by approximate methods in a reasonable computational time, polynomial time problems may be solved by general or specific methods, exact or even approximate, keeping in mind that optimality can only be guaranteed by exact methods.

According to such classification, Table 1 shows some examples of the different methods and problems they can solve. For example, although it is possible to use an approximation technique to solve a relatively easy problem (polynomial time problem) like Genetic Algorithms for the $J2/\ /Cmax$ problem (minimizing the makespan in a Job Shop with two jobs), it may not be necessary since there are general and specific techniques available that guarantee optimality. In the same vein, it is possible to unnecessarily use a general purpose method like branch and bound to find the optimal solution to a polynomial time problem like $1/\ /U$ (minimizing the number of tardy jobs on a single machine), while the tailored optimal algorithm by Moore (1968), known as Hodgson's algorithm, can solve it optimally.

**Table 1. Example of solution techniques according to the type of method**

| Purpose Optimality | General | Specific |
|---|---|---|
| **Exact** | Branch and Bound or Integer Linear Programming for $1//U$ | Moore (1968) for $1//U$ Johnson (1954) for $F2//Cmax$ Jackson (1956) for $J2//Cmax$ |
| **Approximation** | Genetic Algorithms or Tabu Search for $J2//Cmax$ (not NP-hard) or $J100//Cmax$ (NP- hard) | Giffler and Thompson (1960) for $J2//Cmax$ (not NP-hard) or $J100//Cmax$ (NP-hard) |

In order to present some examples and discussions throughout this dissertation, and due to the research relevance, the Job Shop Scheduling Problem and the so-called regular and non-regular performance measures are briefly explained in the next two sections.

## 2.4 Job Shop Scheduling

The Job Shop Scheduling Problem has been extensively studied in scheduling theory since it is a common case in manufacturing. It consists, in the most classical case, of a finite set of available jobs to be executed on a set of finite machines that are continuously available. Each job has a sequence of operations for which the processing times are known and assumed to be deterministic. Each operation needs to be processed on one machine. The sequence of operations through the machines is not necessarily the same for all jobs. The resulting schedule must respect two main constraints: one machine can process only one operation at a time, and the sequence of operations on the machines must be respected for each job. The most commonly used objective function for the Job Shop Scheduling Problem (JSSP) is the makespan ($Cmax$), i.e. minimizing the completion time of the job ending last, which represents the time necessary to process all jobs. In terms of Graham's notation, the problem is classified as $Jm/\ /Cmax$, where $Jm$ denotes a job shop with $m$ machines. Given its combinatorial complexity (Garey and Johnson, 1979), numerous optimal and non-optimal approaches have been presented to deal with this problem. An extensive review can be found in Jain and Meeran (1999).

## 2.5 Regular and Non-Regular Performance Measures

Most research on scheduling has been devoted to problems with objective functions that belong to a category called Regular Performance Measures. These are defined as functions that are not decreasing in the completion times (Pinedo, 2008). The makespan, explained in the previous section, belongs to such a class. To familiarize the reader with the concept, an example is presented below.

Suppose a 2 job, 2 machine JSSP with the processing times and routes through the machines shown in Table 2.

**Table 2. A 2 x 2 Job Shop Problem**

| Job | Processing Time(Operation Routing) | |
|---|---|---|
| 1 | 3(1) | 5(2) |
| 2 | 4(2) | 1(1) |

Job 1 consists of two operations with processing times 3 and 5 respectively; the first operation is processed on machine 1 and the second one on machine 2. Job 2 has also two operations with processing times 4 and 1 respectively; the first operation is processed on machine 2 and the second one on machine 1.

A feasible solution for a scheduling problem is usually represented in a Gantt chart as the one shown in Figure 1 for the example at hand, where the horizontal axis represents time and the machines (M1 and M2 in this case) are on the vertical axis. The number of each operation in the chart corresponds to the job to which they belong.

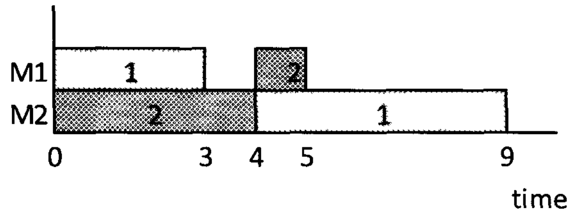


**Figure 1 A 2 x 2 Job Shop Problem**

If we want to minimize the makespan, the problem we have, in terms of Graham's notation, is $J2//Cmax$. In this case, $Cmax$, the completion time of the last operation is calculated as:

$$Cmax = \max\{C_i\} \quad \forall i = 1, \dots n \qquad (1)$$

where $C_i$ corresponds to the completion time of job $i$ and $n$ is the number of jobs. Therefore, we have:

$$Cmax = \max\{C_1, \ C_2\} = \max\{9, \ 5\} = 9.$$

We can see that $Cmax$ is not decreasing in function of $C_1$, $C_2$; that is to say, after increasing $C_1$ or $C_2$, $Cmax$ will increase or remain the same, but it will not decrease regardless of the sequence.

Let us use now an objective function called Earliness and Tardiness, which calculates the summation of the time deviation of each job depending on whether it finishes earlier or later than its due date, aiming that all jobs finish exactly on their correspondent due dates. This function, which we will denote as $ET$, is defined as:

$$ET = \sum_{i=1}^{n} | C_i - d_i | \quad (2)$$

where: $C_i$ corresponds to the completion time of job *i*,

$d_i$ is the due date of job $i$ and

$n$ is the number of jobs.

If we want to minimize $\mathrm{ET}$, the problem we have, in terms of Graham's notation, is $J2//\mathrm{ET}$. Now suppose that both jobs' due dates are equal; that is, $d_1 = d_2 = 9$.

Since all jobs have the same due date, this is called a Common Due Date in the literature. Therefore the value of $\mathrm{ET}$ is:

$$\mathrm{ET} = |C_1 - d_1| + |C_2 - d_2| = |9-9| + |5-9| = 4.$$

Suppose that we increase $C_1$ by one unit; that is, we start the last operation of job 1 on time 5 and consequently it finishes on time 10, then:

$$\mathrm{ET} = |10-9| + |5-9| = 5.$$

As in the case of $\mathrm{Cmax}$, the $\mathrm{ET}$ objective function increased after increasing $C_1$. However, if we do not change $C_1$, but increase $C_2$, so the last operation of job 2 starts at time 5 and therefore finishes at time 6, we have:

$$\mathrm{ET} = |C_1 - d_1| + |C_2 - d_2| = |9-9| + |6-9| = 3.$$

So $\mathrm{ET}$ decreased when certain $C_i$ increased. Consequently, $\mathrm{ET}$ is not a regular performance measure. While in regular measures the objective function never decreases as the completion time increases, in non-regular measures the objective function may decline. Figure 2 represents the behavior of the $\mathrm{ET}$ as a function of the completion time. As the completion time approaches the due date from the left, the Earliness decreases. When the job is completed just in time, at the due date, the objective function value is zero. As the completion time starts deviating from the due date to the right, the Tardiness increases. Non-regular functions have caught the attention of researchers in the last three decades as they capture the essence of the Just-In-Time philosophy, but they make scheduling problems more challenging due to the nature of the objective function.

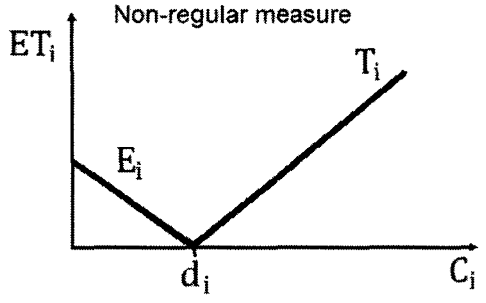


**Figure 2 Earliness/Tardiness ET of job i**

# 3 LITERATURE REVIEW

## 3.1 Scheduling and Rescheduling

Several papers (e.g. MacCarthy and Liu (1993)), report the existence of a gap in scheduling between a previously generated schedule and common real life unforeseen situations, like machine breakdowns, rush jobs, order cancellations, delays in the arrival of materials, etc. This gap may be caused by the fact that scheduling theory has mainly focused on the problem of producing the initial (predicative) schedule and most of the research assumes that it will be executed exactly as it was initially created. But, when such a schedule is being executed unexpected events may make it infeasible, indicating the necessity to update it. This schedule updating phase has not been as broadly studied as much as the initial phase. In the same sense, Graves (1981), states that "there is no scheduling problem but rather a rescheduling problem".

In response to that, some researchers started presenting strategies to deal with such unexpected events in different shop environments. Scheduling models can be divided into those that deal with the mentioned unforeseen events, usually referred to as rescheduling, and those that do not. The vast majority of research papers have been focusing on traditional scheduling that does not consider the rescheduling problem.

## 3.2 Rescheduling

Regarding the research approaches that deal with unforeseen events, three types of approaches have been identified: online, predictive-reactive and robust scheduling (Mehta and Uzsoy (1999), Arnaout (2006)).

In online scheduling there is not an initial schedule; the decisions are made locally, using previously established priority rules (dispatching rules) to select the next job to be processed on each machine once it is ready after being unavailable for any reason. In such cases the shop can be modeled by simulation and different policies can be tested. Since there is no initial schedule, this approach has the weakness of not allowing any resource planning, as Mehta and Uzsoy (1998) and Arnaout (2006) stated.

In the predictive-reactive scheduling, an initial (predictive) schedule is generated and then changed according to different policies when an unexpected event occurs; hence, it is called predictive-reactive scheduling. The original schedule is changed attempting to minimize the impact on the system's performance. To produce the reactive schedule, three main policies are reported in the literature (Vieira, Herrmann, and Lin, 2003): periodic, event driven or hybrid. Three strategies are known in the literature to generate the reactive schedule: (Abumaizar and Svestka, 1997) total rescheduling, right-shift rescheduling and affected operations rescheduling (Other authors like Herrmann (2006) and Vieira et al., (2003), refer to total rescheduling as complete regeneration, and affected operations rescheduling as partial rescheduling).

Total Rescheduling consists of solving the new scheduling problem that a shop has once an unexpected event occurs, taking into account the operations not yet started and the one(s) interrupted. Right-shift rescheduling delays the starting of all operations in the schedule by the time required to make the schedule feasible. In Affected Operations Rescheduling, given a disruption, not necessarily all the operations in the Gantt chart have to be moved to the right; only some operations are affected due to the delay in their preceding operation in their machine or in their job and only those are moved to the right (recall that a job typically consists of multiple operations).

The differences between right-shift rescheduling and Affected Operations Rescheduling are presented in Figures 3 to 5. In Figure 3, part of a predictive schedule for the famous 6 x 6 job shop benchmark problem (i.e. 6 jobs and 6 machines where each job has one operation on each machine) from Muth and Thompson (1963) is shown in a Gantt chart, where the number of each operation on the Gantt chart represents the job to which it belongs. The makespan (completion time of the last operation) is 55. In Figure 4 a breakdown takes place on machine 4 from time 25 to 28. Assuming the so-called "interrupt repeat mode", that is, when an operation is interrupted, its processing has to start from scratch when it is rescheduled, the interrupted operation (the one of job 1 on machine 4) as well as all the operations scheduled to start at or after time 25 are "right shifted" by 6 time units, corresponding to the amount of time the start of the interrupted operation had to be delayed. The new makespan is 61 time units. In Figure 5, Affected Operations Rescheduling is applied instead. Only operations in bold are right shifted, and as can be seen, there is no need to displace the other ones. The new makespan is 56 time units, much closer to the predictive schedule's makespan.

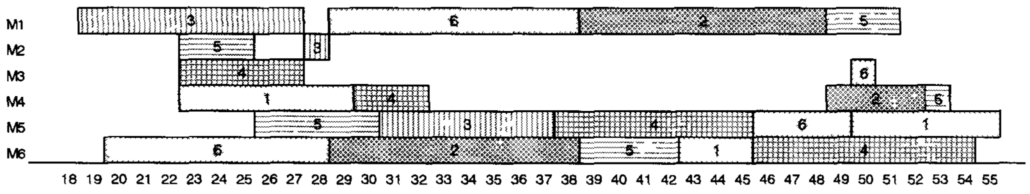


**Figure 3 Example of a Schedule represented on a Gantt chart**

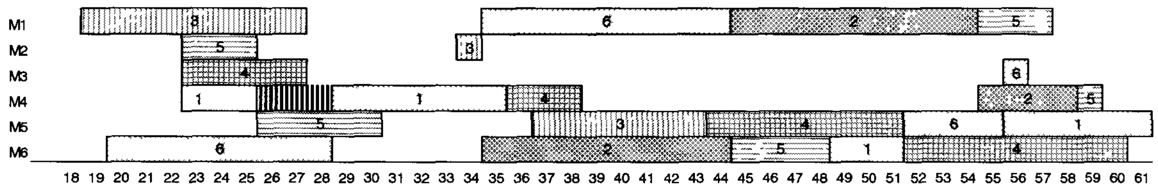


**Figure 4 A breakdown on M4 from time 25 to 28.Operations starting after the breakdown and disrupted operation are right shifted 6 time units**

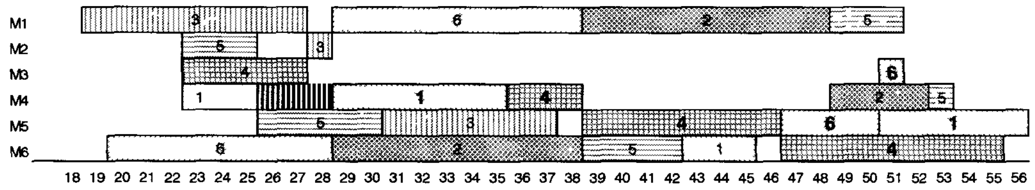


**Figure 5 A breakdown on M4 from time 25 to 28. Only the affected operations are right shifted 6 time units**

Total Rescheduling may produce a very different sequence on the machines in response to the disruption, while the other two approaches maintain the predictive sequence. Wu, Storer, and Chang (1993) define stability, in the context of rescheduling, in two ways: the starting time deviations between the new schedule and the original schedule and a measure of the sequence difference between the two schedules. Using these terms, predictive-reactive scheduling by Total Rescheduling may cause a high level of instability.

Mason, Jin and Wessels (2004) compare the right-shift rescheduling, affected operations rescheduling (Fixed-Sequence Rescheduling as they call it) and Total Rescheduling in a Complex Job Shop i.e. a job shop that has different job release times, batching machines, parallel machines, sequence dependent setup times and recirculation  reentrant product flow through some machines), when the objective is to minimize the total weighted tardiness. They conduct diverse experiments with one unplanned machine breakdown happening early and late in the schedule's span, with different breakdown durations. They compare all methods in terms of efficiency (in this case, total weighted tardiness). Due to the production environment complexity, in all cases Total Rescheduling performs better than both right-shift rescheduling and affected operations rescheduling despite requiring more computational time.

Abumaizar and Svestka (1997) introduce the Affected Operations Rescheduling (AOR) algorithm to reschedule job shops when the objective function is the makespan and compare it to total rescheduling and right-shift rescheduling in terms of efficiency and stability. They conduct an experiment in which a breakdown may occur in one of two levels: early or late in the predictive schedule. The experimental design shows that the AOR algorithm outperforms the other two methods. Subramaniam and Raheja (2003) study different types of disruptions in a shop: machine breakdown, process time variation, arrival of an unexpected job and a job that becomes urgent. They propose repairing the schedule if one of these disruptions occurs by using AOR as a basis and by utilizing a set of algorithms oriented to four actions: Insert Idle Time, Insert Adjustment Time, Insert Operation and Delete Operation. They call their approach Modified Affected Operations Rescheduling (mAOR). In a later work, Subramaniam, Raheja, and Rama Bhupal Reddy (2005) compare the mAOR to the right-shift rescheduling, this time simulating multiple disruptions that occur in a Job Shop using different mean time between disruptions, proportion of disrupted operations and average duration of the disruptions. According to their experimentation parameters, they focus on one-disruption-at-a-time scenarios for their experiments. The experimental results show that mAOR performs better than right-shift rescheduling.

Match up rescheduling is a special type of affected operations rescheduling AOR. Match up uses different strategies to repair a schedule up to a certain point in time in which the repaired (reactive) schedule matches the predictive schedule. There is little research on this approach. It was proposed and studied first in the context of a single machine by Bean and Birge (1986). Later on, Bean, Birge, Mittenthal, and Noon (1991) address a case of parallel machines from the automobile manufacturing industry and find that match up outperforms right-shift and online rescheduling. Akturk and Gorgulu, (1999) study a cellular manufacturing system where each cell is what they call a Modified Flow Shop, a flow shop where not all jobs start at the same machine and/or leave the system at the same machine. Moratori, Petrovic and Vázquez (2008) propose a match up strategy to include new rush jobs in a flexible job shop's schedule. In the experiments they compare their match up approach with right-shift and total rescheduling. They report that the algorithm produces stability similar to the one of right-shift and a performance similar to the one produced by total rescheduling.

Robust scheduling, on the other hand, tries to anticipate unexpected events by using different strategies like the following:

- Schedule flexibility: As defined by Esswein et al. (2005), it is achieved by generating a schedule composed of sequences of groups of permutable operations on each machine. They use the concept of groups based on the former work of Erschler, Roubellat, and Vernhes

(1976). Artigues, Billaut, and Esswein (2005) use this approach to solve the JSSP where ready times are not necessarily zero and the objective function becomes a compromise between the quality of the schedule and a flexibility measure that they defined. In their case, the schedule quality was defined by the maximum lateness $L_{max}$ where Lateness is defined as the difference between the completion time of a job and its due date.

- Redundancy: It can be oriented either to the resources or to time (Herroelen and Leus (2005)). In the first case, it can be accomplished by introducing multiple machines, tools, personnel, etc. ready to absorb the disturbance. In the second case, idle time is inserted in the initial schedule between activities so that, if a disruption happens, a simple adjustment can be made, maintaining system performance. As for the time redundancy, the idea is to insert idle time to obtain a pre-schedule, able to be reconstructed after a breakdown. Mehta and Uzsoy (1998) propose an approach to insert idle time in a predictive schedule for a Job Shop where the objective function is to minimize the maximum Lateness, assuming the probability distribution of the time between machine breakdowns and their duration are known. Arnaout (2006) presents a robust rescheduling architecture for the Unrelated Parallel Machine Scheduling problem to minimize the makespan $(Rm/\ /C_{max})$ where he used a rule to insert idle time to generate the predictable schedule. He states that the architecture could be adapted to other environments different from parallel machines.

In some cases, operations are started as soon as they become available as in Mehta and Uzsoy (1998) to absorb idle time, while in others the predictive schedule is respected when it is possible as in Arnaout and Rabadi (2008). To achieve robustness, the first option (starting operations as soon as they are available) may be desirable, but to avoid earliness penalties the second option (trying to hold on to the preemptive schedule's starting times) is more appropriate. Pinedo (2008) confirms that there is a trade-off between starting operations early, in order to have robustness in the schedule, and starting them as late as possible to avoid holding costs.

Another strategy to implement robust scheduling, according to Pinedo (2008) is to keep the bottleneck machines fed. Regarding the concept of robustness in general, he states that little research has been done on this topic.

## 3.3 Concluding Remarks

The previous literature points out some important facts. Although the online rescheduling is a more appropriate option when disruptions are too frequent (Bean et al. 1991) it is disadvantageous in the sense that it does not provide any resource planning (Arnaout, 2006).

When there is available information on the disruptions (like the time between breakdowns and their duration), this may be used to produce a robust predictive schedule as in Mehta and Uzsoy (1998) and Arnaout (2006), among others.

There are, however, other classes of uncertainties in manufacturing environments, like the ones McKay, Buzacott and Safayeni (1989) call "complete unknowns", which are unpredictable and may cause multiple disruptions on the machines like a power outage or a sudden strike. Consequently, it is difficult to take into account situations like those by a robust approach.

The Predictive-reactive approach, on the other hand, does not require previous information about the disruptions. However, the reviewed literature shows that the research has focused on "one disruption at a time" scenarios for their experiments (see for example: Abumaizar and Svestka, (1997) and Subramaniam et al. (2005)), and to our knowledge, no prior research on rescheduling describes the study of simultaneous breakdowns.

