Instituto Tecnológico y de Estudios Superiores de Monterrey

Campus Monterrey

School of Engineering and Sciences

Minimizing the total completion time in scheduling problems with sequence dependent setup times.

A dissertation presented by

Sarahí Berenice Báez Viezca

Submitted to the
School of Engineering and Sciences
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy

In

Engineering Science

Major in Industrial Engineering

Monterrey Nuevo León, May 21th, 2019

Dedication

To my parents, who have been my example to follow and main teachers in life.

To my husband, with whom I share this goal, who has patiently supported me in multiple ways, for years.

Acknowledgements

First of all, I thank God in whom I have decided to trust every stage of my life and now I can finish this thesis work.

To my thesis directors Dr. Francisco Roman Ángel-Bello and Dr. Ada Álvarez for their direction and support to develop this project and for their incredible human quality with which they have taught me.

To my thesis co-advisor, Dr. Belén Melián, for contributing her knowledge and guidance, as well as receiving me warmly during my stay in Tenerife with her family.

To the Dr. Jobish Vallikavungal for being part of my thesis committee and for its valuable analysis and contributions that allowed me to enrich this document.

Tecnológico de Monterrey and CONACyT for financial support to do my doctoral studies.

Many thanks to my family for their deep support and unconditional love, for being so comprehensive, never leave me alone and make me strong.

To Jorge and Adriana, for opening the doors of their home in Tenerife and teaching me invaluable life lessons. To Dayana and Natalie, thanks for your friendship and advice even in the distance.

Thank you, Ana, for your incredible friendship and be always present, thanks for the laughter no matter what.

Abdiel, thank you for accompanying me at each stage and helping me to enjoy each of them.

.

Scheduling problems with sequence dependent setup times minimizing the total completion time.

By

Sarahí Berenice Báez Viezca

Abstract

Objectives and study method: The main goal of this thesis work is to develop tools to solve some scheduling problems obtaining quality solutions in an efficient way, reaching improvements in the production times.
This work is dedicated to solving two kinds of scheduling problems. The first problem consists in giving a schedule for a set of jobs that should be processed in a set of machines. The term sequence dependent setup times means that the necessary time to get ready all the necessary to process a certain job depends on the job just performed. The second problem is actually four problems analyzed in two environments, and they are the single and parallel machine scheduling problem with learning and deterioration effects over sequence dependent setup times.

The addressed method in both cases is, firstly, developing a mathematical formulation for each one able to obtain optimal solutions and verify its scope. Secondly, it is designed and developed a heuristic algorithm that provides good solutions in short periods of time, for the first problem.

Contributions and conclusions: For the first problem, the computational experiments showed that time-dependent based formulations performed much better than the classic formulations and for the heuristic algorithm, this one has a better performance than the founded in the literature.

Regarding to the second problem, in this work we present the first mathematical formulation for the case with learning and deterioration effects overs the sequence dependent setup times.

Content

List of algorithms

List of figures

List of tables

Chapter 1 Introduction

1.1 Introduction

Nowadays, the problems in the industry are more complex each time. It is necessary to develop solution methods capable of achieving high-quality solutions, in very short computational times.

The scheduling problem is one of these industrial problems that have been extensively studied due to the multiple applications in several manufacturing systems. For example, steel, textile, painting, and plastic industries are the major industries requiring highly skilled scheduling.

The scheduling problem basically consists of assigning resources (for example machines) to a set of jobs that should be performed in such a way that the process is optimized, in terms of time, cost or even number of resources.

Bektas (2006) gives some examples of scheduling problems in the production area like print press scheduling and hot rolling scheduling as applications of the multi- traveling salesman problem where the setup costs in the production sequence are equivalent to the costs between cities.

There are many variants for the scheduling problems according to the environment of the manufacturing or service process.

Some kinds of scheduling problems according to the environment are job shop, flow shop, single machine, parallel machines, just to mention some.

Another distinctive feature is the objective of the system, which can be reducing costs, time, delays, etc. And even there are many ways to address the same element to improve. For example, when it is desired to minimize the time of production, one approach could minimize the completion time of the final job sequenced at the schedule (Makespan) or minimize the sum of the completion times of all the jobs (TCT).

Therefore, the problems dealt with in this work are the parallel scheduling problem with sequence dependent setup times and the parallel and a single machine scheduling problem with learning/deterioration effects over the sequence dependent setup times. In both problems, the objective is to minimize the Total Completion Time.

Taking into account the areas of research opportunity obtained from the literature review, the proposed solution strategies for the first problem are to develop mixed integer formulations capable of solving medium-sized instances and to design and implement efficient metaheuristic algorithms. For the second kind of problems (with learning or deterioration effects) the strategy is to develop mixed integer formulations that allow us to study the effects of learning and deterioration in scheduling problems with sequence-dependent setup times.

1.2 Motivation

The initial objective of this research was to study the effect of learning or deterioration on the total completion time in parallel machine problems with sequence dependent setup times. Although the scheduling problems with sequence dependent setup times and sequence dependent costs have been studied since the mid-sixties (Allahverdi, Gupta, & Aldowaisan, 1999) and continue to be extensively studied (Allahverdi et al., 2008; Allahverdi, 2015), we found that the problem of minimizing the TCT in parallel machine scheduling with sequence dependent setup times had not been practically addressed in the literature. For this reason, in this research we first treat the problems without learning or deterioration effect in order to develop efficient mathematical formulation and solution methods and then to study these problems considering learning or deterioration effects.

1.3 Problem Statement and Context

The two addressed problems of this work are scheduling problems where all the jobs are required to have a single operation. In the first problem, the machines work in parallel and all of them have the same production speed. In the second problem, we studied two machine environments, single machine and parallel machines. In both problems, setup depends on the job to be processed and the immediately preceding job; this is sequence dependent setup times. The objective function for the two addressed problems is minimize the Total Completion Time.

Problem 1: Parallel machines scheduling problem with sequence dependent setup times, minimizing the Total Completion Time.

The problem is to assign n independent jobs to m identical parallel machines and to determine the order in which jobs should be processed by the machines in such a way that the sum of jobs' completion time is minimized, that is, the objective is to minimize the total completion time. Each job has associated a processing time and there are machine setup times that depend on the order in which the jobs are processed. All the machines are in an initial state represented by a dummy job. The dummy job does not have processing time, but there is a setup time to prepare the machines to process a job just after the dummy job. Each machine can process one job at a time without preemption, that is, once the processing of a job has started, it cannot be interrupted. All the machines must be used and they do not have availability restrictions.

Problem 2: Single and Parallel machines scheduling problem with sequence dependent setup times, and learning and deterioration effects, minimizing the Total Completion Time.

The second addressed problem has the characteristics of problem 1. Furthermore, it considers deteriorating and learning effects over the sequence dependent setup times. These effects are also studied in a single machine environment.

1.4 Research Question

The general objectives of this work are, firstly, contributing to the state of art for the parallel machines scheduling problem with sequence dependent setup times minimizing the total completion time, and for the scheduling problems with learning and deteriorating effects over sequence dependent setup times for one and parallel machines. Secondly, developing efficient tools to solve this kind of problems for the industry, specifically, tools like mathematical models or heuristic algorithms.

Therefore, we derivate some questions about the solution methods proposed in this work.

1. Will the new time-dependent integer formulations proposed here for problem one perform better than the classic formulation based on a formulation for the multi-Travelling Salesman Problem?
2. Will the algorithm designed for problem one be able to obtain high quality solutions in reasonable computational?
3. Will the proposed formulation for the parallel scheduling problem with learning and deteriorating effect over sequence dependent setup times, minimizing the total completion time allow us to find optimal solutions through a commercial solver for some instances?
4. Will the optimal solutions found for the first problem still be optimal for the parallel machines environment of the second addressed problem?

1.5 Solution overview

Our approach for solving the problems tackled in this work is the following:

For the first problem, we propose a new time-dependent integer formulation with a better performance than other formulations developed from formulations for the multi-Travelling Salesman Problem (m-TSP) or for minimizing the makespan in parallel machine scheduling problems. In addition, we design and implement a hybrid metaheuristic algorithm able to obtain high-quality solutions in reasonable computational time for the underlying problem.

The second addressed problem considers learning and deteriorating effects. We develop two mathematical formulations, one for the single machine problem and one for the parallel machine problem. Using these formulations, it is possible to addressed the learning effect and the deterioration effect on sequence dependent setup times.

1.6 Dissertation Organization

The present thesis work is structured as follows in the next chapters:

Chapter 2. It is presented a literature review considering the relevant aspects of both problems and works related to the solution approach.

Chapter 3. Two formulations for minimizing the total completion time for a parallel machine scheduling problem considering sequence dependent setup times are proposed and compared with the classical formulation based on an m-TSP.

Chapter 4. It is proposed a metaheuristic algorithm designed to solve the parallel machines scheduling problem with sequence dependent setup times, minimizing the total completion time. The performance is compared with the exact method discussed in chapter 3 for medium instances. Furthermore, it is compared with a previous work founded in the state of art.