## 3.4 Research Gap

The previous section shows some important gaps. Most of the current scheduling research solves scheduling problems using methods that are tailored to specific problems with specific objective functions and constraints. There is a need for general representations that can be used for reactive scheduling in different shop environments and for various objective functions.

As was shown, the research has focused on "one disruption at a time" scenarios. There is a lack of research studying simultaneous disruptions. To our knowledge, no prior research addresses simultaneous breakdowns in the context of reactive scheduling.

Most of the work published has focused on regular objective functions. The literature review shows a lack of research on reactive scheduling considering both regular and non-regular measures (specifically the non-regular objective function of minimizing the total Earliness and Tardiness).

This research will address the previous research gaps and develop a Predictive- Reactive Scheduling system usable in different shop environments and with different objective functions.

# 4 RESEARCH PURPOSE AND SCOPE

It is the overall objective of this research to develop a predictive - reactive scheduling system that is capable of repairing schedules for the most common production environments when unexpected events take place. The purpose is to coherently integrate new and existent approaches for rescheduling by implementing a higher level tool to repair a schedule once disrupted by unforeseen events such as (multiple) machine breakdown, job priority change, arrival of urgent jobs, and longer than expected processing times among other typical events. The scope will encompass solutions for single machine, parallel machine, flow shop, and job shop environments with regular and non-regular objective functions. The specific objectives of the project can be summarized as follows:

1. Introduce a coding schema general enough to approach scheduling problems in the most common production environments including: single machines, parallel machines, flow shops, and job shops with the most common regular performance measures (objective functions) including: the makespan, total tardiness, maximum lateness, and total completion time, in addition to the non-regular objective function of the total earliness and tardiness.

2. Develop and implement an encoding/decoding algorithm to translate a solution representation into a schedule and vice versa based on the coding scheme in objective 1.

3. Develop and implement a Meta-heuristic schedule repair algorithm that will be able to react to unexpected events in various production environments. Specifically, a Random Key Genetic Algorithm (RKGA) will be the Meta-heuristic of choice for reasons that will be discussed in Chapter six in which the Predictive model is presented.

4. Perform experiments to test the individual components of the proposed scheduling systems as well as the whole system. The experiments will mostly be based on randomly generated data that will cover realistic and common

manufacturing and production systems (listed earlier) as well as different types of disruptions and objective functions.

# 5 A PROCEDURE FOR THE NON-REGULAR MEASURE OF EARLINESS AND TARDINESS

Most of the body of literature on scheduling has been dedicated to problems with Regular Performance Measures (see, for example, the books of French (1982), Pinedo (2008) and Baker and Trietsch (2009)). The introduction of the just-in-time (JIT) production approach brought to attention an important fact to scheduling theory which is that it is not necessarily always beneficial to complete the jobs as early as possible as this may increase the holding cost. Therefore, it became necessary to minimize both earliness and tardiness for jobs from their due dates. Minimizing tardiness would reduce the cost of missing due dates or the loss of customers while minimizing earliness would reduce the holding or inventory cost. This problem is known in the literature as the early/tardy (ET) problem. Although JIT entails more detailed concepts, the ET problem seems to mathematically capture the scheduling essence of it. In this chapter we study some characteristics of the $1/\ /ET$ problem when the due date of all jobs is common in order to propose a procedure that aims to reduce the ET of a schedule by delaying the start time of some operations.

## 5.1 Minimizing ET in a Single Machine

Consider the problem of scheduling four jobs on a single machine with processing times *3, 4, 5* and *2* for jobs *1, 2, 3* and *4* respectively. Suppose we decide to process them in the sequence: 4 – 3 – 1 – 2. If the objective function is a regular measure such as $C_{max}$, we must build a schedule where all operations must be started as early as possible. In the single machine case the resulting schedule will not have any idle time between operations.

The resulting schedule can be represented in the Gantt chart in Figure 6, where M1 is the only machine in the problem.

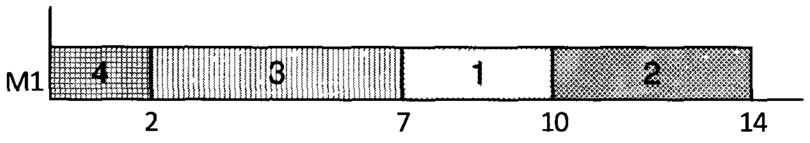


**Figure 6. Schedule for a single machine problem**

Such a solution is reasonable when optimizing a regular function since schedules without inserted idle time determine a dominant set for any regular measure of performance (Baker and Trietsch, 2009). However, if the objective function is not regular, it may be desirable to have idle time before some or all of the jobs start processing. Consider the same set of jobs, but in this case all of them have the same due date, $d = 15$, and the objective function is the non-regular measure of Earliness/Tardiness as defined in expression (2) of Chapter 2.

In this case, any optimal solution will have some idle time inserted on machine 1 before the first job starts processing (Baker and Trietsch, 2009). Three optimal solutions for the problem with $ET = 11$ are shown in Figure 7.

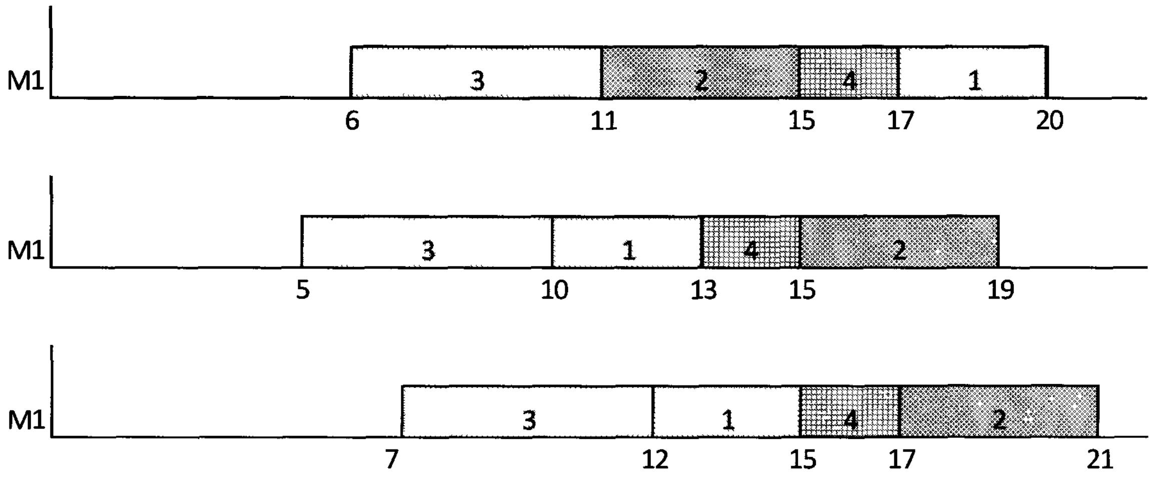


**Figure 7. Three Optimal Schedules for a $1/\ /ET$ problem with a Common Due Date= 15**

### 5.1.1 The Unrestricted Single Machine Model with a Common Due Date

Regarding the 1/ /ET problem with a common due date (CDD) problem Baker and Scudder (1990) defined a problem as unrestricted as follows.

If we sequence the jobs in a longest processing time order, we can call Δ to the summation of every other processing time. If the common due date CDD ≥ Δ, then the problem is unrestricted.

They list four dominance properties identified by Kanet (1981) for the unrestricted problem:

*I.* *"There is no inserted idle time in the schedule. (If job j immediately follows job i in the schedule, then Cj = Ci + pj.)*

*II.* *The optimal schedule is V-shaped. (Jobs for which Cj < d are sequenced in non-increasing order of processing time; jobs for which Cj > d are sequenced in non-decreasing order of processing time.)*

*III.* *One job completes precisely at the due date. (Cj = d for some j.)*

*IV.* *In an optimal schedule, the bth job in sequence completes at time d, where b is the smallest integer greater than or equal to n/2."*

If we call $p_{[i]}$ the processing time of the job in position *i* in the sequence, according to the definition, the problem is unrestricted if CDD ≥ Δ, where:

$$\Delta = \begin{cases} p_{[1]} + p_{[3]} + \cdots + p_{[n]} & \text{if n is odd} \\ p_{[2]} + p_{[4]} + \cdots + p_{[n]} & \text{if n is even} \end{cases} \qquad (3)$$

$$p_{[1]} \leq p_{[2]} \leq p_{[3]} \cdots \leq p_{[n]} \quad (4).$$

Notice that the condition for a problem to be unrestricted, CDD ≥ Δ, guarantees that there is enough time for the bth job to complete on the CDD.

Notice that given any predefined sequence, properties I, III and IV can be used to reduce the ET value (Baker and Trietsch, 2009). Suppose that we have a predefined sequence. First we must check if the problem is restricted or not in order to know if we can utilize the properties. In this case since the sequence is predetermined we must check that the summation of processing times of jobs in positions from 1 to b is not greater than the due date. If the condition holds, we can proceed to find the start times of all jobs. According to properties I and III, our schedule must not have inserted idle time and there must be a job that finishes exactly at the due date. Finally, property IV will let us determine which job should finish at the due date; therefore, we can determine the start and finish time of all jobs in the schedule.

As an example, suppose we have three jobs to schedule on a single machine with a common due date of 9 and processing times 1, 3 and 2 for jobs 1, 2 and 3 respectively. Suppose as well that we have decided to process them in the natural sequence 1 – 2 - 3. First note that, according to property IV, position b is obtained as the smallest integer greater than or equal to 3/2, which is 2. That corresponds to job 2 which is in the second position. The summation of processing times of jobs 1 and 2 equals 4 which is less than or equal to the common due date; therefore, the problem with a fixed sequence is unrestricted. By property III we know that job 2 must complete on 9 and property I states that no idle time should be inserted, which lets us determine the rest of the start and completion times in the schedule, as shown in Figure 8.

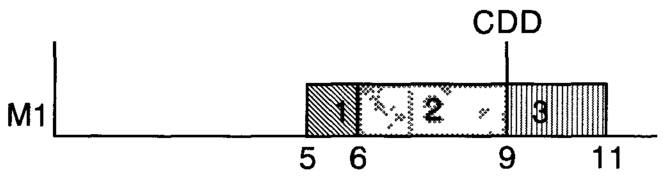


**Figure 8. Optimal Schedule to minimize ET for a single machine problem with a predefined sequence and a Common Due Date**

The previous properties have another application. Suppose that an initial feasible schedule is provided for the problem; that is, not only a predefined sequence is given but also the start or completion times. We can use the properties to check whether the schedule can be improved in terms of earliness and perform the correspondent changes in the start times if so. As an example, suppose the initial schedule in Figure 9 is provided.

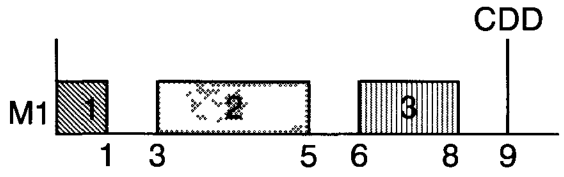


**Figure 9. Initial Schedule for a $1//ET$ problem with a common due date and a fixed sequence**

We already checked that the problem is unrestricted, so we can use Kanet's properties above to find the best starting times for all jobs. According to property I the solution does not have inserted idle time. Using properties III and IV we find the same schedule as the one in Figure 8.

**Definition 1**: Let the "optimal completion time" $C_0$ be the completion time of the last job of a job sequence that has been scheduled in a way such that the bth job completes at the common due date, satisfying property IV.

According to definition 1, in Figure 8 the optimal completion time $C_0 = 11$. Suppose now that we obtain the schedule shown in Figure 8. However, due to an event such as a maintenance job, the machine cannot be used starting from a time B, which we will refer to as a "boundary time", that is earlier than $C_0$, that is $B < C_0$ , in which case the schedule in Figure 8 cannot be completed. An example of the second scenario is shown in Figure 10.

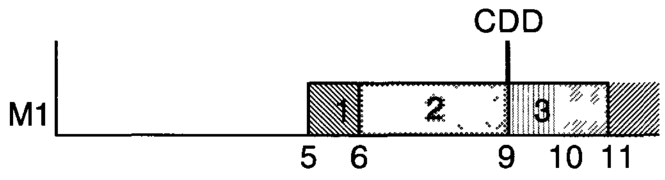


**Figure 10. An event at time 8 overlapping the schedule in Figure 8**

As can be seen, there is an event taking place from time B=10 to time 11 that prevents the schedule in Figure 10 from being completed.

Let us denote $c_{[i]}$ as the completion time of the job in position *i* in the sequence.

Regarding Kanet's property IV, from the work of Rabadi, Mollaghasemi and Anagnostopoulos (2004), it can be seen that when CDD > $C_{[b]}$, as the jobs move to the right in the schedule so that $C_{[b]}$ gets closer to CDD, ET decreases or remains the same. Therefore, when the last job cannot be completed beyond a boundary time B < $C_0$ it is always convenient to move the block of jobs to the right in the Gantt chart so the job in position b completes as close to the Common Due Date (CDD) as possible. This will decrease the ET value and may create idle time to the left that may be useful as will be explained later.

Therefore, in our example coming from the situation in Figure 9, it is still a good idea to start job 1 at time 5 so job 3 completes at time 10, although it cannot complete at $C_0$, as shown in Figure 11.

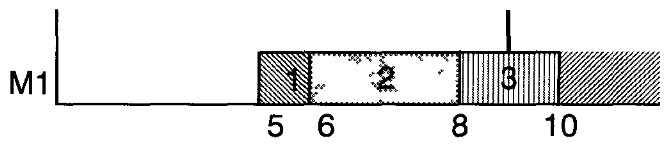


**Figure 11. Optimal solution for the initial situation of Figure 9**

The previous properties are used as building blocks of an improving strategy to solve single machine sub problems present in more general problems such as single machine, parallel machine, flow shop or job shop problems where there is not a common due date.

### 5.1.2 The General Single Machine Model (1/ /ET)

In the 1/ /ET problem, a set of simultaneously available jobs whose processing times and due dates are known in advance is given. Unlike the previous case, the due dates are not necessarily the same. In this case, the optimal sequence may not be V-shaped, and it may have inserted idle time (Baker and Scudder, 1990).

As an example, suppose we have four jobs to schedule on a single machine with processing times 1, 3, 2 and 2 for jobs 1, 2, 3 and 4 respectively. Additionally, jobs 1, 2 and 3 have a due date of 7, while job 4 has a due date of 12. Suppose again that the natural sequence of jobs 1-2-3-4 is used. Given that predefined sequence, a solution for the problem is shown in Figure 12.

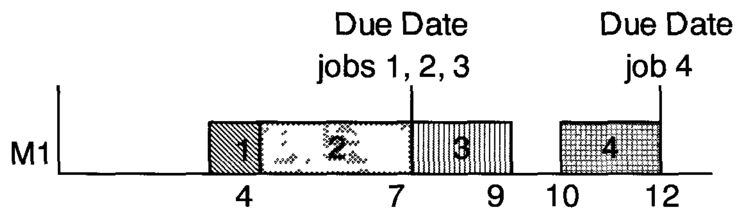


**Figure 12. Optimal Schedules for the $1//ET$ problem given a predefined sequence and different due dates**

Note that in this case, jobs 1, 2 and 3 form a common due date sub problem and job 4 does not interfere which, according to Kanet's property IV, lets us anticipate that the solution provided is optimal. However, it may not always be the case. Job 4's processing time could have been 4 time units, or its due date could have been 10 or earlier, which would give us a situation similar to the one presented in Figure 10. Another important thing to notice here is that we can see the problem as formed by two common due date problems not interfering with each other in this case, the first one with three jobs and a common due date = 7, and the second one with only one job and a "common" due date of 12.

An Earliness Reduction Procedure based on the previous properties is presented below.

### 5.1.3 Earliness Reduction Procedure for the 1/ /ET problem

Using an existing schedule as an input, the Earliness Reduction Procedure consists of exploring all jobs from right to left in the Gantt chart, identifying groups of jobs that share the following characteristics:

1. Are adjacent,
2. Have the same (common) due date,
3. Complete earlier or at due date.

It is important to notice here that as a result of the procedure, for a problem with $n$ jobs, there may be at the end $n$ unitary sets in one extreme case, no sets at all on the other extreme, or something in between.

Notice as well that if all jobs from a group complete earlier or at the due date, at most one can complete at the due date; otherwise there would be an overlap. Moreover, that means that $\sum_{i=1,\dots n} \mathrm{p}_i \leq$ CDD; consequently, from expression (3), $\Delta \leq$ CDD; therefore the problem of the group of jobs is unrestricted.

The procedure is described via pseudo code below and starts by determining the previously described groups of jobs, and the initial boundary time up to which the jobs in the group can be shifted to the right without overlapping with other jobs. That is done in steps 1 and 2 of the algorithm. In step 3 the idle time between jobs of the same group, if there is any, is eliminated by displacing all jobs to the right as close as possible to each other. In step 4, each group, starting from right to left in the Gantt chart, is moved to the right so its bth job (according to the definition given in property IV), completes as close as possible to the group's due date. Each group's boundary time is updated every time a group is right shifted.

The variables, functions and operations used by the procedure are described next. Then, the pseudo code of the procedure is presented.

**Variables:**

*CurrDueDate :* Stores the due date of the group currently being analyzed.

*GC* : Group counter in steps 1 and 2. After that stores the total number of groups.

*$Bound_{GC}$ :* Variable that stores the boundary time of group number *GC.*

*Due*_$Date_{GC}$ : Variable that stores the due date of group number *GC.*

$G_{GC}$ *:* Stores chronologically the set of jobs belonging to group *GC.*

**Complementary Functions:**

Job [*x*]: Returns the job in position *x* in a sequence.

*Position_of*(*x*): Returns the position of job *x* in sequence.

Due_Date(x): Returns the due date of job *x*.

*Start_time_of(x):* Returns the start time of job *x.*

*Completion_time_of(x):* Returns the completion time of job x.

*Earliest_job_of_group (g)*: Returns the earliest job in the schedule of group *g.*

Latest_job_of_group (*g*): Returns the latest job in the schedule of group *g*.

**Operations**:

Include *x* in $G_{GC}$ to the left: Inserts job x to the left of the set $G_{GC}$ displacing the other members to the right.

**Inputs:**

Number of jobs in the sequence (*Number_of_jobs*).

An initial schedule.

### 5.1.4 Pseudo Code of the Earliness Reduction Procedure for the 1/ /ET problem

**Step 1. Find the boundary of the first group, if such a group exists.**

Find the first early job *LJ* starting from right to left in the Gantt chart.

If there are no early jobs then, stop. The schedule does not change.

```
Create group counter $GC$. Set $GC$ =1
Create: group $G_{GC}$, $Bound_{GC}$ , $Due\_Date_{GC}$, $CurrDueDate$
Set $CurrDueDate$= Due_Date($LJ$)
Include $LJ$ in $G_{GC}$ to the left
If Position_of(LJ) = Number_of_jobs then
        $Bound_{GC}$ =infinity
Else
        $Bound_{GC}$ =Start_time_of(Job[Position_of(LJ) + 1])
End If
Set $Due\_Date_{GC}$ = $CurrDueDate$
```

**Step 2. Find the boundary of the rest of groups, if such groups exist.**

```
Search for the next early job CJ to left.
If any job CJ is found then
        If Due_Date(CJ) = CurrDueDate and Position_of(CJ) = Position_of (LJ) - 1 then
                Include CJ in set $G_{GC}$ to the left
                Set LJ=CJ
        Else
                Set GC = GC + 1,
                Create: $G_{GC}$, $Bound_{GC}$, $Due\_Date_{GC}$
                Set $Bound_{GC}$ = Start_time_of(Job[Position_of (CJ) + 1])
                Set CurrDueDate= Due_Date(CJ), $Due\_Date_{GC}$ = CurrDueDate
                Include CJ in set $G_{GC}$ to the left
                Set LJ=CJ
        End If
End If
If All Jobs have been explored then go to Step 3, otherwise, Go to Step 2.
```

**Step 3. Delete idle time inside groups.**

```
g= 1
Do while g <= GC
        PLJG= Position_of(Latest_job_of_group(g))
        PEJG= Position_of (Earliest_job_of_group(g))
        CurrStartTime = Start_time_of(Job[PLJG])
        j= PLJG-1
        Do while j >= PEJG
                Right shift Job[ j ] a time = CurrStartTime – Completion_time_of(Job[ j ])
                CurrStartTime  = Start_time_of(Job[ j ])
                j= j -1
        End while
        g= g + 1
End While
```

**Step 4. Displace groups to the right to reduce Earliness.**

```
For i = 1 to GC
   Set b = bth job of group i, according to property IV
   Set Cb= Completion_time_of(b)
   If Bound_i =infinity then
         Right Shift Jobs in set i a distance Due_Date_i – Cb
   Else
         Bound_i = Start_time_of(Job[Position_of(Latest_job_of_group(G_i)) + 1])
         Set PTA = Summation of processing times after bth job in set i, according to property IV
         If Bound_i ≥ Due_Date_i + PTA then
                 Right Shift Jobs in set i a distance Due_Date_i – Cb
         Else
                 Right Shift Jobs in set i a distance Bound_i – ( Cb + PTA)
         End if
   End if
Next i
```

## 5.1.5 Example of the Earliness Reduction Procedure for the 1/ /ET problem

Consider the 1/ /ET problem data in Table 3.

**Table 3. Example of a $1//ET$ problem**

| Job | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|
| processing time | 2 | 2 | 3 | 2 | 3 | 1 |
| due date | 14 | 2 | 14 | 14 | 2 | 18 |

And suppose the input schedule is given in Figure 13.

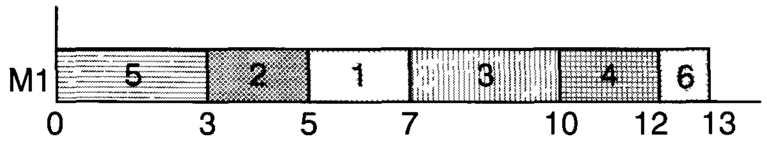


**Figure 13. Input schedule for the problem in Table 3**

**Step 1**

The first early job to the left is *LJ* =6

We create group counter *GC.* Set *GC* =1

And create also: $G_1$, $Bound_1$ , *Due*_$Date_1$, *CurrDueDate*

Set *CurrDueDate*= 18

$G_1$ = {6}

*Position_of(6) = Number_of_jobs then we set* $Bound_1$ *=infinity*

*Set Due*_$Date_1$ = 18

**Step 2.**

The next early job to left is *CJ* = 4

*Due_Date(4) is different from CurrDueDate,* then

We set *GC = 2*,

And create: $G_2$, $Bound_2$, *Due*_$Date_2$

$Bound_2$ = *Start_time_of(Job[5 + 1])* = 12

We set: *CurrDueDate*= 14, *Due*_Date$_2$ = 14

G$_2$ = {4}

Set *LJ=4*

Not all Jobs have been explored then we go to Step 2.

**Step 2.**

The next early job to left is *CJ*=3

*Due_Date(3) = 14* and *Position_of(3) = Position_of (4)* - 1 then

G$_2$ = {3, 4}

Set *LJ=3*

Not all Jobs have been explored then go to 2.

**Step 2.**

The next early job to left is *CJ*=1

*Due_Date(3) = 14* and *Position_of(1) = Position_of (3)* - 1 then

G$_2$ = {1, 3, 4}

Set *LJ=1*

Not all Jobs have been explored then go to 2.

**Step 2.**

There is not any other early job to the left. All Jobs have been explored then go to 3.

**Step 3. Delete idle time inside groups.**

```
g= 1
g <= 2 then
        PLJG= 6
        PEJG= 6
        CurrStartTime  = 12
        j= 6-1 = 5
        since 5 is not >= 6 we set g=g+1 = 2
```

```
g <= 2 then
    PLJG= 5
    PEJG= 3
    CurrStartTime = 10
    j= 5-1 = 4
    j >= PEJG then
        We right shift job in position j a distance = 10 – 10 = 0
        CurrStartTime = 7
        j= 3
        j >= PEJG then
        We right shift job in position j a distance = 7 – 7 = 0
        CurrStartTime = 5
        j= 2
        j is not >= PEJG anymore, then we set g = 3
    g is not <= 2 anymore, then we go to step 4.
```

At this point we have defined our groups: $G_1$ = {6} and $G_2$ = {1, 3, 4}. Additionally, we have:
$\text{Bound}_1$ =*infinity,* $\text{Bound}_2$ =*12* and *Due*_$\text{Date}_1$ = 18, *Due*_$\text{Date}_2$ = 14

Now in step 4 we are going to move each group to the right so its bth job, according to property IV, finishes as close as possible to the group's due date.

**Step 4. Displace groups to the right to reduce ET.**

```
i = 1
    Set b = first job of group 1, according to property IV.
Set Cb= 13
Bound1 =infinity, then we right shift Jobs in set 1 a distance of 18 – 13 = 5
```

So far the schedule would look like the one in Figure 14.

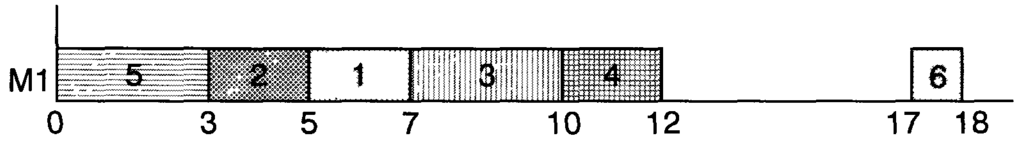


**Figure 14. Partial schedule for the problem in Table 3**

*i=2*

Set *b* = second job of group *2*, according to property IV

Set *Cb= 10*

$\text{Bound}_i$ is not *infinity then*

$\text{Bound}_2$ = *Start_time_of(Job[Position_of(job 4) + 1])=17*

Set *PTA* = Summation of processing times after second job in set *2,* according to property IV

Therefore PTA=2

17 ≥ *14+ 2* then we right Shift Jobs in set *2* a distance of *14 – 10 = 4*

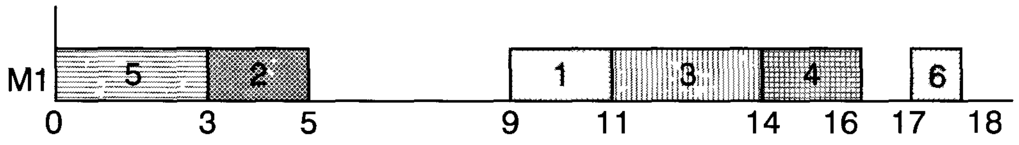


**Figure 15. Final schedule for the problem in Table 3**

*i=3, then we stop.*

The final schedule will be as shown in Figure 15.

## 5.2 Extending the Earliness Reduction Procedure for the Rm/ /ET problem

A schedule for a parallel machine problem allocates each job to a machine and determines its start and completion times on that machine. Using such a schedule as an input, the Earliness Reduction Procedure is performed to improve the ET value in a similar way as for the 1/ /ET problem, except that in this case the procedure is applied to the schedule of each machine independently as shown in the following pseudo code.

```
For each machine m
        Set Number_of_jobs = Number of jobs allocated to m
        Perform Earliness Reduction Procedure for schedule of machine m
End for.
```

## 5.3 Extending the Earliness Reduction Procedure for the Jm/ /ET problem

A job shop schedule is expected to have idle time between some operations even when the objective function is a regular measure as some operations will be dependent on the completion of preceding operations. Similar to the parallel machine problem, the Earliness Reduction Procedure is performed for each machine independently, but in this case each machine will have scheduled some operations that are the last ones of their job and some that are not. If we right shift one operation that is *not* the last one of its job, additional computational work will be required to find out the time up to which the operation can be shifted without affecting the schedule's feasibility, and the move by itself will not improve the value of the ET. If, on the other hand, we right shift one or more groups of operations that are the last ones of their jobs, after having identified the group's boundary, this will not affect the feasibility of the schedule and may in fact improve (decrease) the ET value.

Therefore, in this particular production environment, we need to include a fourth characteristic to form groups of operations in the adjusting procedure. Consequently, the procedure consists of exploring all operations from right to left in the Gantt chart, identifying groups of them that share the following characteristics:

1. Are adjacent.
2. Belong to jobs with the same (common) due date.
3. Belong to jobs that complete early or at the common due date.
4. Are the last operation of its job.

Taking into account the four characteristics, the pseudo code for the $Jm/\ /ET$ problem is the same as the one for the $Rm/\ /ET$ case presented in the former section.

## 5.4 Conclusion

We studied some characteristics of the $1/\ /ET$ problem when the due date of all jobs is common. Based on that, we proposed an Earliness Reduction Procedure to reduce the $ET$ to be performed on a schedule produced for a single machine, parallel machine, flow shop or job shop problem when the objective function is the $ET$ and the due dates may be distinct.

Notice that the input schedule to perform the Earliness Reduction Procedure is not necessarily one produced by the Random Key Genetic Algorithm (RKGA) (which will be explained in the next chapter) but a feasible one. This makes the procedure more general and usable by other scheduling models once an initial schedule has been produced.

# 6 A GENERIC RANDOM KEYS GENETIC ALGORITHM FOR SHOP SCHEDULING PROBLEMS

A wide variety of exact and heuristic methods exists in the literature to address specific scheduling problems for specific environments, objective functions and problem characteristics.

Among the great body of literature dedicated to scheduling problems with Regular Performance Measures, the work of Bean (1994) and Norman and Bean (1997) stands out as their approach can be used to solve scheduling problems in different production settings. Bean (1994) proposed a Random Keys Genetic Algorithm (RKGA) encoding to solve single and parallel machine problems, while Norman and Bean (1997) proposed another version for the classical Job Shop Scheduling Problem considering the regular performance measure of the makespan. In this chapter we present a generalization of the RKGA approach to address single machine, parallel machine, flow shop and job shop problems when the objective function is a regular measure. Then, a connection will be made with the Earliness Reduction Procedure discussed in Chapter 5 to consider the non-regular measure of Earliness and Tardiness.

## 6.1 Random Key Genetic Algorithms (RKGA)

Genetic Algorithms (GA) is a search technique used to solve optimization problems based on the principles of natural selection. Many versions of GA have been proposed to solve scheduling problems based on different solution representations (schedules) by means of diverse types of chromosomes, which encode the genetic information of an individual or solution. Once a representation is established, a fitness function is required to evaluate the quality of each solution. The process starts by generating an initial population of individuals, evaluating their fitness, and repeatedly applying a set of genetic operators to produce new solutions (offspring). Based on fitness values, the selection operator probabilistically chooses individuals for inclusion in the next generation and/or as parents. The crossover operator recombines parents to produce offspring that will form the next generation. In order to improve the species, the fittest individuals are preferred to be recombined. Through a mutation operator, some individuals are randomly altered to guarantee population diversity. The evolution process continues until the GA converges to its best solution.

Despite the diverse GA approaches that have been used to solve scheduling problems based on different representations, most of them have the weakness of producing infeasible solutions after applying the crossover operator to recombine partial solutions, especially for complex environments like Job Shops, where not all jobs have the same route through the machines. The RKGA approach has the advantage that all offspring produced after the crossover operations are feasible solutions. Another advantage of RKGA is that schedules for different environments can be represented in a generic fashion, which will make it possible to represent scheduling problems in various production environments without the need to have too many customized representations for the different problems under consideration. Therefore, we favored the use of RKGA as opposed to other meta-heuristics for our Predictive Reactive scheduling algorithms. The selection of RKGA has also been supported by the fact that it has recently been successfully implemented for different scheduling problems such as the work by Valente and Gonçalves (2009) for the single machine problem of minimizing the earliness and quadratic tardiness, as well as the work by Okada et al. (2009) and Mendes, Gonçalves, and Resende (2009) for project scheduling.

## 6.2 RKGA representation and Decoding Procedure

RKGA representation encodes solutions with uniformly generated random numbers between 0 and 1 called *random keys*. For a single machine problem with $n$ jobs, for example, a chromosome consists of $n$ random numbers between 0 and 1, one for each job. A decoding procedure is used to find the schedule that corresponds to a chromosome. Such a decoding procedure for a single machine problem consists of sorting the random keys in ascending order of their random keys. Starting jobs processing in that order as early as possible will produce a schedule where the objective function can be evaluated to calculate the chromosome's fitness.

As an example, suppose we have four jobs to schedule on a single machine with processing times *3, 4, 5* and *2* for jobs *1, 2, 3* and *4* respectively. According to the representation, a possible chromosome for the problem will be:

| | | | | |
|---|---|---|---|---|
| Job | 1 | 2 | 3 | 4 |
| Random Keys | 0.682 | 0.726 | 0.096 | 0.084 |

After ordering the random keys in ascending order we obtain the sequence: 4 – 3 – 1 – 2. Assuming that all jobs are processed as early as possible the corresponding schedule can be represented in the Gantt chart in Figure 16, where M1 is the only machine in this problem:

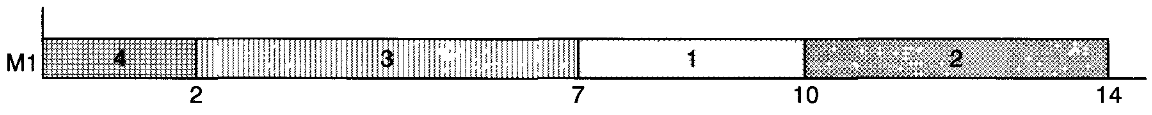


**Figure 16. Schedule for a single machine problem**

In parallel machine problems, there is a set of $n$ one-operation jobs to be processed on $m$ parallel machines which can be identical, with different speeds irrespective of the jobs, or with a speed depending on the job.

To create a chromosome that encodes a solution for a $n$-job, $m$-machine problem, we generate, for each job, a random integer number between 1 and $m$ and add to it a uniform random (0,1) number. While decoding the chromosome, the integer number will represent the machine to which the operation is assigned, and the fractional part will determine the order on each machine in the same way as for the single machine problem. Processing the jobs in that order as early as possible on their corresponding machines will produce a schedule from which the chromosome's fitness is calculated.

In more complex cases like the Job Shop Scheduling Problem (JSSP), a job consists of multiple operations that follow a specific route and different jobs may have different routes. A chromosome is formed by pairs of uniformly generated random numbers between 0 and 1, a pair for each operation. As in the previous cases, the first number, the random key, is used as a sorting key to decode a solution. At each step the decoding procedure takes the random key of the next unscheduled operation of each job and chooses the one corresponding to the lowest value and schedules it next. The second number, called the *delay factor*, gives the possibility of exchanging the winner operation with respect to the random key value criterion, with another operation competing for the same machine since that could produce a better schedule at the end. The resulting solution will belong to the set of semi active schedules and eventually to the subset of active schedules. In a semi active schedule it is not possible to start any operation earlier without altering the sequence on any machine. In an active schedule no operation can be started earlier without either delaying some other operation or violating the constraints (French, 1982). In other words, it is not possible to drag an operation and drop it in a hole, earlier in the schedule keeping the feasibility. That characteristic of the decoding procedure is especially useful for regular measures of performance (or objective functions), since it is known that the set of semi active schedules is dominant for them (Baker and Trietsch, 2009).

As an example, consider a 2 job, 3 machine job shop problem where the objective is to minimize $C_{max}$, $(J3/ /C_{max})$. Each job is ready to be processed at time zero, and all machines are continuously available. The processing time and machine routing of each operation of each job are given in Table 4. In this case job 1's route through the machines is: 1-3-2, and job 2's route is 3-2-1.

**Table 4. A 2 x 3 Job Shop problem**

| Job | Processing Time(Machine) | | |
|---|---|---|---|
| 1 | 2(1) | 3(3) | 2(2) |
| 2 | 3(3) | 4(2) | 1(1) |

An example of a chromosome for the problem is given below:

| | | | | | | |
|---|---|---|---|---|---|---|
| (Job, Operation) | (1, 1) | (1, 2) | (1, 3) | (2, 1) | (2, 2) | (2, 3) |
| Random Key | 0.039 | 0.634 | 0.075 | 0.141 | 0.901 | 0.857 |
| Delay Factor | 0.231 | 0.553 | 0.732 | 0.593 | 0.489 | 0.934 |

Since a job has different operations which may be processed on different machines, in order to distinguish them the triple (*i, j, k*) is used, where *i* represents the job, *j* the operation and *k* the machine.

The random key of each operation will be presented in brackets [ ]. The procedure to decode the previous chromosome in a solution is as follows. Initially the first operation of all jobs is programmable, i.e. ready to be scheduled. So, among them we select the one with the minimum random key. In this case we have two programmable operations (1, 1, 1) [ 0.039] and (2, 1, 3) [0.141]. The minimum random key is 0.0390 corresponding to (1, 1, 1). The operation does not create idle time on its machine, so it is programmed to start at time 0 and end at time 2 (since the processing time is 2). It is then removed from the set of programmable operations.

The new set of programmable operations is formed by (1, 2, 3) [0.634] and (2, 1, 3) [0.141]. The candidate operation this time is (2, 1, 3) which does not create idle time on its machine, so it is programmed from time 0 to 3.

The new programmable operations are (1, 2, 3) [0.634] and (2, 2, 2) [0.901]. The candidate operation this time is (1, 2, 3). The operation does not create idle time on its machine; therefore it is programmed from time 3 to 6.

The new programmable operations this time are: (1, 3, 2) [0.075] and (2, 2, 2) [0.901], and the candidate operation is (1, 3, 2). If we scheduled such an operation it would start at 6 and complete at 8 creating idle time on machine 2 from 0 to 6. Since the candidate operation would create some idle time the Move Search is invoked. Such a procedure aims to improve the quality of the resulting schedule. We check if there are other programmable operations $o_{(l,j,k)}$ on the same machine as the candidate operation. That is the case of operation (2, 2, 2). Then we check the condition of the move search to replace the candidate operation:

If $S_{(l,j,k)} + DF_{(l,j,k)} * p_{(l,j,k)} < S_{CO}$ then $o_{(l,j,k)}$, becomes the candidate operation

Where:

$S_{(l,j,k)}$ is the start time of the programmable operation,

$DF_{(l,j,k)}$ is the delay factor of the programmable operation,

$p_{(l,j,k)}$ is the processing time of the programmable operation,

$S_{CO}$ is the starting time of the candidate operation.

In our case we have: $(l,j,k)$ is (2, 2, 2), $S_{(2,2,2)} = 3$, $DF_{(2,2,2)} = 0.489$, $p_{(2,2,2)} = 4$ and $S_{CO} = 6$. The condition holds, and (2, 2, 2) becomes the candidate operation and is scheduled from 3 to 7.

The new set of programmable operations is formed by (1, 3, 2) [0.075] and (2, 3, 1) [0.857]. The candidate operation this time is (1, 3, 2). If we schedule such an operation, it would start at 7 and complete at 9, not creating idle time on machine 2, which means it is programmed.

Finally, the only remaining programmable operation (2, 3, 1) is scheduled from 7 to 8. The resulting Gantt chart is shown in Figure 17.

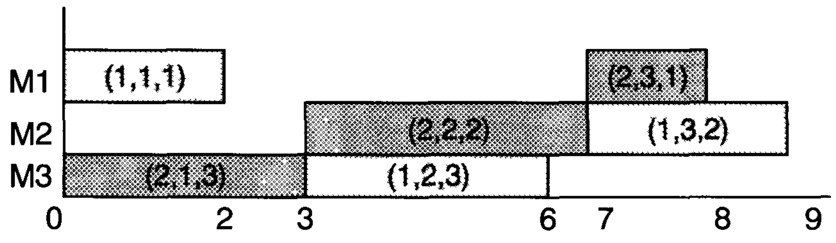


**Figure 17. Schedule for the chromosome on Table 4**

## 6.3 A generalized Random Keys representation for Scheduling Problems

To have a representation that embraces the four production settings targeted in this dissertation, a chromosome must allow for recording of information for the different operations of a job in case there are multiple operation jobs, including their machine allocation in the case of parallel machines.