Chapter 5. This chapter deals with the single and parallel machines problems with learning and deteriorating effects over sequence dependent setup times. Two mathematical formulations are proposed and the computational analysis is presented here.

Chapter 6. Mention the general conclusions made throughout this research. Moreover, it is presented some of the possible practical applications and future research opportunities to the works introduced in this research.

Chapter 2 Literature review

Scheduling is a decision-making process that is used on a regular basis in many manufacturing and services industries. It deals with the allocation of resources to jobs over given time periods and its goal is to optimize one or more objectives (Pinedo, 2016).

The different kinds of environments and the variety of constraints involved with the necessary resources or with the jobs make that the scheduling problem has several variants. Because of this, this chapter presents a summary of the research developed in parallel scheduling problem considering sequence dependent setup times minimizing the Total Completion Time (TCT) and the case when the setup times increase as the number of sequenced jobs increases (deterioration effect) and decrease as the number of sequenced jobs decreases (learning effect).

In section 2.1 we present the classification of scheduling problems and their respective notation proposed by Graham, Lawler, Lenstra, and Kan, (1979). In section 2.2, we discuss some works related to the problem one, while in section 2.3 we present some adaptations of mixed integer formulations from the literature that are related to that problem. In section 2.4 we review some works related to scheduling problem with learning and/or deteriorating effects. Finally, in section 2.5 we describe some works related to the metaheuristics procedures used in this research.

2.1 Classification of scheduling problems

Scheduling problems are classified based on a number of factors including the number of jobs' stages to process them, the number of machines at each stage, job processing requirements, setup time/cost requirements, and the performance measure to be optimized. To be able to refer to the scheduling problems in a concise way, Graham, Lawler, Lenstra, and Kan, (1979) proposed the following notation. This notation was adapted by Allahverdi et al., (1999) in the first of his three scheduling surveys (Allahverdi, 2015; Allahverdi et al., 1999; Allahverdi, Ng, Cheng, and Kovalyov, 2008). The notation consists of three fields $\alpha\,|\beta|\gamma$, where α denotes the machine environment, β describes the job and machine characteristics and γ represents the performance measure to be optimized.

For the first field (α), some kinds of shop environments are:

- Single machine (1)
- Identical machines in parallel (P)
- Machines in parallel with different speeds (Q)
- Unrelated machines in parallel (R)
- Job shop (J)
- Flow shop (F)

The field (β) could have more than one entry according to the specifications of the particular problem. For example, if the problem considers sequence dependent setup times $(STsd)$ and release times for the jobs (r_j), then the second term is $\alpha \mid STsd, r_j \mid \gamma$

Examples of the job and machine characteristics that can be in β fielf of the classification:

- Sequence dependent setup cost ($SCsd$),
- Sequence dependent setup times ($STsd$),
- Release date (r_j),
- Precedence constraints ($Prec$),
- Due dates (d_j).

The last field (γ) is the used to describe the objective of the problem, this could be:

- Maximum completion time ($Cmax$),
- Total completion time ($\sum C_j \ or \ TCT$),
- Maximum Lateness ($Lmax$),
- Number of tardy jobs ($\sum U_j$).

Using this classification scheme, the first problem addressed in this research is referred as $P|\ STsd|\ TCT$.

2.2 The $P|\ STsd|\ TCT$ Scheduling problems.

From the literature review we observed that there are few works dealing with scheduling problems that simultaneously consider parallel machines, sequence dependent setup times and the minimization of the total completion time or total flow time. We will see each of these characteristics in this section in detail.

When the workshop contains more than one machine to perform the same operation, it is said that the machines are working in parallel. This set of machines is classified as identical (P), when all the machines have the same speed; as uniform (Q), when have different speed or completely unrelated (R).

A setup time is the time required to prepare the necessary resources (people, machines) to perform a job (Allahverdi and Soroush, 2008). In different situations, the setup time varies depending on the sequence of jobs performed in a machine. Chou, Wang, and Chang (2009) provide some examples of this fact, such as in the chemical, pharmaceutical and metal processing industries, where cleaning or fixing tasks should be performed to get ready the equipment to perform the next job.

There are several performance criteria to measure the quality of a scheduling. The most broadly used criteria is the minimization of the maximum completion time (makespan), the minimization of the sum of all completion times (TCT) and the minimization of some kind of tardiness. In particular, the minimization of the TCT is a criterion that contributes to the maximization of the production flow, the minimization of the work-in process inventories and balanced usage of resources.