Let us define for each operation, a machine list Lo as:

$$\text{Lo} = (m_1, m_2, \ldots m_{to}) \qquad (5)$$

where $Lo$ is the set of machines in which operation $o$ can be processed, and $to$ is the total number of machines in which operation $o$ can be processed.

A chromosome will be formed by triplets of random numbers for each operation $o$, namely, the machine key, the random key and the delay factor. The machine key is a random integer number between 1 and $to$ that will determine a position in the operation's machine list, thus the machine selected to process the operation. The random key and the delay factor are defined and used in the same way as discussed earlier. As an example, suppose we have five jobs to process on two parallel identical machines (same speed). The processing times are given in Table 5.

**Table 5. A 5-job 2 parallel machine**

| Job | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| Processing time | 3 | 4 | 5 | 2 | 4 |

According to the introduced representation, since all jobs can be processed on either of the two machines, the machine list for each operation will be the same: (1, 2), and the equivalent chromosome will be as follows:

| | | | | | |
|---|---|---|---|---|---|
| Job | 1 | 2 | 3 | 4 | 5 |
| Machine Key | 2 | 1 | 1 | 1 | 2 |
| Random Key | 0.548 | 0.380 | 0.693 | 0.639 | 0.497 |
| Delay Factor | 0.448 | 0.973 | 0.397 | 0.392 | 0.732 |

Note that as all jobs have one operation, no idle time is created when scheduling each of them; therefore, it is not necessary to use the delay. The resulting schedule is the same as the one in Figure 18.

This representation has an advantage. Consider a job shop in which there may be one or several machines of the same type. That is, jobs may have several operations that may be processed over a set of parallel machines. In this case we can think of a job shop with different work centers with a series of parallel machines in each one. This is a more general environment called in the literature a Flexible Job Shop (Pinedo, 2008). Since we are considering both the list of machines for each operation and the random keys and delay factors, the representation can be used with Flexible Job Shop Problems as well.

As an example, consider the job shop problem of Table 4, but in this case each operation can be processed on the machines listed as shown in Table 6.

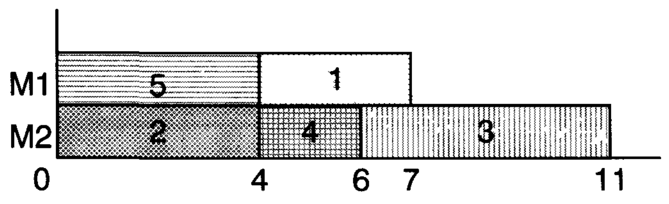


**Figure 18. Schedule for a 5-job 2-parallel machine problem**

**Table 6. A 2 x 3 Flexible Job Shop problem**

| Job | Processing Time(Machine List) | | |
|---|---|---|---|
| 1 | 2(1,5,6) | 3(3,4) | 2(2,7) |
| 2 | 3(3,4) | 4(2,7) | 1(1,5,6) |

According to the generalized representation, a possible chromosome for the problem is shown next:

| | | | | | | |
|---|---|---|---|---|---|---|
| (Job, Operation) | (1, 1) | (1, 2) | (1, 3) | (2, 1) | (2, 2) | (2, 3) |
| Machine Key | 2 | 1 | 1 | 1 | 1 | 2 |
| Random Key | 0.039 | 0.634 | 0.075 | 0.141 | 0.901 | 0.857 |
| Delay Factor | 0.231 | 0.553 | 0.732 | 0.593 | 0.489 | 0.934 |

That corresponds with the machine allocation presented in Table 7.

**Table 7. Machine selection for the problem of Table 6**

| Job | Operation | | |
|---|---|---|---|
| 1 | 5 | 3 | 2 |
| 2 | 3 | 2 | 5 |

Once we know the machine selection for each operation, we can follow the decoding procedure explained for the case of job shop problems. The resulting schedule is presented in Figure 19.

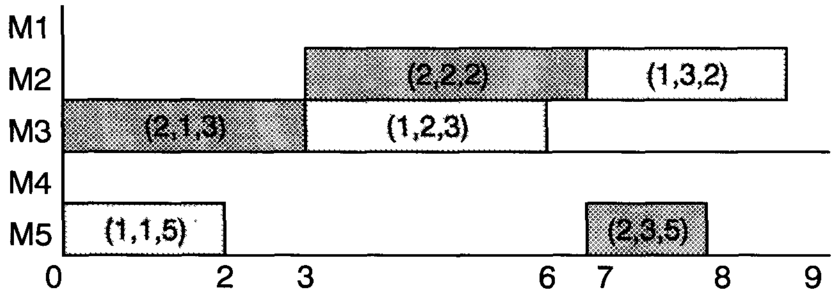


**Figure 19. A schedule for the problem on Table 6**

Notice that the decoding procedure will build a feasible solution based on the chromosome for any objective function, regular or non-regular, that can be evaluated. However, the solution produced is a semi active schedule, which will compact to the left. This is advantageous for regular measures but may not always be advantages for non-regular measures as was explained in chapter 5 for the case of ET where the optimal solution may not necessarily be a schedule that is compact to the left. Therefore, when the objective is to minimize ET, the Earliness Reduction Procedure is applied to the schedule produced by the decoding procedure before evaluating the fitness of a chromosome.

## 6.4 Incidence of the Move Search procedure to solve Jm/ /ET problems

As explained previously, the Move Search is a procedure invoked every time a candidate operation, if scheduled, would create idle time on its respective machine. Depending on the Delay Factor value, if there are other programmable operations that could start before the candidate operation, one of them may become the new candidate and be scheduled before the former candidate operation, even if it delays the start of the former candidate. However, if one of these operations can start and complete prior to the start of the candidate operations, it will then be scheduled first regardless of its delay factor value due to the condition of the Move Search (If $S_{(l,j,k)} + DF_{(l,j,k)} * p_{(l,j,k)} < Sco$ then $o_{(l,j,k)}$, becomes the candidate operation). Note that If $S_{(l,j,k)} + p_{(l,j,k)} < Sco$, then the condition will always hold; thus, $o_{(l,j,k)}$ becomes the candidate operation.

The way in which the Move Search is designed implies that for certain problems some sequences will never be produced by the RKGA. As an example, consider the small J2/ /ET problem instance in Table 8.

**Table 8. A J2/ /ET Job Shop problem**

| Job | Processing Time(Machine) | | Due date |
|---|---|---|---|
| 1 | 3(1) | 4(2) | 9 |
| 2 | 2(2) | | 9 |

According to the RKGA decoding procedure, the first operation to schedule will be either (1, 1, 1) or (2, 1, 2) since (1, 2, 2) is the second one of its job.

If (1, 1, 1) is scheduled first, either (2, 1, 2) or (1, 2, 2) may be the next candidate. If (2, 1, 2) is the candidate it will be scheduled first on machines 2 and the final sequence on that machine will be (2, 1, 2) - (1, 2, 2) as shown in Figure 20.

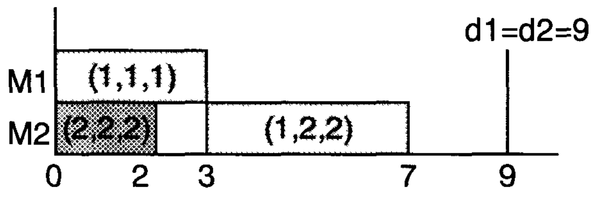


**Figure 20. Schedule for the J2/ /ET problem of Table 8**

If (1, 1, 1) is scheduled first but (1, 2, 2) is the next candidate, the condition for the Move Search procedure holds in this case: $S_{(2,1,2)} + DF_{(2,1,2)} * p_{(2,1,2)} < S_{(1,2,2)}$ that is: $0 + DF_{(2,1,2)} * 2 < 3$ (recall that by definition $0 < DF < 1$). Therefore, (2, 1, 2) is scheduled first, and the final sequence on machine 2 will be again (2, 1, 2) - (1, 2, 2), as shown in Figure 20.

If (2, 1, 2) is scheduled first, it will start at time 0 on machine 2, then (1, 1, 1) and (1, 2, 2) will be scheduled in that order since they belong to the same job. Consequently, the final schedule will correspond to the one shown in Figure 20.

In conclusion, for this problem, regardless of the random keys and delay factors that are randomly generated, the schedule yielded by the RKGA will be the same in all cases. Notice that this schedule is active, which is very convenient for the case of regular objective functions. However, if we consider the ET measure and therefore apply the Earliness Reduction Procedure to such a schedule, we obtain only one group, formed by operations (2, 1, 2) and (1, 2, 2), which will be compacted and right shifted until operation (2, 1, 2) completes at time 9, the group's common due date, as shown in Figure 21.

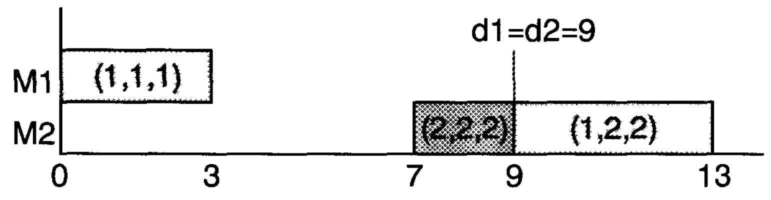


**Figure 21. Resulting schedule after Earliness Reduction Procedure for figures 15's schedule**

As can be seen, the ET value for this schedule is 4. However, this is not the optimal solution. If we schedule (1, 2, 2) from 5 to 9 and (2, 1, 2) from 9 to 11 the ET value is 2. But as was just shown, the sequence (1, 2, 2) - (2, 1, 2) will not be yielded by the RKGA since the Move Search procedure will prevent some solutions from being obtained. To avoid that some potentially optimal solutions are discarded for ET problems in this work; the decoding procedure does not perform the Move Search if the objective function is ET.

In the following section the genetic operators and the GA dynamics are explained.

## 6.5 Crossover

The crossover operator recombines two parents P1 and P2 to produce two offspring, C1 and C2, that compete to be included in the next generation. As was stated earlier, one of the main advantages of the RKGA approach is that all offspring produced after the crossover operations are feasible solutions. The presented generalized representation maintains that characteristic.

Specifically, parameterized uniform crossover is applied in the RKGA as used by Norman and Bean (1997). In this type of crossover, a uniformly distributed (0, 1) random number is generated for each operation of the problem. If the number is greater than a value called the *crossover probability* (0.7 in this work), the first child solution, C1, will have the same triplet (machine key, random key and delay factor) as P1 for this operation, and child solution C2 will have the same triplet as P2 for this operation. Otherwise, C1 takes P2's triple, and C2 takes P1's triple. An example with two randomly generated chromosomes for the previous flexible job shop problem is presented below. The Random Numbers shown after the two parent chromosomes correspond to the value to compare against the crossover probability. The value for this probability is taken from Norman and Bean (1997).

| | | | | | | |
|---|---|---|---|---|---|---|
| Parent 1 | | | | | | |
| Machine Key | 3 | 1 | 1 | 1 | 1 | 2 |
| Random Key | 0.591 | 0.790 | 0.930 | 0.130 | 0.687 | 0.934 |
| Delay Factor | 0.995 | 0.082 | 0.939 | 0.007 | 0.882 | 0.851 |
| Parent 2 | | | | | | |
| Machine Key | 2 | 2 | 1 | 2 | 1 | 1 |
| Random Key | 0.786 | 0.987 | 0.570 | 0.166 | 0.694 | 0.743 |
| Delay Factor | 0.990 | 0.799 | 0.239 | 0.600 | 0.929 | 0.041 |
| | | | | | | |
| Random Number | 0.176 | 0.996 | 0.719 | 0.378 | 0.341 | 0.590 |

| | | | | | | |
|---|---|---|---|---|---|---|
| Offspring 1 | | | | | | |
| Machine Key | 2 | 1 | 1 | 2 | 1 | 1 |
| Random Key | 0.786 | 0.790 | 0.930 | 0.166 | 0.694 | 0.743 |
| Delay Factor | 0.990 | 0.082 | 0.939 | 0.600 | 0.929 | 0.041 |
| Offspring 2 | | | | | | |
| Machine Key | 3 | 2 | 1 | 1 | 1 | 2 |
| Random Key | 0.591 | 0.987 | 0.570 | 0.130 | 0.687 | 0.934 |
| Delay Factor | 0.995 | 0.799 | 0.239 | 0.007 | 0.882 | 0.851 |

Notice that any chromosome will inherit a valid machine key, random key and delay factor, therefore, all offspring will represent feasible solutions.

After generating both offspring, fitness is evaluated, and the best one is selected to become part of the next generation.

## 6.6 Selection

The selection operator acts in one of two forms. In the first form, pairs of individuals are randomly selected from the entire population to produce two offspring by using the crossover operator. The fittest one is selected to become part of the next generation. Therefore, there is no guarantee that individuals with the best fitness found so far will survive to each subsequent generation. To overcome this problem, there is a second form of selection, used as a complement in which a certain percentage of the fittest individuals are selected directly to survive to the next generation. This second form is referred to as reproduction. As a consequence, the best solution improves (decreases) monotonically. The fittest individuals of the current population are selected to survive to the next generation by reproduction. The percentage of the fittest individuals that will survive will be referred to as the *Reproduction_%*. Another percentage of the next generation, that we will call *Cross_%*, is created by crossover as was explained.

## 6.7 Mutation

The objective of mutation is to diversify the species. Traditionally, mutation is done by altering part of the genetic information of some individuals. Following the line of Bean (1994)'s work, in order to add diversity to the population, a percentage of new individuals, which we will refer to as *Mutation_%*, is created to become part of the next generation. This type of mutation is sometimes called immigration.

## 6.8 Stopping Criteria

Two stopping criteria are used here:

1. A limit of generations (*Generations)* is completed.
2. The limit of iterations without an improvement (*IWI*) in the best solution found so far has been reached.

## 6.9 Evaluation of fitness

The fitness of all individuals is evaluated by using the decoding procedure to build the schedule. If the objective function is ET, the Move Search is not performed while decoding the chromosome into a schedule and the Earliness Reduction Procedure (discussed in Chapter 5) is applied to the resulting schedule. Otherwise, the Move Search is performed, and the Earliness Reduction Procedure is not invoked.

## 6.10 Overall view of the GA

The GA uses two inputs: the problem data and the parameter values (*Reproduction_%, Cross_%, Mutation_%, Generations, IWI, crossover probability)* and creating an initial population (Recall that *IWI* is the limit of iterations without an improvement). The fitness of all individuals is evaluated. Then the population is subjected to the genetic operators of reproduction, crossover and mutation until the stopping criterion is reached. Pseudo code for the algorithm is presented below.

## 6.11 Pseudo Code of the RKGA

```
Generate initial population
Evaluate fitness of all individuals
Order the population based on their fitness
Repeat until the stopping criterion is met
    Copy the Reproduction_% best individuals to next generation
```

```
        Generate Cross_% individuals by crossover. Copy them to next generation
        Generate Mutation_% individuals. Place them in the next generation
        Evaluate fitness of new individuals
        Order the population based on their fitness
End Repeat
Return the individual with the best global objective fitness
```

## 6.12 Conclusions

We have reviewed the RKGA approach for scheduling problems where the objective function is a regular measure. We presented an integrated encoding approach to be used in the four different production settings targeted in this dissertation. Then we explained how to connect to the Earliness Reduction Procedure with the RKGA when the objective function is the ET. The final version of the RKGA, which we have called the predictive model, allows for generate predictive (or initial) schedules for problems coming from different environments with regular measures or the non-regular Earliness and Tardiness measures, with common or distinct due dates. The experiments to test this predictive model are presented in the next chapter.

# 7 COMPUTATIONAL EXPERIMENTS FOR THE PREDICTIVE MODEL

As discussed in earlier chapters, two models are presented in this dissertation. What we have called the "predictive model" to the generalized RKGA presented on Chapter 6, creates a predictive (or initial) schedule. A second RKGA, called the "reactive model", which will be introduced in Chapter 8, produces a schedule in response to a disruption. Both models were implemented in Visual Basic 2008 and tested on a 2.5 GHz Intel Core Quad running Windows Vista.

Since the model can be used in four different basic production environments, for regular and non-regular measures, two representative types of problems were selected to test its quality. The first problem consists of minimizing the maximum completion time of a set of one operation jobs over a set of unrelated parallel machines, which are machines that are capable of processing any of the available jobs but the processing times for the same job may differ from one machine to another. In Graham's notation the problem is represented as $\mathrm{Rm}/\ /\mathrm{Cmax}$. The second problem, a Job Shop type, consists of minimizing the total Earliness and Tardiness for a set of multiple-operation jobs which have to be processed on a set of machines, each of which is unique in the shop. In Graham's notation the problem is represented as $\mathrm{Jm}/\ /\mathrm{ET}$. The problem is similar to the one presented in the example in section 2.5 but the due dates are not the same for all jobs.

The first type of problem is representative in the sense that it will let us know how the predictive model behaves in the most general case of parallel machine environments with a regular measure such as the makespan ($\mathrm{Cmax}$). The second type of model is representative in the sense that it will let us know how the predictive model behaves in the more general case of jobs with multiple operations, such as the Job Shop Scheduling Problem, when using the non-regular measure of Earliness and Tardiness, which will let us test the quality of the Earliness Reduction Procedure proposed on chapter 5 to solve problems with such objective functions.

## 7.1 Unrelated Parallel Machines Problems

Sets of problems with two, four and six machines and 20, 40 and 60 jobs, with three problem instances per problem size, were generated. Using the same distribution as in Martello, Soumis and Toth (1997) and Arnaout (2006), the processing times were randomly generated following a discrete uniform distribution U[1,9].

Based on preliminary tests and the work of Bean (1994), the reproduction, crossover, and immigration rates were set as follows. A reproduction rate of 20% is used. 84% of the next generation is obtained by parameterized uniform crossover as explained in Chapter 6. 6% of the population is mutated by applying the concept of immigration, where at each generation a certain number of individuals is randomly generated to become part of the new generation. The parameter values for the stopping criteria and population size, found to be appropriate by Norman and Bean (1997), are used here. The GA stops when a maximum number of 250 iterations has been reached or 75 generations have passed without any improvement of the best solution found so far. Norman and Bean (1997) used for $Jm/\ /Cmax$ problems a population size of 300 plus two times the number of operations to be scheduled. Pilot tests on representative problems for the $Rm/\ /Cmax$ problem showed that a size of 300 individuals plus one time the number of operations to be scheduled is enough for the problem set under consideration. Therefore, since $Rm/\ /Cmax$ problems are composed of one-operation jobs, for problems with 20, 40 and 60 jobs the population size was set to 320, 380 and 420 individuals respectively. Each instance was run for 15 replications using the previously mentioned parameters.

## 7.2 Integer programming formulation for $Rm/\ /Cmax$

The $Rm/\ /Cmax$ problem can be formulated as follows (Potts, 1985).

Objective: Minimize $Cmax$

Subject to

$$\sum_{i=1}^{n} p_{ij} * x_{ij} \leq Cmax, \forall i = 1, \dots, m \qquad \text{(C1)}$$

$$\sum_{i=1}^{m} x_{ij} = 1, \forall j = 1, \dots, n \qquad \text{(C2)}$$

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

Where:

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

$x_{ij}$: binary decision variable = $\begin{cases} 1 \text{ if job } j \text{ is assigned to machine } i \\ 0 \text{ Otherwise} \end{cases}$

Constraints (C1) guarantee that the makespan is at least as large as the total completion time of any machine.

Constraints (C2) and (C3) ensure that each job will be assigned exactly to one machine.

The previous mixed integer programming model was used to formulate and obtain the optimal solution of all the generated instances in LINGO 12.0, a tool provided by Lindo Systems, Inc.

## 7.3 Results for the Rm/ /Cmax problem

Table 9 summarizes the results over the 15 runs for each of the benchmark problems. The fourth column presents the optimal solution obtained in LINGO. The Best found (the fifth column) is the best solution found over the 15 runs. The deviation from the optimum (OF Deviation) for each run is calculated as:

$$\text{OF Deviation} = \frac{[\text{RKGA Solution} - \text{Optimal Solution}] \text{x } 100}{\text{Optimal Solution}} \quad (6).$$

The values in the sixth and seventh columns correspond to the average and standard deviation of the OF Deviation calculated over the 15 runs according to equation 6 above. The last two columns report the average and standard deviation of the runtime over the 15 runs of each instance.

For the instances tested, the deviation from the optimal value is, in all cases, no greater than 3.8%. The optimal value was found in at least one run for all problems. The RKGA behaves very well for problems with two machines finding the optimal solution in all runs for all problems. As the number of machine increases, and since there are more feasible solutions, the RKGA takes more iterations and, consequently, more time to find a final solution.

Table 10 presents the average deviation values for the different sizes of the Rm/ /Cmax problem, and we see that the overall average deviation from the optimum is 0.9%.

Although for the studied problem sizes the instances are solved by LINGO in a second, we should take into account that the problem is NP-hard. Furthermore, for other objective functions such as total tardiness (Rm/ /T) and maximum lateness (Rm/ /Lmax) the problem is NP-hard as well, and the use of mathematical programming is computationally demanding and the results not practical even for small problem instances, as reported by Pfund, Fowler and Gupta (2004). On the other hand, the computational effort taken by the RKGA to run using such objective functions instead of Cmax, is similar, since calculating the fitness value consists of a sequential search on an array with a size equal to the number of jobs in all cases.

**Table 9. Results for the Rm/ /Cmax problem**

| Jobs | Machines | Problem number | Optimal Value | Best found | Deviation from Optimum (%) Average | Deviation from Optimum (%) Std. Dev. | Runtime (minutes) Average | Runtime (minutes) Std. Dev. |
|---|---|---|---|---|---|---|---|---|
| 20 | 2 | 1 | 39 | 39 | 0.0 | 0.0 | 3.8 | 0.1 |
| | | 2 | 44 | 44 | 0.0 | 0.0 | 4.0 | 0.1 |
| | | 3 | 39 | 39 | 0.0 | 0.0 | 3.9 | 0.1 |
| | 4 | 4 | 11 | 11 | 0.0 | 0.0 | 4.5 | 0.2 |
| | | 5 | 17 | 17 | 0.0 | 0.0 | 4.5 | 0.1 |
| | | 6 | 14 | 14 | 0.5 | 1.8 | 4.7 | 0.2 |
| | 6 | 7 | 7 | 7 | 3.8 | 6.5 | 5.4 | 0.9 |
| | | 8 | 7 | 7 | 1.0 | 3.7 | 5.1 | 0.3 |
| | | 9 | 11 | 11 | 3.0 | 4.4 | 5.1 | 0.4 |
| 40 | 2 | 10 | 57 | 57 | 0.0 | 0.0 | 11.0 | 0.2 |
| | | 11 | 78 | 78 | 0.0 | 0.0 | 11.3 | 0.6 |
| | | 12 | 67 | 67 | 0.0 | 0.0 | 11.2 | 0.5 |
| | 4 | 13 | 27 | 27 | 0.0 | 0.0 | 14.1 | 0.6 |
| | | 14 | 24 | 24 | 1.4 | 2.0 | 15.0 | 1.2 |
| | | 15 | 21 | 21 | 1.6 | 2.3 | 14.5 | 0.8 |
| | 6 | 16 | 10 | 10 | 0.0 | 0.0 | 14.6 | 0.4 |
| | | 17 | 13 | 13 | 0.0 | 0.0 | 15.7 | 0.7 |
| | | 18 | 14 | 14 | 3.8 | 3.7 | 17.5 | 2.5 |
| 60 | 2 | 19 | 109 | 109 | 0.0 | 0.0 | 22.5 | 0.8 |
| | | 20 | 118 | 118 | 0.0 | 0.0 | 23.9 | 0.8 |
| | | 21 | 107 | 107 | 0.0 | 0.0 | 22.5 | 0.5 |
| | 4 | 22 | 37 | 37 | 0.9 | 1.3 | 28.7 | 2.3 |
| | | 23 | 39 | 39 | 0.9 | 1.3 | 29.7 | 2.2 |
| | | 24 | 33 | 33 | 2.6 | 1.1 | 28.3 | 1.4 |
| | 6 | 25 | 21 | 21 | 2.5 | 2.5 | 32.6 | 2.9 |
| | | 26 | 18 | 18 | 0.7 | 2.0 | 32.0 | 2.8 |
| | | 27 | 18 | 18 | 0.4 | 1.4 | 35.3 | 3.1 |

**Table 10. Average Results for Rm/ /Cmax problem**

| Jobs | Machines | Deviation from Optimum (%) Average | Deviation from Optimum (%) Std. Dev. | Runtime (minutes) Average | Runtime (minutes) Std. Dev. |
|---|---|---|---|---|---|
| 20 | 2 | 0.0 | 0.0 | 3.9 | 0.1 |
| | 4 | 0.2 | 0.6 | 4.6 | 0.2 |
| | 6 | 2.6 | 4.9 | 5.2 | 0.5 |
| 40 | 2 | 0.0 | 0.0 | 11.2 | 0.4 |
| | 4 | 1.0 | 1.5 | 14.5 | 0.9 |
| | 6 | 1.3 | 1.2 | 16.0 | 1.2 |
| 60 | 2 | 0.0 | 0.0 | 23.0 | 0.7 |
| | 4 | 1.5 | 1.2 | 28.9 | 2.0 |
| | 6 | 1.2 | 1.9 | 33.3 | 2.9 |
| Average | | 0.9 | 1.3 | 15.6 | 1.0 |

## 7.4 Results for the Jm/ /ET problem

Three sets of problems with five jobs and five machines, seven jobs and seven machines, and nine jobs and nine machines and with 12 problem instances per set were generated. The processing times were randomly generated following a discrete uniform distribution U[1,9] as in Demirkol, Mehta and Uzsoy (1998). In order to have all jobs visiting all machines in some random order, the job routes were generated from another discrete uniform distribution U[1,*m*] where *m* is the number of machines.

As in Blocher, Chhajed and Leung (1998) the due dates are set as a multiple of the total job's processing time. Such multiple determines the tightness of the due date. Three levels of tightness were used to calculate the due dates of the jobs: 3, 1.4 and 0.7. The first one is a loose due date factor The last multiple, 0.7 is considered a tight due date factor in the sense that guarantees in advance that the job is going to be late in any solution. The third multiple, 1.4, was chosen arbitrarily as something in between, for which there is no certainty that the job will be able to complete on time or not. Among the 12 instances of each problem size, four problems with each of the three tightness levels were solved.

## 7.5 Integer programming formulation for Jm/ /ET

Based on the disjunctive graph representation of the Jm/ /Cmax problem (see Pinedo (2008)), the Jm/ /ET problem can be formulated as follows.

Objective: Minimize $\sum_{j=1}^{n} ET_j$

Subject to

$y_{hj} - y_{ij} \geq t_{ij} \quad \forall (i,j) \rightarrow (h,j) \epsilon\, A,$ (C4)

$C_j - y_{ij} = t_{ij} \quad \forall (i,j) \epsilon\, N,$ (C5)

$y_{ij} - y_{ik} \geq t_{ik} \text{ or } y_{ik} - y_{ij} \geq t_{ij} \quad \forall (i,k) \text{ and } (i,j), i = 1, \ldots m,$ (C6)

$C_j - d_j \leq ET_j \quad \forall j = 1, \ldots, n,$ (C7)

$d_j - C_j \leq ET_j \quad \forall j = 1, \ldots, n,$ (C8)

$y_{ij} \geq 0 \quad \forall (i,j) \epsilon\, N,$ (C9)

Where:

$y_{ij}$: starting time of operation $(i, j)$, that is, operation on machine $i$, of job $j$.

$t_{ij}$: processing time of operation $i$ of job $j$

$C_j$: completion time of job $j$

$d_j$: due date of job $j$

$ET_j$: Earliness/Tardiness objective for job $j$

$ET$: total Earliness/Tardiness

$N$: Set of all operations $(i, j)$

$A$: Set of precedence constraints $(i, j) \longrightarrow (h, j)$, this denotes two consecutive operations of job $j$.

Constraints (C4) guarantee the precedence relations between any two consecutive operations of each job.

Constraints (C5) express the completion time of any job as its starting time plus its processing time.

Constraints (C6) ensure that there are not overlaps between any pair of operations of different jobs on the same machine. They are implemented in Linear Programming as follows:

$$y_{ij} - y_{ik} \geq t_{ik} - Mb \qquad \text{(C6)}$$

$$y_{ik} - y_{ij} \geq t_{ij} - M(1-b) \quad \text{(C7)}$$

where M is a big constant such that M>$t_{ik}$ and M>$t_{ik}$, and b is an integer binary variable. If b =1, the second constraint is executed and the first one becomes redundant; if b=0, the first constraint is executed and the second one becomes redundant.

Constraints (C7) and (C8) imply the minimization of the absolute deviation between each job's completion time and its due date.

The previous integer programming model was used to formulate and obtain the optimal solution of all the generated instances in LINGO 12.0.

## 7.6 Results for the problem $Jm//ET$

Table 11 presents the results of the 15 runs for each instance. The fourth column presents the optimal solution obtained by LINGO.

The Best found (fifth column) is the best solution found by the RKGA over the 15 runs. The deviation from the optimum is calculated in the same way as explained in equation 6 section 7.3.

Similarly to section 7.3, the values in the sixth and seventh columns correspond to the average and standard deviation of the Objective Function Deviation calculated over the 15 runs. The last two columns report the average and standard deviation value of the runtime over the 15 runs of each instance.

**Table 11. Results for the Jm/ /ET problem**

| Jobs, machines | Due date tightness | Problem number | Optimal Value | Best found | Deviation from Optimum (%) | | Runtime (minutes) | |
|---|---|---|---|---|---|---|---|---|
| | | | | | Average | Std. Dev. | Average | Std. Dev. |
| 5,5 | Loose | 1 | 0 | 0 | 0.0 | 0.0 | 0.30 | 0.10 |
| | | 2 | 0 | 0 | 0.0 | 0.0 | 0.00 | 0.00 |
| | | 3 | 0 | 0 | 0.0 | 0.0 | 0.10 | 0.10 |
| | | 4 | 0 | 0 | 0.0 | 0.0 | 0.10 | 0.10 |
| | Moderate | 5 | 7 | 7 | 0.0 | 0.0 | 4.50 | 0.00 |
| | | 6 | 8 | 8 | 0.0 | 0.0 | 4.40 | 0.10 |
| | | 7 | 15 | 15 | 0.0 | 0.0 | 4.30 | 0.10 |
| | | 8 | 4 | 4 | 0.0 | 0.0 | 4.40 | 0.10 |
| | Tight | 9 | 75 | 75 | 0.0 | 0.0 | 3.90 | 0.10 |
| | | 10 | 67 | 67 | 0.0 | 0.0 | 4.00 | 0.10 |
| | | 11 | 67 | 67 | 0.0 | 0.0 | 3.80 | 0.00 |
| | | 12 | 67 | 67 | 0.0 | 0.0 | 4.00 | 0.20 |
| 7,7 | Loose | 13 | 4 | 4 | 0.0 | 0.0 | 12.50 | 0.70 |
| | | 14 | 0 | 0 | 0.0 | 0.0 | 2.90 | 1.10 |
| | | 15 | 4 | 4 | 0.0 | 0.0 | 13.50 | 0.70 |
| | | 16 | 5 | 5 | 0.0 | 0.0 | 13.00 | 0.90 |
| | Moderate | 17 | 2 | 2 | 6.7 | 17.6 | 17.30 | 2.80 |
| | | 18 | 9 | 9 | 0.0 | 0.0 | 15.00 | 0.80 |
| | | 19 | 3 | 3 | 0.0 | 0.0 | 13.70 | 0.70 |
| | | 20 | 8 | 9 | 13.3 | 3.2 | 16.60 | 3.00 |
| | Tight | 21 | 166 | 166 | 0.0 | 0.0 | 15.20 | 3.00 |
| | | 22 | 163 | 163 | 0.0 | 0.0 | 15.80 | 3.00 |
| | | 23 | 151 | 151 | 0.0 | 0.0 | 18.00 | 4.00 |
| | | 24 | 144 | 144 | 0.0 | 0.0 | 13.90 | 1.30 |
| 9,9 | Loose | 25 | 0 | 0 | 0.0 | 0.0 | 10.50 | 2.20 |
| | | 26 | 65 | 65 | 0.0 | 0.0 | 37.30 | 8.80 |
| | | 27 | 0 | 0 | 0.0 | 0.0 | 11.20 | 1.30 |
| | | 28 | 69 | 69 | 9.0 | 34.8 | 35.90 | 7.50 |
| | Moderate | 29 | 210 | 252 | 35.7 | 9.5 | 58.70 | 7.00 |
| | | 30 | 336 | 374 | 13.4 | 3.5 | 56.60 | 7.30 |
| | | 31 | 304 | 368 | 29.0 | 5.4 | 49.50 | 9.90 |
| | | 32 | 269 | 322 | 27.6 | 10.5 | 50.20 | 12.20 |
| | Tight | 33 | 2555 | 2574 | 1.6 | 0.5 | 46.20 | 11.80 |
| | | 34 | 2479 | 2529 | 4.2 | 2.3 | 47.30 | 10.10 |
| | | 35 | 2730 | 2730 | 0.9 | 1.3 | 50.00 | 9.10 |
| | | 36 | 2410 | 2526 | 6.1 | 1.1 | 46.20 | 10.40 |

For 5 jobs / 5 machines and 7 jobs / 7 machines problems, the algorithm's performance is excellent. The deviation from the optimal solution is zero in 22 of 24 cases, meaning that it found the optimal solution in all 15 runs for all those 22 instances. The two instances for which the optimum was not found in all runs correspond to the moderate value of the due date tightness, instances 17 and 20 in Table 11, with average deviation from the optimum of 6.7% and 13.3% respectively. In this case the general average of the deviation from the optimal is higher (5%) for the problems with a moderate level of due date than that for problems with loose and tight due dates (0%) as can be seen in Table 12. Such a pattern is more obvious for the 9 jobs / 9 machines problems, where the averages of the Deviation from the optimal for the problems with loose and tight due dates are 2.2% and 3.2% respectively, while the correspondent value for the moderate due date type is 26.4%. The results show that problems with such a tightness level are harder to optimize for the proposed RKGA than problems belonging to the other two levels. For instances with tight due dates, it is known in advance that all jobs are late, so the problem becomes a Total Tardiness one. The total tardiness belongs to the set of regular measures in which case the solution space is restricted to the set of active schedules (Baker and Trietsch, 2009). Such knowledge is included in the algorithm, by enabling the Move Search, explained in Chapter 5, and disabling the Earliness Reduction Procedure as there is no need to move any operations to the right. The Move Search causes the RKGA to produce semi active schedules that are more compact to the left. However, we cannot do the same for non-regular measures like ET as was explained in section 6.4. Therefore, for problems with moderate and loose due dates the RKGA has to search in the more general set of all semi active schedules, which affects its performance. We conjecture that this explains why the results for problems with a tight due date are better (1.1% in average for all 12 instances with a tight due date) than those for problems with moderate due dates (10.5% on average for all 12 problems with a moderate due date).

**Table 12. Average Results for Jm/ /ET problem**

| Jobs, machines | Due date tightness | Deviation from Optimum (%) Average | Deviation from Optimum (%) Std. Dev. | Runtime (minutes) Average | Runtime (minutes) Std. Dev. |
|---|---|---|---|---|---|
| 5,5 | Loose | 0.0 | 0.0 | 0.1 | 0.1 |
| | Moderate | 0.0 | 0.0 | 4.4 | 0.1 |
| | Tight | 0.0 | 0.0 | 3.9 | 0.1 |
| 7,7 | Loose | 0.0 | 0.0 | 10.5 | 0.8 |
| | Moderate | 5.0 | 5.2 | 15.6 | 1.8 |
| | Tight | 0.0 | 0.0 | 15.7 | 2.8 |
| 9,9 | Loose | 2.2 | 8.7 | 23.7 | 5.0 |
| | Moderate | 26.4 | 7.2 | 53.8 | 9.1 |
| | Tight | 3.2 | 1.3 | 47.4 | 10.3 |
| Average | | 4.1 | 2.5 | 19.5 | 3.3 |

Regarding the moderate and loose due dates we observe the following. For certain sequences, some jobs that could not complete on time to meet a moderate due date may make it on time when such a due date is sufficiently extended by using a higher (loose) tightness multiple factor. Therefore, some sequences that were not optimal when using a tighter (moderate) due date may be optimal in the new problem resulting from extending the due dates by using a loose tightness multiple and keeping the rest of the data (processing times, and routing) unchanged. Therefore, as the due date becomes looser, the number of alternative optimal sequences may increase. When there are more alternative optimal solutions, it will be easier for the RKGA to find an optimal one, which explains why the algorithm performs better for loose due dates than for moderate due dates.

Conversely, as the due date level goes from loose to moderate for the same sequence jobs have a more limited time to complete. We conjecture that as there is less time between the jobs' release times and their due dates (in all our cases the release times are zero), the optimal schedules must be more compact to the left to meet the due dates, and fewer combinations may be optimal, which explains why the problems with loose due dates are better than the ones with moderate due dates.

Finally, we must acknowledge that the proposed generalization of the RKGA is able to find optimal or near optimal solutions for the Jm/ /ET problem for the problem sizes studied in this dissertation producing good results mainly for problems with tight and loose due dates. Although the results were not as good for problems with moderate due dates, the proposed generalized RKGA is a starting point for further research.

### 7.6.1 Runtime for Jm/ /ET problem

Unlike the Rm/ /Cmax problems, where the runtimes to find the optimal solution were less than one second, LINGO may take a considerable amount of time to solve the MIP for some instances of the Jm/ /ET problem.

Table 13 presents the time taken by LINGO to solve the 36 generated instances.

Table 14 resents a comparison between the times taken by LINGO to find the optimal solution for all problems with loose, moderate and tight due dates, and the average times observed for the proposed RKGA over all the 15 runs for the corresponding problems.

Based on Table 14, Figures 22, 23 and 24 show the behavior of runtimes for each type of due date. For problems with a loose due date, LINGO finds an optimal solution in less time than the average taken by the proposed RKGA, which in the worst case, problems with 9 jobs and 9 machines, is 23.7 minutes.

In the case of moderate and tight due dates, the behavior is similar for problems with up to 7 jobs and 7 machines. However, for problems with 9 jobs and 9 machines, the average runtime for the proposed RKGA grows more softly than LINGO runtime. Recall that Jm/ /ET is an NP-hard problem and the proposed RKGA becomes important if we think, for example, of a situation of a job shop manager waiting 23 hours and 13 minutes in the case of problem 33, for LINGO to find the optimal schedule for the next 8 hour shift. This may be totally impractical compared to waiting 46.2 minutes (on average) for the proposed RKGA to produce a schedule that has an average deviation from the optimal solution of 1.6%.

**Table 13. LINGO Rutimes (in minutes) to solve Jm/ /ET benchmark problems**

| Due date type | Instance | Runtime | Instance | Runtime | Instance | Runtime |
|---|---|---|---|---|---|---|
| Loose | 1 | 0 | 13 | 0 | 25 | 0 |
| | 2 | 0 | 14 | 0 | 26 | 0 |
| | 3 | 0 | 15 | 0 | 27 | 0 |
| | 4 | 0 | 16 | 0 | 28 | 0 |
| Moderate | 5 | 0 | 17 | 0 | 29 | 505 |
| | 6 | 0 | 18 | 240 | 30 | 278 |
| | 7 | 0 | 19 | 60 | 31 | 535 |
| | 8 | 0 | 20 | 300 | 32 | 317 |
| Tight | 9 | 0 | 21 | 269 | 33 | 1393 |
| | 10 | 0 | 22 | 146 | 34 | 837 |
| | 11 | 0 | 23 | 113 | 35 | 1006 |
| | 12 | 0 | 24 | 20 | 36 | 529 |

The equivalent situation for the case of moderate due dates with 9 jobs and 9 machines problems, is not as clear as the average deviation from the optimal solution is 26.4%. Thus, there is a tradeoff between the runtime and the deviation from the optimal. Nevertheless, as was previously mentioned, the proposed RKGA is a starting point for more research and some improvements may be accomplished in the future.