Guinet (1991) is one of the first researchers that proposed heuristic algorithms for minimizing the mean flow time and the mean tardiness in parallel machine scheduling problems with sequence dependent setup times. In this research, he studied different problems of sequencing jobs in parallel processor shops of a textile company with sequence dependent machine changeover times.

There are many researches that considered parallel machines problem with sequence dependent setup times and the objective of minimizing the Total Weighted Completion Time (TWCT). Fan and Tang (2006) studied a scheduling problem of minimizing the TWCT on identical parallel machines considering sequence dependent setup times. They developed a column generation algorithm that can solve problems up to 10 machines and 60 jobs. Weng, Lu, and Ren (2001) addressed the problem of scheduling a set of independent jobs on unrelated parallel machines with sequence dependent setup times to minimize a weighted mean completion time. They proposed and tested seven heuristic algorithms. Fowler, Horng, and Cochran (2003) studied parallel machine scheduling problems with sequence dependent setup times considering as objective function the makespan, the TWCT and the Total Weighted Tardiness (TWT). They developed a genetic algorithm to assign jobs to machines and applied single machine dispatching rules to each machine to obtain the sequences.

A related objective to TWCT is the Total Weighted Tardiness (TWT). Driessel and Mönch (2011) proposed a Variable Neighborhood Search (VNS) to solve a parallel machines scheduling problem with sequence dependent setup times with precedence constraints and ready times, minimizing the TWT. The initial solution is constructed using the rule of apparent tardiness cost with setups and ready times and in cases where different schedules with the same TWT value are obtained, they used the makespan as a tie breaker. Schaller (2014) considered the problem of scheduling on parallel machines with family setup times to minimize total tardiness (TT). They proposed solution methods based on tabu search and genetic algorithms for that problem.

Many researchers considered TCT as the objective function along with additional constraints like job release dates, precedence constraints or machine eligibility constraints. Nessah, Chu, and Yalaoui (2007) addressed an identical parallel machine scheduling problem to minimize TCT with sequence dependent setup times and release dates. For this problem, they proved a dominance theorem, developed a lower bound and proposed an efficient heuristic procedure that is incorporated in a branch and bound algorithm (B&B). Their computational experiments showed that the B&B algorithm solved instances up to 40 jobs and 2 machines. Gacias, Artigues, and Lopez (2010) studied a parallel machine scheduling problem with sequence dependent setup times and precedence constraints with the objective of minimizing the TCT. They proposed dominance conditions, defined an exact B&B procedure and a climbing discrepancy search heuristic.

To solve the identical parallel machine scheduling problem with job deadlines and machine eligibility constraints minimizing the TCT, Su (2009) designed a B&B algorithm and a heuristic algorithm. The heuristic algorithm assigns jobs to available machines one-by-one combining the shortest processing time rule and the minimum slackness rule to do the assignment. The heuristic is used as an initial upper bound to the B&B algorithm. Lee, Liao, and Chao (2014) addressed a real-life scheduling problem in the manufacturing industry. They modelled the problem considering parallel machines, sequence-dependent setup times, dedicated machines constraints and a common deadline for all jobs, and developed heuristic methods to solve it. Joo and Kim (2015) considered an unrelated parallel machine scheduling problem with sequence and machine dependent setup times, machine dependent processing times, and production availability constraints. They proposed a mathematical model to find an optimal solution and a hybrid genetic algorithm.

For parallel scheduling problems with sequence dependent setup times and the objective function of minimizing the TCT, we only found one work reported in the literature. Morales, M. F. (2015) developed a first metaheuristic algorithm for the addressed problem. She implemented an Iterated Greedy Algorithm (IGA) that used as diversification generator which consists of a destructive-constructive process and as improvement phase a composited local search procedure based on intra-machine and inter-machine moves by means of relocation and interchange of jobs.

2.3 Mixed integer formulations

It is known that the Asymmetric Travelling Salesman Problem (ATSP) is a general case of a scheduling problem where the asymmetric matrix of sequence dependent setup times s_{ij} is equivalent to the asymmetric distance matrix and the job processing times p_j are equivalent to the client service times. For that reason, in this section, we show some mixed integer formulations related to the multiple *ATSP* that can be modified to represent the $P|\,STsd|\,TCT$ problem.