Table 14. Runtime (in minutes) comparison of LINGO and the proposed RKGA

| Problem Size (Jobs/Machines) | Due date type | | | | | |
|---|---|---|---|---|---|---|
| | Loose | | Moderate | | Tight | |
| | LINGO | RKGA | LINGO | RKGA | LINGO | RKGA |
| 5/5 | 0 | 0.1 | 0 | 4.4 | 0 | 3.9 |
| 7/7 | 0 | 10.5 | 2.5 | 15.6 | 2.3 | 15.7 |
| 9/9 | 0 | 23.7 | 408.8 | 53.8 | 941.3 | 47.4 |

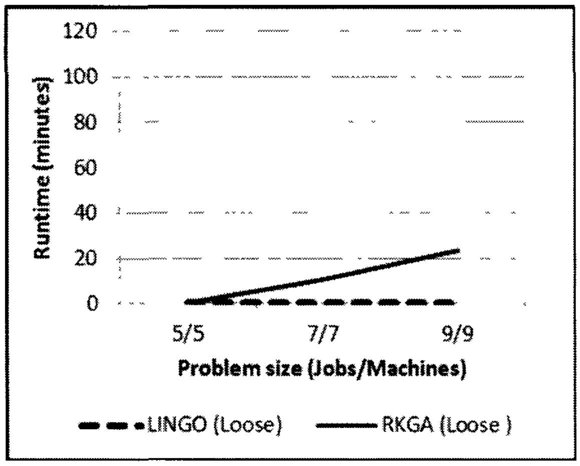


Figure 22. Runtime comparison for problems with loose due dates

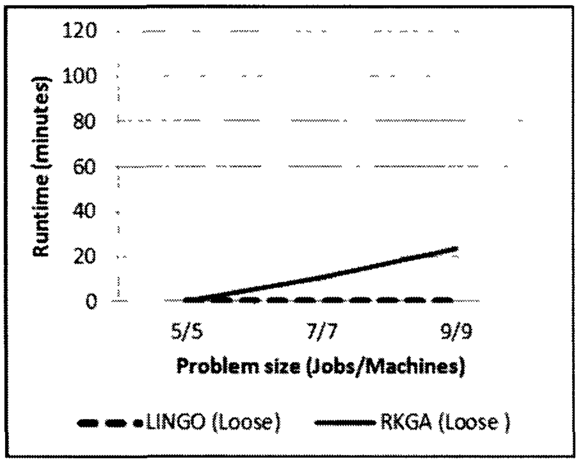


Figure 23. Runtime comparison for problems with moderate due dates

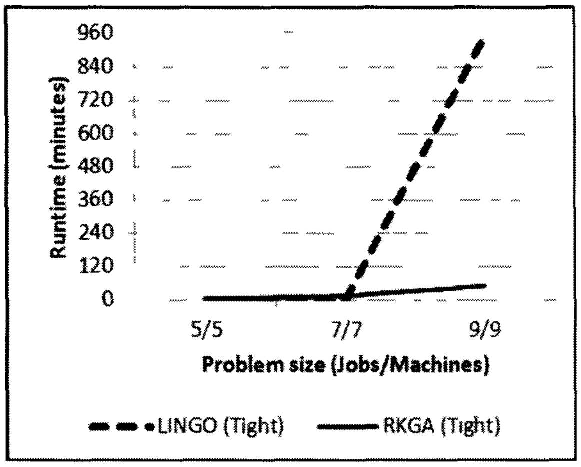


**Figure 24. Runtime comparison for problems with tight due dates**

# 8 A REACTIVE RANDOM KEYS GENETIC ALGORITHM APPROACH FOR SHOP SCHEDULING PROBLEMS

In this chapter, a RKGA able to react to unexpected events in the production environments targeted in this dissertation, which we will call the reactive RKGA, is presented. A similar algorithm to the one previously developed to produce the predictive schedule is presented here to generate a reactive schedule once an unexpected event occurs.

## 8.1 Unexpected events

Hall and Potts, (2010), Vieira et al. (2003), Subramaniam and Raheja (2003), and Abumaizar and Svestka, (1997), review the different unexpected events reported in the rescheduling literature. Among them we find: urgent job arrival, rework (or quality problems), job cancellation, delay in the arrival of materials, change in job priority, due date change, machine breakdown, tool breakdown, operator absenteeism, process time variation and changes in release times. They reference rescheduling approaches designed for a specific production setting and a specific type of disruption event, or several events as in the case of Vieira et al. (2003) and Subramaniam and Raheja (2003) for the $Jm/\ /C_{max}$ problem.

The model proposed here considers some of the previous disruptions and a specific case not addressed in the reviewed literature, which is the study of simultaneous breakdowns. The disruptions considered by our reactive RKGA are:

1. Urgent job arrival.
2. Job Rejection  implying immediate reprocessing of operations.
3. Delay in the arrival of materials.

4. Machine unavailability. This covers the specific situations of machine breakdowns, tool breakdowns, and operator absenteeism, all of which imply that one or several machines will become unavailable.

**8.2 Rescheduling performance measures**

It is our objective in this phase to generate a reactive schedule that deviates from the predictive schedule as little as possible. This can be measured in two ways in the literature: how much the reactive schedule changes compared to the predictive one (a measure of stability) and how much its performance changes (a measure of efficiency). The resulting reactive schedule should therefore be as efficient and stable as possible. We use the concept of efficiency in the sense of measuring the change in the schedule's performance. Subramaniam and Raheja (2003) and Subramaniam et al. (2005) measure the efficiency, $e$, of the reactive schedule as a percentage of change in the value of objective function under consideration, the makespan:

$$e = \left\{1 - \frac{[Mnew - Mo]}{Mo}\right\} \times 100\% \qquad (7)$$

where $Mo$ is the makespan for the predictive schedule, and $Mnew$ is the makespan of the reactive schedule. In the same line we can define in general the inefficiency as ratio as follows:

$$ineficciency = \frac{[\phi new - \phi o]}{\phi o} \qquad (8)$$

where $\phi$o is the value of the objective function for the predictive schedule, and $\phi$new is the value of the objective function of the reactive schedule.

In the context of reactive scheduling, stability is referred to providing a reactive schedule that deviates from the predictive one as little as possible (Herroelen and Leus, 2005). The stability is measured in two dimensions. First, there is the starting time deviation, used by authors like Abumaizar and Svestka

(1997), Subramaniam and Raheja (2003) and Subramaniam et al. (2005). It is defined as:

$$\xi = \frac{\sum_{j=1}^{k}\sum_{i=1}^{no_j}\left|\left(S_{ji}^{*} - S_{ji}\right)\right|}{\sum_{j=1}^{n} no_j} \qquad (9)$$

where:

$\xi$ is the normalized deviation,

$no_j$ the number of operations of job *j*,

*k* the number of jobs,

$S_{ji*}$ the starting time of *i*th operation of job *j* in the repaired schedule,

$S_{ji}$ is the starting time of *i*th operation of job *j* in the original schedule.

Under the strategies of right shift and modified affected operations rescheduling, in which the predictive sequence is kept unchanged and only the starting time may change, calculating the starting time deviation of operations in both the predictive and reactive schedules gives a measure of the reactive schedule's stability. From production perspective, however, measuring the instability by the starting time deviations may be useful when secondary resources (like tools in the manufacturing situation) are expected to be used by a machine during an operation in the predictive schedule and then delivered to another machine. However, there is a second way to measure instability based on the sequence deviation. Having a reactive schedule that does not deviate much from the initial sequence may be very useful especially in that a series of setups and queues of material organized according to the predictive sequence may have to be changed when a disruption occurs. Changing the sequence of the material in the queues or the setup order may turn out to be costly. Moreover except from the right shift approach, a reactive schedule does not necessarily produce the same predictive sequence. That is why we favor the use of a sequence deviation based stability measure.

Abumaizar and Svestka (1997) define a sequence deviation stability measure based on the summation for each operation j of the amount of operations processed before j in the predictive schedule which are processed after j in the reactive schedule. Based on that, Moratori et al. (2008) propose the following way to measure the sequence deviation based on the following concept. Let $M$ be the number of machines and $O_i$ the number of operations that have to be processed on machine $i = 1, \ldots M$. Let $R_{ij}$=1 if the immediate successor of operation $j = 1, \ldots O_i$ on machine $i$ in the initial schedule remains a successor in the new schedule but not necessarily an immediate one and 0 otherwise. To each machine $i$ a measure of sequence stability $R_i \in$ [0, 1] is assigned in (10):

$$R_i = \sum_{j=1}^{O_{i-1}} \frac{R_{ij}}{O_{i-1}} \qquad (10).$$

Similarly, and using the same definitions of $M$ and $O_i$, let us define a sequence deviation ratio $Sequence\_Dev$ as:

$$Sequence\_Dev = \frac{\sum_{i=1}^{M} \sum_{j=1}^{O_{i-1}} Prec_{ij}}{\sum_{i=1}^{M} O_{i-1}} \qquad (11)$$

where $Prec_{ij} = 1$ if operations $i$ and $j$ remain on the same machine in the reactive schedule and the immediate successor of operation $j$ on machine $i$ in the predictive schedule remains a successor in the reactive schedule but not necessarily an immediate one; otherwise, $Prec_{ij} = 0$.

The previous index measures how much the precedence was respected in the reactive schedule. In cases of parallel machines, we want to measure as well the machine allocation changes. Let $OCC$ be the set of all operations that can be processed on more than one machine and $TOCC$ the cardinality of $OCC$. Let us define the machine deviation ratio as:

$$Machine\_Dev = \frac{\sum_{j=1}^{TOOC} B_j}{TOCC} \qquad (12)$$

where

$$B_j = \begin{cases} 1, \mathit{if\ operation}\ j \in \mathit{OCC\ changed\ machine} \\ \quad \mathit{allocation\ in\ the\ reactive\ schedule} \\ 0, \mathit{otherwise} \end{cases}$$

Our reactive scheduling problem consists, therefore, of obtaining a reactive schedule that is stable and whose performance degrades as little as possible with respect to the predictive schedule. In other words, we want to produce a reactive schedule that minimizes the *inefficiency* and the *instability* expressed in terms of the sequence deviation and machine assignment deviation. In order to do this, we will consider in our objective function the three ratios defined earlier.

## 8.3 Multi objective optimization

Our rescheduling problem is then a multi-objective optimization problem. That is, a problem which has two or more objectives that need to be simultaneously optimized. In the context of multi objective optimization, a compromised solution is one that is as close as possible to the utopia point. That is, a point that simultaneously succeeds in optimizing each objective.

Without loss of generality, the multi optimization objective problem MOOP may be described as follows: if $x$ is a p-dimensional vector of decision variables $x = (x_1, \dots, x_2, \dots, x_p)$ in the decision space $X$, and $f(x)$ evaluates the quality of a specific solution $x$ by assigning to it an objective vector $(f_1(x), f_2(x), \dots, f_k(x))$, and we require the simultaneous optimization of $k$ objectives, the general MOOP can be stated as:

Min $\quad f(x) = (f_1(x), f_2(x), \dots, f_k(x))$

Subject to $\quad g_i(x) \leq b_i, i = 1,2 \dots, c \quad$ (C9)

$\quad x \geq 0$

where (C9) are certain inequality constraints.

### 8.3.1 Pareto dominance

Assuming a minimization problem as in our case, a vector $u = (u_1, u_2, \dots, u_p)$ is considered to dominate another vector $v = (v_1, v_2, \dots, v_p)$ if no component of $u$ is greater than the corresponding component of $v$ and at least one component is smaller. A solution $x_u \in X$ is considered to be Pareto-optimal or non-dominated if and only if there is no $x_v \in X$ for which $v = f(x_v) = (v_1, v_2, \dots, v_p)$ dominates $u = f(x_u) = (u_1, u_2, \dots, u_p)$.

Fonseca and Fleming (1998) classify multi objective optimization methods into the following three categories depending on how the decision processes and the computation are articulated in the search for a compromise solution.

**Apriori methods**

Before running the optimization algorithm, the decision maker indicates the relative importance of the desired goals in terms of an aggregating scalar function that combines all of the objective function terms, making the problem, according to Fonseca and Fleming (1998), a single-objective one prior to optimization.

**A posteriori methods**

Before expressing any preferences, the optimizer presents a set of candidate non-dominated solutions to the decision maker who chooses from that set.

**Progressive articulation of preferences**

At each step of the optimization process, and by an interactive process between the decision maker and the optimizer, the optimizer provides a non-dominated solution for which the decision maker expresses his/her preferences, which defines a new search direction for a better alternative. The process goes on until a satisfactory solution is reached.

By using an a priori approach, the weighted sum method will be utilized by the reactive RKGA to minimize an aggregating scalar objective function of efficiency and instability. The method is described in the following section.

## 8.4 The weighted sum method

Using the weighted sum method to solve a multi objective optimization problem requires selecting scalar weights $w_i$ and the minimization of an aggregating objective function as follows:

$$U = \sum_{i=1}^{k} w_i f_i(x) \qquad (13).$$

The weighted sum method has been used in the literature in two ways. First, a posteriori, to provide numerous solution points by systematically altering the weights to explore the Pareto optimal set. Second, apriori which provides a single solution point that reflects preferences of the decision maker in a single set of weights. For this work we consider the second approach, in which the decision maker expresses from the beginning her/his preferences in a single set of weights apriori.

Unlike the posteriori approach that uses a set of weights that add up to 1, there is no need for such restriction, which makes it easier to determine the appropriate weight values (Marler and Arora, 2010).

Finally, our objective function will be expressed as follows:

$$\text{Min } Z = w_1 \times inefficiency + w_2 \times Sequence\_dev + w_3 \times Machine\_dev \qquad (14)$$

where $w_1$, $w_2$, and $w_3$ represent the decision maker preferences. Note that in general the objective functions may have different units in (13). In our cases the inefficiency measure as well as the two types of instability proposed are formulated as ratios, so the three of them are dimensionless and can be consistently aggregated.

## 8.5 Mode of operation of the Reactive RKGA

The mode of operation of the reactive RKGA is as follows. There is a predictive schedule being executed. At a certain time $t$ a disruption occurs. Such disruption may be of two types. In the first type the processing of one or more operations suddenly stops and the operations will need to be repeated once it is possible. We will refer to these operations as "affected operations". Situations like machine breakdowns, tool breakdowns or a power failure that affects one or several machines belong to this category. Under the second type, the operations that are being processed in the time when the unexpected event happens, may complete before the schedule execution is stopped. To this category belong all other unexpected events considered for the present reactive model.

The pool of jobs that need rescheduling includes the information of the new jobs that must be included in the reactive schedule, plus the information of the jobs not yet finished for which all the affected and not yet started operations must be included.

The new machine availability times and the information of the new job pool, as well as the predictive schedule, are used as inputs by the RKGA to produce the reactive schedule minimizing the inefficiency and instability.

## 8.6 The rollback mechanism

Since the predictive schedule is the result of the evolutionary process of the RKGA, the genetic information of that schedule may prove useful in the search of a stable and efficient reactive solution. Therefore, a certain percentage of chromosomes with the genetic information of the predictive schedule may be inserted in the initial population. More specifically, the chromosome of the predictive schedule, which is the best schedule produced by the predictive RKGA as a result of the evolutionary process, is cloned a certain number of times as determined by a parameter called *Rollback Percentage* and inserted as a part of the initial population of the reactive RKGA. Some experiments with different values of the *Rollback Percentage* it are presented in the next chapter.

In the case of new jobs that enter the reactive job pool, new values of random keys and delay factors are created for them to complete the genetic information of the chromosomes to be inserted by the rollback.

Except for the multiple objective function and the inserted chromosomes, the rest of the RKGA, namely the problem representation, genetic operators and stopping criteria, work in the same way as the predictive schedule.

## 8.7 Encoding of a non RKGA generated schedule

In case there is a predictive schedule produced by a mean other than the RKGA (e.g. manual schedule), the reactive algorithm should be able to use it as an input to produce a reactive solution. Such predictive schedule and its objective function value are used as an input for the reactive RKGA.

The predictive sequence may be used as well to produce surrogate chromosomes to input by rollback as explained in the previous section. A triplet of machine key, random key and delay factor is created then for each operation of the predictive schedule. The machine key value is an integer that must coincide with the position that occupies the machine assigned to each operation in the predictive schedule on its machine list, as defined in expression (5) of Chapter 6. A total of *ops* (0, 1) uniformly distributed random numbers is created, where *ops* is the total of operations in the predictive schedule. The random numbers are ordered increasingly. The operations are ordered chronologically as they were scheduled. Each random number is assigned then to each operation, so the same sequence results on each machine once the decoding procedure is applied to the chromosome. The delay factor is randomly created and not used in the process of creating the surrogate chromosome as the predictive schedule may be any feasible sequence of jobs. The encoding process for a non RKGA generated schedule is presented below.

**Inputs:**

*ops:* total of operations to schedule

*Schedule*(): Array of two dimensions(*ops,* 3), with the schedule information

*Schedule*(*i*, 1): Stores an operation number

*Schedule*(*i*, 2): Stores the start time of *Schedule*(*i*, 1)

*Schedule*(*i*, 3): Stores the machine key of *Schedule*(*i*, 1)

**Variables:**

*Co*: integer to store current operation

*Random_Key*( ):One dimensional array of size *ops,* where *Random_Key*(*i*) stores the Random Key of operation *i*

*Delay_Factor*(): One dimensional array of size *ops,* where *Delay_Factor*(*i*) stores the Delay Factor of operation *i*

*Machine*(): One dimensional array of size *ops,* where *Machine*(*i*) stores the machine key of operation *i*

*Random_Val*(): Two dimensional array of size *ops* x 2, where:

*Random_Val*(i, 1) stores operation i's surrogated random key and

*Random_Val*(i, 2) stores operation i's surrogated delay factor

**Complementary Functions:**

Random(0,1) Returns a random number between 0 and 1.

**Pseudo Code of the Schedule Encoding Procedure**

```
For i= 1 to ops
  For j=1 to 2
      Random_Values(i, j)=Random(0,1)
  Next j
Next i
Order Schedule() increasingly by column 2 (Start time)
Order Random_Values() increasingly by column 1 (Random keys)
For i= 1 to ops
  Co =Schedule(i, 1)
  Random_Key(Co) = Random_Values(i,1)
  Delay_Factor(Co) = Random_Values(i,2)
  Machine(Co) = Schedule(i, 3)
Next i
```

## 8.8 Integrated Predictive Reactive Scheduling System

In Chapter 6 we presented the predictive model, capable of producing initial schedules for problems coming from different environments with regular measures or the non-regular measure of Earliness and Tardiness. In this chapter we presented the reactive model, which is able to produce a reactive schedule when different unexpected events occur during the execution the predictive schedule. A flow chart representing the integration of both models in a whole system is presented in Figure 25.

At first, an initial schedule is produced by the predictive model and adopted as the current schedule. Then, the schedule execution starts. If an unexpected event occurs, such as those explained in section 8.1, which makes the initial schedule infeasible or obsolete, the reactive model is used to generate a new efficient and stable schedule in response to the event. The new schedule includes the disrupted and not yet started operations and becomes the current schedule.

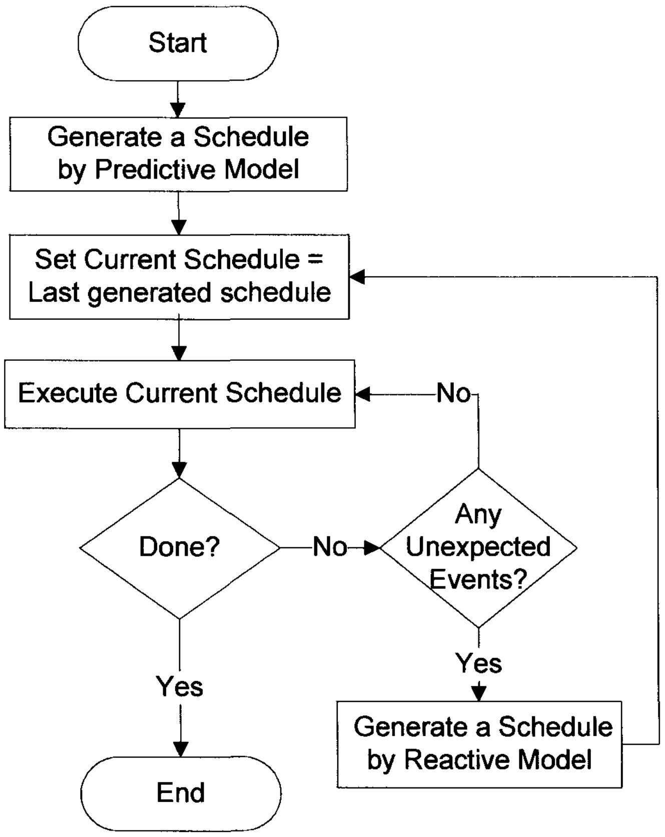

Start
Generate a Schedule by Predictive Model
Set Current Schedule = Last generated schedule
Execute Current Schedule
Done?
No
Any Unexpected Events?
No
Yes
Yes
Generate a Schedule by Reactive Model
End

**Figure 25. Predictive Reactive Scheduling System**

## 8.9 Conclusion

We have presented a reactive RKGA that is able to produce a reactive schedule that minimizes inefficiency, and instability when different unexpected events occur in the various production environments targeted in this dissertation.

At the end of this chapter the integrated view of the Predictive Reactive Scheduling System was presented. The next chapter presents some computational experiments to test the reactive model.

# 9 COMPUTATIONAL EXPERIMENTS FOR THE REACTIVE MODEL

According to characteristics of the reactive model, it can be used in four different basic production environments, for regular and non-regular measures and under different types of disruptions. We must make a choice among all the embraced environments, objective functions and disruptions to test how the reactive RKGA works. Based on the representation proposed in section 6.3, it is possible to cover hybrid environments, such as flexible flow shops and flexible job shops. Since the flexible job shop is the most general case that the proposed representation can account for, it has been selected to conduct our experiments as the environment of choice. Regarding the objective function, most of the literature reviewed for reactive approaches focuses on regular objective functions, and the case of ET has not been researched enough although it has recently become an important objective in the literature in general. Therefore, the ET will be our objective function choice on which the experiments will be conducted. Concerning the disruptions, we select two types that are representative of the unexpected events covered by this research, namely machine breakdowns and the arrival of a rush job that is, a job that arrives after the execution of the predictive schedule has started and must be included in it when it arrives. The first disruption implies that some operations will be interrupted. The second one, on the other hand, is of the type in which the operations being processed may complete before the predictive schedule execution is stopped. Each of those types of unexpected events has different experimental factors and levels to take into account such as: disruption time, duration of the disruption and number of affected machines in the case of machine breakdowns. Additionally, we want to study the effect of the rollback mechanism at different levels in the reactive RKGA.

## 9.1 Benchmark Problems

In order to experiment with different levels of those factors, we consider one size of flexible job shop with seven jobs that must be scheduled in a shop with seven work centers with three unrelated parallel machines on each. Recall that a flexible job shop is a job shop with possibly multiple parallel machines at each work center, and not necessarily just one machine per work center. A set of 120 benchmark flexible job shop problems with those characteristics was generated. In order to have all jobs visiting all work centers in some random order, the job routes were generated following a discrete uniform distribution U[1,$wc$] where $wc$ is the number of work centers. The processing times over the different machines of each work center were randomly generated from a discrete uniform distribution U[1,9] as in Demirkol et al. (1998) and Brandimarte (1993). The due date of each job is set as a multiple $TF$ of the summation of the minimum job processing time on each work center. Such multiple determines the tightness of the due date and was generated following a uniform distribution U[0.7,1.4]. In the case of the arrival of a rush job, it will have the same characteristics explained above, namely routing through the work centers, processing times and due date.

## 9.2 Predictive Schedule Generation

Each problem was run by the predictive model to minimize ET using the same genetic operators, parameter values, and stopping criteria as those of the experiments of the predictive model (a reproduction of 20%, a uniform crossover 84% and a mutation by immigration of 6%). Likewise, the reactive RKGA stops when a maximum number of 250 iterations has been reached or 75 generations have passed without any improvement of the best solution found so far. The population size is set to 300 plus two times the number of operations to be scheduled.

## 9.3 Machine breakdown experiments

The first experiment consists of one or multiple machine breakdowns that occur during the execution of the predictive schedule. The dimensions of the study are presented next.

### 9.3.1 Dimensions of the study

The following factors and levels are used to generate the experiments.

- Disruption time: Refers to the moment in which the breakdown occurs expressed as a percentage of the makespan of the predictive schedule $\mathrm{Cmax}p$ and is set at two levels: Early, that corresponds to a value generated from a uniform distribution U[0.05 $\mathrm{Cmax}p$, 0.4 $\mathrm{Cmax}p$] and Late, corresponding to a value generated from a uniform distribution U[0.6 $\mathrm{Cmax}p$,0.9 $\mathrm{Cmax}p$]
- Duration: Refers to the duration of the breakdown and is expressed as a percentage of the makespan of the predictive schedule, $\mathrm{Cmax}p$ and set at two levels: Short and Long, corresponding to values generated from uniform distributions U[0.05 $\mathrm{Cmax}p$,0.2 $\mathrm{Cmax}p$] and U[0.4 $\mathrm{Cmax}p$,0.6 $\mathrm{Cmax}p$], respectively.
- Percentage of Affected Machines: Refers to the number of machines affected by the breakdown. It is expressed as a percentage of the total number of machines $m$ that may be busy at any time $t$ and is set at two levels: Low, taking a value from a uniform distribution U[0.01$m$, 0.33$m$] and High, taking a value from a uniform distribution U[0.67$m$, $m$]
- Rollback Percentage: Refers to the percentage of chromosomes with the genetic information of the predictive schedule to be inserted in the initial population. Three levels were considered for the experiments: 0%, 10% and 20%.

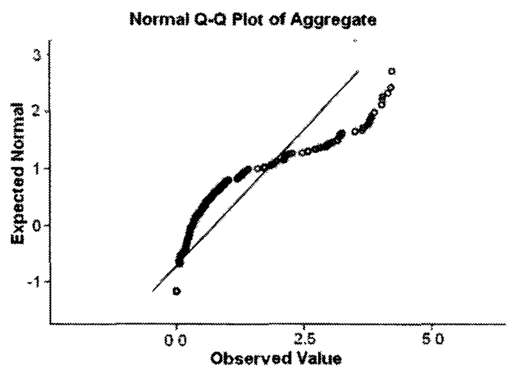
The response variables considered for the study are the Aggregate objective function as given in expression (14) of Chapter 8, and the Runtime.

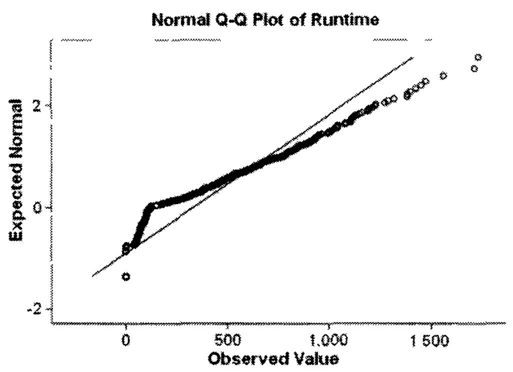
Since we are studying flexible job shop problems with 9 jobs, 9 stages and 3 unrelated parallel machines, five instances were used for each treatment combination and each of them was run five times. Table 15 presents the experimental combinations for this experiment and the average values for the aggregate objective function, the instability, inefficiency and runtime.

**Table 15. Treatment Combinations and average values**

| Rollback Level (%) | Disruption Moment | % Affected Machines | Duration | Aggregate objective | Instability | Inefficiency | Runtime (sec) |
|---|---|---|---|---|---|---|---|
| 0 | Early | High | Long | 2.60 | 0.59 | 2.00 | 723.16 |
| 0 | Early | High | Short | 0.62 | 0.27 | 0.35 | 656.68 |
| 0 | Early | Low | Long | 0.95 | 0.45 | 0.50 | 797.72 |
| 0 | Early | Low | Short | 0.25 | 0.14 | 0.11 | 849.08 |
| 0 | Late | High | Long | 1.30 | 0.36 | 0.94 | 73.04 |
| 0 | Late | High | Short | 0.09 | 0.00 | 0.09 | 41.12 |
| 0 | Late | Low | Long | 0.15 | 0.00 | 0.15 | 49.44 |
| 0 | Late | Low | Short | 0.08 | 0.00 | 0.08 | 78.60 |
| 10 | Early | High | Long | 3.34 | 0.56 | 2.77 | 751.00 |
| 10 | Early | High | Short | 0.80 | 0.16 | 0.64 | 546.76 |
| 10 | Early | Low | Long | 0.33 | 0.13 | 0.20 | 371.60 |
| 10 | Early | Low | Short | 0.15 | 0.05 | 0.10 | 332.96 |
| 10 | Late | High | Long | 0.89 | 0.43 | 0.46 | 84.40 |
| 10 | Late | High | Short | 0.32 | 0.00 | 0.32 | 59.48 |
| 10 | Late | Low | Long | 0.16 | 0.01 | 0.15 | 23.00 |
| 10 | Late | Low | Short | 0.05 | 0.00 | 0.05 | 30.24 |
| 20 | Early | High | Long | 2.77 | 0.88 | 1.89 | 671.44 |
| 20 | Early | High | Short | 0.61 | 0.18 | 0.43 | 770.72 |
| 20 | Early | Low | Long | 0.89 | 0.30 | 0.59 | 392.84 |
| 20 | Early | Low | Short | 0.16 | 0.06 | 0.09 | 359.48 |
| 20 | Late | High | Long | 0.92 | 0.01 | 0.91 | 94.80 |
| 20 | Late | High | Short | 0.17 | 0.00 | 0.17 | 66.08 |
| 20 | Late | Low | Long | 0.33 | 0.10 | 0.22 | 27.48 |
| 20 | Late | Low | Short | 0.05 | 0.00 | 0.05 | 46.52 |

In order to compare the behavior of the RKGA at different factor levels, we must check the assumption of normality for parametric analysis methods. This was done by examining the residuals of the response variables (aggregate objective function (Aggregate) and runtimes (Runtime)) produced by the ANOVA model through a normal probability plot and Kolmogorov-Smirnov normality tests in SPSS statistical software. The results, presented in Figure 26, show that none of them follow a normal distribution ($p < 0.05$). Thus, the statistical significance of performance among the experimental factors can be analyzed by the nonparametric Kruskal–Wallis test.





| Tests of Normality | | | |
|---|---|---|---|
| | Kolmogorov-Smirnov(a) | | |
| | Statistic | df | Sig. |
| Aggregate | .236 | 600 | .000 |
| a Lilliefors Significance Correction | | | |

| Tests of Normality | | | |
|---|---|---|---|
| | Kolmogorov-Smirnov(a) | | |
| | Statistic | df | Sig. |
| Runtime | .224 | 600 | .000 |
| a Lilliefors Significance Correction | | | |

**Figure 26. Normal Probability Plot for Aggregate Objective Function and Runtime. Breakdown Experiment**

Nonparametric statistical methods do not make assumptions about the data distribution. That makes them particularly useful under situations of non-normality. The Kruskal-Wallis test is a nonparametric alternative to the analysis of variance, used "to test the null hypothesis that k treatments are identical against the alternative hypothesis that some of the treatments generate observations that are larger than others" (Montgomery, 2009). The test uses the rank of the observations rather than the actual observations for the analysis. The Kruskal–Wallis tests are performed using SPSS 14.0 statistical software. Besides the two mentioned response variables of Aggregate objective function and Runtime, the Instability (summation of expressions (11) and (12)) and the Inefficiency are analyzed by the Kruskal-Wallis test as well.

### 9.3.2 Results for Disruption Time

Regarding the disruption moment, the results in Table 16 show that there is a statistical difference ($P < \alpha=0.05$) for the Aggregate objective function and for the Runtime between the Early and Late levels.

**Table 16. Results for factor: Disruption Time**

| Response | Factor Level | Mean Rank | P |
|---|---|---|---|
| Runtime | Early | 433.7 | 0.000 |
| | Late | 167.3 | |
| Instability | Early | 410.16 | 0.000 |
| | Late | 190.84 | |
| Inefficiency | Early | 355.97 | 0.000 |
| | Late | 245.03 | |
| Aggregate | Early | 378.87 | 0.000 |
| | Late | 222.14 | |

The mean rank suggests that late disruptions result in better values of the Aggregate objective function and Runtime. That is explained by the fact that when a breakdown occurs late in the schedule execution, most of operations of the schedule have been completed and therefore the reactive schedule will have mainly the same sequence and machine allocation. In fact, in a late breakdown, it is more likely that some jobs have completed and efficiency is not as hardly affected compared to when a breakdown occurs early on in the schedule. In a similar vein, since most schedule operations will have been completed, the required runtime to schedule the remaining operations will be less.

### 9.3.3 Results for Duration of the breakdown

According to the results in

Table 17, it cannot be concluded that there exists a significant difference in the Runtime under short and long breakdowns (P > α=0.05). In terms of the problem, under a long duration some machines simply become available later than in a short breakdown, which does not imply any additional effort for the decoding procedure to produce a schedule compared to that of building a schedule where some machines are available earlier.

The results show as well that there is a statistical difference for the Aggregate objective function under short and long breakdowns (P < α=0.05).

**Table 17. Results for factor: Duration of the breakdown**

| Response | Factor Level | Mean Rank | P |
|---|---|---|---|
| Runtime | Short | 292 | 0.221 |
| | Long | 309 | |
| Instability | Short | 249 | 0.000 |
| | Long | 352 | |
| Inefficiency | Short | 233 | 0.000 |
| | Long | 368 | |
| Aggregate | Short | 232 | 0.000 |
| | Long | 369 | |

The mean rank suggests that short breakdowns produce lower values of the Aggregate objective function. Longer breakdowns imply either displacing more operations to the right which will affect the efficiency or doing more changes on machine allocation, which will deteriorate the stability. Conversely, shorter breakdowns may be solved with less operation right shifts and/or less machine allocation changes so the RKGA is able to produce a reactive schedule more similar to the predictive one which represents more stability and efficiency.

### 9.3.4 Results for Percentage of Affected Machines

Regarding the percentage of affected machines, the test results in Table 18 show that there is a statistical difference in the Runtime, Instability, Inefficiency, and Aggregate objective function at the two levels of affected machines. At low percentages of affected machines, the runtime required is less as fewer operations are hit and, therefore, fewer operations need to be rescheduled. Likewise, less affected operations will cause less instability and inefficiency and therefore a better value of the Aggregate objective function.

**Table 18. Results for factor: Percentage of Affected Machines**

| Response | Factor Level | Mean Rank | P |
|---|---|---|---|
| Runtime | Low | 264 | 0.000 |
| | High | 337 | |
| Instability | Low | 260 | 0.000 |
| | High | 341 | |
| Inefficiency | Low | 211 | 0.000 |
| | High | 390 | |
| Aggregate | Low | 219 | 0.000 |
| | High | 382 | |

### 9.3.5 Results for Rollback Level

The results of the Rollback Level, which was defined as the percentage of chromosomes cloned from the one of predictive schedule and inserted in the reactive RKGA's initial population, are given in Table 19. It cannot be concluded that there exists significant difference between the three levels for the Instability, Inefficiency and the Aggregate objective function.

An interesting observation, however, is present in the results for the rollback level regarding the runtime. The results show that there is a statistical difference in the runtime at the different levels of rollback. However, the test does not reveal which means differ significantly. Thus, pair tests need to be performed and their results in Table 20 show that the levels of 10% and 20% imply less runtime for the RKGA than the 0% level. This means that inserting a number of individuals in the initial population of the reactive problem with the genetic information of the predictive schedule speeds up the search for a reactive solution. However, it cannot be concluded that there exists significant difference between the rollback levels of 10% and 20%.

The average of the runtimes taken over the three rollback levels in Table 15 shows that the average time taken by the RKGA to generate a reactive solution at a 20% rollback level (303.67 seconds) is 74.3% of the time taken at a 0% of rollback level (408.61 seconds); the average time taken by the RKGA to generate a reactive solution at a 10% rollback level (274.93 seconds) is 67.3% of the time taken at a 0% of rollback level. Using those average times, the level of 10% of rollback seems to be adequate.

**Table 19. Results for factor: Rollback Level**

| Response | Factor Level | Mean Rank | P |
|---|---|---|---|
| | 0% | 305.58 | |
| Aggregate | 10% | 290.2 | 0.584 |
| | 20% | 305.73 | |
| | 0% | 316.55 | |
| Instability | 10% | 286.29 | 0.181 |
| | 20% | 298.66 | |
| | 0% | 290.21 | |
| Inefficiency | 10% | 298.2 | 0.400 |
| | 20% | 313.1 | |
| | 0% | 330.56 | |
| Runtime | 10% | 274.36 | 0.005 |
| | 20% | 296.59 | |

**Table 20. Additional tests for factor: Rollback Level and Runtime as response variable**

| Response | Levels | Mean Rank | P | Levels | Mean Rank | P | Levels | Mean Rank | P |
|---|---|---|---|---|---|---|---|---|---|
| Runtime | 0% | 218.9 | 0 | 10% | 192.8 | 0.180 | 0% | 212.1 | 0.04 |
| | 10% | 182.1 | | 20% | 208.2 | | 20% | 188.9 | |

## 9.4 Rush Job experiment

In the second experiment, an urgent job arrives during the execution of the predictive schedule, and the schedule execution must be stopped in order to include the new job. The dimensions of the study are presented next.

### 9.4.1 Dimensions of the study

The following factors are used to generate the experiments.

- Arrival time: Refers to the time point in which the new job arrives and needs to be scheduled. It is expressed as a percentage of the makespan of the predictive schedule $Cmax_p$ and is set at two levels: Early, that corresponds to a value generated from a uniform distribution U[0.05 $Cmax_p$, 0.4 $Cmax_p$] and Late, corresponding to a value generated from a uniform distribution U[0.6 $Cmax_p$,0.9 $Cmax_p$]
- Rollback Percentage: The same three levels were considered for the experiments: 0%, 10% and 20%.

The response variables considered for the study are the Aggregate objective function and the Runtime.

Five instances were used for each treatment combination, and each of them was run five times. Table 21 presents the experimental combinations for this experiment and the average values obtained for the experiments.

**Table 21. Treatment combinations and average values for the Rush Job experiment**

| Rollback Level (%) | Arrival time | Instability | Inefficiency | Aggregate | Runtime (sec) |
|---|---|---|---|---|---|
| 0 | early | 0.08 | 0.04 | 0.14 | 1069.24 |
| 0 | late | 0.00 | 0.06 | 0.06 | 68.64 |
| 10 | early | 0.00 | 0.00 | 0.00 | 222.96 |
| 10 | late | 0.00 | 0.20 | 0.20 | 143.28 |
| 20 | early | 0.00 | 0.01 | 0.01 | 399.76 |
| 20 | late | 0.00 | 0.17 | 0.17 | 49.08 |

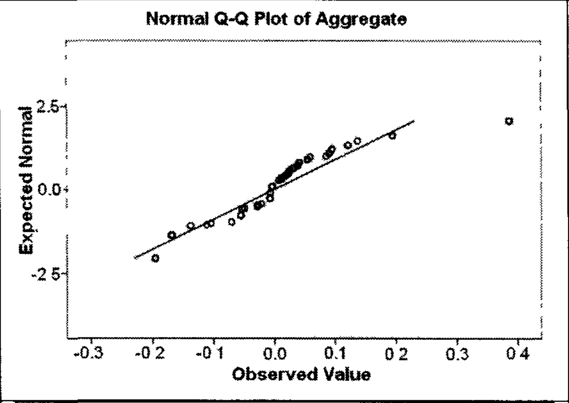
As was done in the breakdown case, in order to compare the behavior of the RKGA at the different factor levels and to check the assumption for parametric analysis methods that the observations are normally distributed, the residuals were tested by a normal probability plot and Kolmogorov-Smirnov normality test in SPSS statistical software for the aggregate objective function

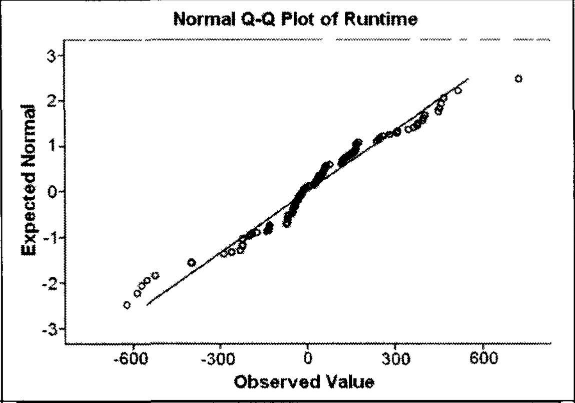
(Aggregate) and for the runtimes (Runtime). The results, presented in Figure 27, show that none of them follow a normal distribution ($p < 0.05$). Thus, the statistical significance of performance between the algorithms can be analyzed by the nonparametric Kruskal–Wallis test. Besides the two mentioned response variables of Aggregate objective function and Runtime, the Instability, calculated as a summation of both types of instability (expressions (11) and (12)), and the Inefficiency are analyzed by the test as well.

### 9.4.2 Results for the Arrival Time

Regarding the rush job arrival time, the results in Table 22 show that there is a statistical difference at its two levels in all of the response variables, namely: Runtime, Instability, Inefficiency and Aggregate.

Regarding the Runtime, at the arrival of a late job most of the operations will most likely be completed, and so, fewer operations remain to be rescheduled together with the new job, which takes less time for the RKGA to produce a reactive schedule. As for the instability, at the arrival time of a late job, most of the operations are completed and, therefore, fewer changes in the sequence or in the machine allocations will be needed in the reactive schedule. The best value in the inefficiency for early jobs is explained by the fact that the earlier the job arrives to the system, the more time it will have to complete by its due date, which was generated using the same parameters used for the rest of jobs.





**Tests of Normality**

| | Kolmogorov-Smirnov(a) | | |
|---|---|---|---|
| | Statistic | df | Sig. |
| **Residual for Agregate** | .158 | 150 | .000 |

a Lilliefors Significance Correction

**Tests of Normality**

| | Kolmogorov-Smirnov(a) | | |
|---|---|---|---|
| | Statistic | df | Sig. |
| **Residual for Runtime** | .142 | 150 | .000 |

a Lilliefors Significance Correction

**Figure 27. Normal Probability Plot for Aggregate Objective Function and Runtime. The Rush Job Experiment**

**Table 22. Results for the factor: Arrival Time**

| Response | Level | Mean Rank | P |
|---|---|---|---|
| Runtime | Early | 96.59 | 0.000 |
| | Late | 54.41 | |
| Instability | Early | 84.00 | 0.000 |
| | Late | 67.00 | |
| Inefficiency | Early | 61.7 | 0.000 |
| | Late | 89.3 | |
| Aggregate | Early | 65.93 | 0.004 |
| | Late | 85.07 | |

### 9.4.3 Results for the Rollback Level

Regarding the Rollback level, it cannot be concluded that there exists a significant difference between the three levels for the Aggregate objective function nor for the Inefficiency as shown in Table 23.

**Table 23. Results for the factor: Rollback Level**

| Response | Level | Mean Rank | P |
|---|---|---|---|
| | 0% | 92.32 | |
| Runtime | 10% | 69.49 | 0.003 |
| | 20% | 64.69 | |
| | 0% | 91.16 | |
| Instability | 10% | 67.00 | 0.000 |
| | 20% | 68.34 | |
| | 0% | 74.49 | |
| Inefficiency | 10% | 81.29 | 0.389 |
| | 20% | 70.72 | |
| | 0% | 83.81 | |
| Aggregate | 10% | 76.57 | 0.091 |
| | 20% | 66.12 | |

Concerning the Instability and the Runtime, the results show that there is a statistical difference at the different levels of rollback; however, the test does not reveal which means differ significantly. Thus, pair tests need to be performed.

The results of the additional tests presented in Table 24 show that there is no statistical difference between the levels of 10% and 20%, but both differ statistically from the level of 0%; the means suggest that both of them yield less instability than the level of 0%. Therefore, inserting a number of individuals in the initial population of the reactive problem with the genetic information of the predictive schedule seems to help the RKGA to build schedules similar to the predictive one in terms of sequence and machine allocation.

**Table 24. Additional tests for factor: Rollback Level and Instability as response variable**

| Response | Levels | Mean Rank | P | Levels | Mean Rank | P | Levels | Mean Rank | P |
|---|---|---|---|---|---|---|---|---|---|
| Instability | 0% | 58.50 | 0.000 | 10% | 50 | 0.317 | 0% | 58.16 | 0.000 |
| | 10% | 42.50 | | 20% | 51 | | 20% | 42.84 | |

Since the results of the Kruskal-Wallis test for Runtime in Table 23 show a statistical difference at the different levels of rollback, additional pair tests were conducted, and their results are shown in Table 25.

**Table 25. Additional tests for factor: Rollback Level and Runtime as response variable**

| Response | Levels | Mean Rank | P | Levels | Mean Rank | P | Levels | Mean Rank | P |
|---|---|---|---|---|---|---|---|---|---|
| Runtime | 0% | 58.59 | 0.005 | 10% | 52.58 | 0.473 | 0% | 59.23 | 0.003 |
| | 10% | 42.41 | | 20% | 48.42 | | 20% | 41.77 | |

According to the results of the additional tests, it cannot be concluded that there is a statistical difference between the levels of 10% and 20%; however, both of them differ statistically from the level of 0%. The means suggest that both take less runtime than the level of 0%. Therefore, inserting a number of individuals in the initial population of the reactive problem with the genetic information of the predictive schedule seems to speed up the search for a reactive solution.

It is important to notice here that the average runtime taken over the three rollback levels in Table 21 shows that the average time taken by the RKGA to generate a reactive solution at a 20% rollback level (224.4 seconds) is 39.4% of the time taken at a 0% of rollback level (568.9 seconds); and the average time taken by the RKGA to generate a reactive solution at a 10% rollback level (183.1 seconds) is 32.2% of the time taken at a 0% of rollback level, which emphasizes the importance of the rollback mechanism proposed in this dissertation in order to produce a reactive solution more rapidly. Although it cannot be concluded that there exists a significant difference between the rollback levels of 10% and 20%, according to the observed data, the level of 10% of rollback seems to be adequate.

# 10 CONCLUSIONS AND FUTURE RESEARCH

## 10.1 Conclusions

The main contribution of this dissertation is the introduction of a generic scheduling and rescheduling system with models and algorithms that can produce predictive (initial) schedules and react to disruptions in the most common production environments and objective functions. Namely, the introduced system can handle single machine, parallel machines, flow shops, and jobs shop scheduling problems or combination of these environments. The objectives can be regular such as the makespan, the total tardiness, total completion time, etc, and non-regular such as the total earliness and tardiness (ET) with a common due date or distinct due dates. The algorithms are based on the Random Keys Genetic algorithms (RKGA) that were introduced by Bean (1994) and intended to solve problems with regular objective functions. The combination of the generalized representation and the Earliness Reduction Procedure results in a more generalization of the RKGA where different environments and objective functions, not only regular but also the non-regular measure of Earliness and Tardiness, can be addressed by one scheduling system. To accomplish this generic system, several important modifications were introduced in this dissertation to the RKGA including changes in the chromosome and decoding procedure depending on the environment in which it is applied. Depending on the production environment (Single Machine, Parallel machines, Flow Shop, Job Shop or hybrids of them like Flexible Flow shop and Flexible Job Shop) information can be embodied with one type of chromosome and decoding procedure.

We also proposed and implemented a reactive RKGA that is able to produce a reactive schedule that minimizes the schedule's inefficiency and instability when different unexpected events occur in the various production environments targeted in this dissertation. To make the reactive RKGA more robust, it can repair schedules that are produced by other systems and not necessarily the ones initially produced by the RKGA itself.

The reactive RKGA system deals with the problem as multi-objectives optimization problem using the introduced instability ratios that measure the sequence deviation and machine allocation deviation, to minimize instability in addition to minimizing the inefficiency.

Among the most important original contributions of this research is the introduction of the rollback mechanism, by which the genetic information of the predictive schedule is inserted to a certain extent, in the initial population of the reactive RKGA. The computational experiments showed that the use of a rollback percentage greater than 0%, in our case 10% or 20%, reduces the runtime of the reactive model.

Computational experiments for both the predictive and the reactive RKGA were performed. Two types of problems were selected to test the predictive model, namely, the unrelated parallel machine scheduling problem $\mathrm{Rm}/\ /\mathrm{Cmax}$, and the Job Shop scheduling problem $\mathrm{Jm}/\ /\mathrm{ET}$. The computational experiments show that the model is able to produce optimal or near optimal schedules in several benchmark problems for the studied production environments and objective functions. In the case of $\mathrm{Jm}/\ /\mathrm{ET}$ the results show that the model performs very well especially for problems with loose and tight due dates. The test results of the reactive scheduling system showed that it is robust and capable of repairing schedules in a generic environment such as Flexible Job Shops. The statistical tests also demonstrate the various conditions under which the reactive RKGA is more efficient than others.

## 10.2 Future Research

The present research may be extended in several directions, as follows:

- The extension of the generalized predictive and reactive RKGA's to include additional constraints to the problems, such as sequence dependent setup times, and precedence constraints.
- New types of disruptions can be considered in the reactive model, such as due date changes and priority changes.

- From the reactive scheduling perspective, the weighted sum method may be used in an a posteriori approach to produce a set of non-dominated or good solutions to the decision maker.
- The Earliness Reduction Procedure may be used in conjunction with other metaheuristics such as Tabu Search, Simulated Annealing or Ant Colonies, for ET problems.
- Although in the case of $Jm/\ /ET$ the results show that the predictive model performs very well especially for problems with loose and tight due dates, additional experiments may be conducted to get insights into how to improve the performance in the case of problems with moderate due dates.
- Additional experiments with the predictive and reactive model, considering different release times, additional objective functions and environments, as well as other problem sizes, may be conducted.

## REFERENCES

Abumaizar, R.J., & Svestka, J.A. (1997). Rescheduling job shops under random disruptions. *International Journal of Production Research,* 35(7), 2065–2082.

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

Arnaout, J-P. (2006). A Robust Reactive Scheduling System with Application to Parallel Machine Scheduling. Doctoral Dissertation, Old Dominion University, Norfolk, VA.

Arnaout, J-P., & Rabadi, G. (2007). Predictable Scheduling With Learning Capability for the Unrelated Parallel Machine Problem. *International Journal of Operations & Quantitative Management,* 13(2), 115–127.

Arnaout, J-P., & Rabadi, G. (2008). Rescheduling of Unrelated Parallel Machines Under Machine Breakdowns. *International Journal of Applied Management Science,* 1(1), 75–89.

Artigues, C., Billaut, J.C. & Esswein, C. (2005). Maximization of solution flexibility for robust shop scheduling. *European Journal of Operational Research,* 165, 314–328.

Baker K. R. & Scudder G. D. (1990). Sequencing with earliness and tardiness penalties: A review. *Operations Research*, 38(1), 22-36.

Baker, K. R. (1974). *Introduction to Sequencing and Scheduling*. New York, NY: Wiley & Sons.

Baker, K. R., & Trietsch, D. (2009). *Principles of Sequencing and Scheduling*. John Wiley & Sons.

Bean, J. C. (1994). Genetic algorithms and random keys for sequencing and optimization. *ORSA Journal on Computing,* 2, 154–160.

Bean, J. C., & Birge, J. R. (1986). Match-up real-time scheduling. *Proceedings of the Symposium on Real-Time Optimization in Automated Manufacturing Facilities, National Bureau of Standards, Special Publication,* 724, 197–212.

Bean, J. C., Birge, J. R., Mittenthal, J. & Noon, C. E. (1991). Matchup scheduling with multiple resources, release dates and disruptions. *Operations Research,* 39(3), 470-483.

Blocher, J.D., Chhajed, D., & Leung, M. (1998). Customer order scheduling in a general job shop environment. *Decision Sciences,* 29 (4), 951-981.

Brandimarte P. (1993). Routing and scheduling in a flexible job shop by tabu search. *Annals of Operations Research*, 41, 157–183.

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

Demirkol, E., Mehta, S., & Uzsoy, R. (1998). Benchmarks for shop scheduling problems. *European Journal of Operational Research,* 109 (1), 137-141.

Erschler, J., Roubellat, F. & Vernhes, J. P. (1976). Finding Some Essential Characteristics of the Feasible Solutions for a Scheduling Problem. *Operations Research,* 24(4), 774-783.

Esswein, C., Billaut, J. C. & Strusevich, V. A. (2005). Two-machine shop scheduling: Compromise between flexibility and makespan value. *European Journal of Operational Research,* 167, 796–809.

Fonseca, C. M. & Fleming, P. J. (1998). Multiobjective optimization and multiple constraint handling with evolutionary algorithms - Part I:: A unified formulation. *IEEE Transactions on Systems, Man, and Cybernetics*, 28(1), 26–37.

French, S. (1982). *Sequencing and Scheduling: An Introduction to the Mathematics of the Job-Shop*. New York, NY: Wiley & Sons.

Garey M. R. & Johnson, D. S. (1979). *Computers and Intractability: A Guide to the Theory of NP-Completeness*. New York, NY: Freeman

Giffler, B. & Thompson, G.L., (1960). Algorithms for solving production scheduling problems. *Operations Research,* 8(4), 487-503.

Graham, R.L., Lawler, E.L., Lenstra, J.K. & Rinnooy Kan, A.H.G. (1979). Optimization and approximation in deterministic sequencing and scheduling: a survey. *Annals of Discrete Mathematics,* 5, 287-326.

Graves, S. C. (1981). A review of production scheduling. *Operations Research,* 29, 646-675.

Hall, N.G. & Potts, C.N. (2010).Rescheduling for job unavailability.Operations Research, 58(3) p 746-755.

Herrmann, J. (2006). Rescheduling Strategies, Policies, and Methods Using the rescheduling framework to improve production scheduling. In J. Herrmann (Eds.), *Handbook of Production Scheduling* (pp. 135-148) Springer, Heidelberg.

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

Jackson, J.R. (1956). An extension of Johnson's result on job lot scheduling. *Naval Research Logistics Quarterly,* 3(3), 201-203.

Jain, A.S. & Meeran, S. (1999). Deterministic Job-Shop Scheduling: Past, Present and Future. *European Journal of Operational Research,* 113(2), 390-434.

Johnson, S. M. (1954). Optimal Two and Three-Stage Production Schedules with Setup Times Included. *Naval Research Logistics Quarterly,* 1(1), 61-68.

Kanet, J. J. (1981), Minimizing the average deviation of job completion times about a common due date. *Naval Research Logistics Quarterly*, 28(4), 643-651.

MacCarthy, B. L. & Liu, J. (1993). Addressing the Gap in Scheduling Research: A Review of Optimization and Heuristic Methods in Production Scheduling. *International Journal of Production Research,* 31(1), 59-79.

Marler, R.T. & Arora, J.S.(2010). The weighted sum method for multi-objective optimization: new insights.Structural and Multidisciplinary Optimization, 41(6), 853-862.

Martello, S., Soumis, F. & Toth, P. (1997). Exact and approximation algorithms for makespan minimization on unrelated parallel machines. *Discrete Applied Mathematics*, 75, 169–188.

Mason, S.J., Jin, S. & Wessels, C. (2004). Rescheduling strategies for minimizing total weighted tardiness in complex job shops. *International Journal of Production Research,* 42(3), 613–628.

Mckay, K. N., Buzacott, J. A. & Safayeni, F. R., (1989). The scheduler's knowledge of uncertainty: The missing link. In J. Browne (Eds.), *Knowledge Based Production Management Systems* (pp.171-189) Amsterdam, Netherlands: Elsevier Science.

Mehta, S.V. & Uzsoy, R. (1998). Predictable Scheduling of a Job Shop Subject to Breakdowns. *IEEE Transactions on Robotics and Automation,* 14(3), 365-378.

Mehta, S.V. & Uzsoy, R. (1999). Predictable scheduling of a single machine subject to breakdowns. *International Journal of Computer Integrated Manufacturing,* 12, 15-38.

Mendes, J.J.M., Gonçalves, J.F. & Resende, M.G.C. (2009). A random key based genetic algorithm for the resource constrained project scheduling problem. *Computers & Operations Research,* 36(1), 92–109.Montgomery, D.C. (2009). *Design and Analysis of Experiments.* 7th Edition. John Wiley & Sons

Montgomery, D.C. (2009) *Design and Analysis of Experiments.* 7th Edition. John Wiley & Sons.

Moore, J.M. (1968). A n-job, one machine sequencing algorithm for minimizing the number of late jobs. *Management. Science*, 15(1), 102–109.

Moratori, P., Petrovic, S. & Vázquez, A. (2008). Match-Up Strategies for Job Shop Rescheduling. In N.T. Nguyen, L. Borzemski, A. Grzech, and M. Ali (Eds.). *New Frontiers in Applied Artificial Intelligence* (pp. 119–128) Springer Berlin/Heidelberg.

Muth, J. F. & Thompson, G. L. (1963). *Industrial Scheduling*. Prentice Hall

Norman, B. & Bean, J. (1997). Random keys genetic algorithm for job-shop scheduling. *Engineering Design & Automation,* 3(2), 145-156.

Okada, I., lin, I. & Gen, M. (2009). Solving Resource Constrained Multiple Project Scheduling Problems by Random Key-Based Genetic Algorithm. *Electronics and Communications in Japan,* 92(8), 25-35.

Pfund, M., Fowler, J.W. & Gupta, J.N.D. (2004). A survey of algorithms for single and multi-objective unrelated parallel-machine deterministic scheduling. *Chinese Journal of Industrial Engineers*, 21, 230–241.

Pinedo, M. (2008). *Scheduling theory, algorithms and systems*. Third edition. New York, NY: Prentice Hall.

Potts, C.N. (1985). Analysis of a linear programming heuristic for scheduling unrelated parallel machines. *Discrete Applied Mathematics,* 10, 155-164.

Rabadi, G., Mollaghasemi, M., & Anagnostopoulos, G.C. (2004). A Branch-and-Bound Algorithm for the Early/Tardy Machine Scheduling Problem with a Common Due-Date and Sequence-Dependent Setup Time. *Computers & Operations Research Journal*, 31(10), 1727-1751.

Sipper, D. Jr. & Bulfin, R.L. (1997). *Production: Planning, Control, and Integration*. New York, NY: McGraw-Hill.

Subramaniam V. & Raheja, A. S. (2003). mAOR: a heuristic based reactive repair mechanism for job shop schedules. *The International Journal of Advanced Manufacturing Technology,* 22, 669–680.

Subramaniam, V., Raheja, A. S. & Rama Bhupal Reddy, K. (2005). Reactive repair tool for job shop schedules. *International Journal of Production Research,* 43(1), 1-23.

Valente, J.M.S. & Gonçalves, J.F. (2009), A genetic algorithm approach for the single machine scheduling problem with linear earliness and quadratic tardiness penalties. *Computers and Operations Research,* 36(10), 2707-2715.

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

Wu, S. D., Storer H. S. & Chang P-C. (1993). One-machine rescheduling heuristics with efficiency and stability as criteria. *Computers and Operations Research,* 20, 1–14.

# VITA

ELKIN RODRÍGUEZ-VELÁSQUEZ

Engineering Management and Systems Engineering Department

Elkin Rodríguez-Velásquez received his Master of Sciences in Systems Engineering in 2000 and his Bachelor's degree in Industrial Engineering in 1997 from the Universidad Nacional de Colombia in Medellín. He has served as a graduate research assistant for the Engineering Management and Systems Engineering (EMSE) Department at Old Dominion University, Norfolk, VA. Elkin has had experience in the development of information systems in different organizations. He has been a consultant in the area of scheduling for manufacturing companies and professor in the Industrial Engineering program at the Universidad Nacional de Colombia. He has also authored publications in the areas of scheduling and modeling & simulation. His research interests are focused on the application of Artificial Intelligence tools, for solving scheduling and rescheduling problems.