2.3.1 Assignment-based integer programming formulation

Bektas (2006) presented an assignment based double-index integer linear programming formulation for the multiple-Traveling Salesmen Problem (*m-TSP*) using the Miller-Tucker-Zemlin Subtour Elimination Constraints (Miller, Tucker, and Zemlin, 1960) and defining binary variables x_{ij}:

$$x_{ij} = \begin{cases} 1, & \text{if the arc } (i,j) \text{is used in the path} \\ 0, & \text{otherwise} \end{cases}$$

The model is:

$$\boldsymbol{min\ z = \sum_{i=1}^{n}\sum_{j=1}^{n} c_{ij}x_{ij}} \qquad (\mathbf{2.1})$$

subject to:

$$\boldsymbol{\sum_{j=2}^{n} x_{1j} = m} \qquad (\mathbf{2.2})$$

$$\boldsymbol{\sum_{j=2}^{n} x_{j1} = m} \qquad (\mathbf{2.3})$$

$$\boldsymbol{\sum_{i=0,i\neq j}^{n} x_{ij} = 1} \qquad (\boldsymbol{j = 2,\dots,n}) \qquad (\mathbf{2.4})$$

$$\boldsymbol{\sum_{j=1}^{n} x_{ij} = 1} \qquad (\boldsymbol{i = 2,\dots,n}) \qquad (\mathbf{2.5})$$

$$\boldsymbol{u_j - u_j + px_{ij} \leq p - 1} \qquad (\boldsymbol{2 \leq i \neq j \leq n}) \qquad (\mathbf{2.6})$$

$$\boldsymbol{x_{ij} \in \{0,1\}} \qquad (\boldsymbol{\forall i,j \in A}) \qquad (\mathbf{2.7})$$

The objective function (2.1) minimizes the total travelled distance. Constraints (2.2) and (2.3) ensure that exactly *m* salesmen depart from and return back to node 1. Constraints (2.4) and (2.5) maintain the flow of the route, while constraints (2.5) are the subtour elimination constraints (SECs) developed by (Miller et al., 1960). In these constraints, the continuous variables $\boldsymbol{u_i}$ that indicates the order of the node *i* in the tour, and p is a parameter to limit the number of nodes that can be visited by any salesman.

2.3.2 A flow-based formulation

Another mixed integer formulation presented in Bektas, (2006) is the adaption for the m-TSP to the three-index formulation proposed by Christofides, Mingozzi, and Toth (1981) for Vehicle Routing Problem (VRP) and based on the Miller-Tucker-Zemlin Subtour Elimination Constraints (MTZ-SECs).

$$x_{ijk} = \begin{cases} 1, & \text{if the vehicle } k \text{ visits the node } j \text{ immediately after node } i \\ 0, & \text{otherwise} \end{cases}$$

$$\boldsymbol{min\ z} = \sum_{i=1}^{n}\sum_{j=1}^{n} \boldsymbol{c_{ij}} \sum_{k=1}^{m} \boldsymbol{x_{ijk}} \qquad (\boldsymbol{2.8})$$

$\boldsymbol{subject\ to}$:

$$\sum_{i=1}^{n}\sum_{k=1}^{m} \boldsymbol{x_{1jk}} = \boldsymbol{1} \qquad (\boldsymbol{j = 1, \dots, n}) \qquad (\boldsymbol{2.9})$$

$$\sum_{i=1}^{n} \boldsymbol{x_{ipk}} - \sum_{j=1}^{n} \boldsymbol{x_{pjk}} = \boldsymbol{0} \qquad (\boldsymbol{k = 1, \dots, m, p = 1, \dots, n}) \qquad (\boldsymbol{2.10})$$

$$\sum_{j=1}^{n} \boldsymbol{x_{1jk}} = \boldsymbol{1} \qquad (\boldsymbol{k = 1, \dots, m}) \qquad (\boldsymbol{2.11})$$

$$\boldsymbol{u_i - u_j + n\sum_{k=1}^{m} x_{ij}} \leq \boldsymbol{n - 1} \qquad (\boldsymbol{i \neq j = 2, \dots, n}) \qquad (\boldsymbol{2.12})$$

$$\boldsymbol{x_{ijk}} \in \{\boldsymbol{0, 1}\} \qquad (\forall \boldsymbol{i, j, k}) \qquad (\boldsymbol{2.13})$$

The objective function (2.8) minimizes the sum of distances travelled by the salesmen. Constraints (2.9) assure that each node is visited exactly once. Constraints (2.10) are the flow conservation constraints and they ensure that once a salesman visits a customer, then he must also depart from the same customer. Constraints (2.11) verify that each vehicle is used exactly once, while constraints (2.12) represent the adaption of the MTZ-SECs for a three- index model.

2.3.3 The single commodity flow formulation

Gavish and Graves (1978) extended the formulation of Miller et al., (1960) for the Travelling Salesman Problem (TSP) to the Multi-Travelling Salesman problem. In this formulation, the next variables are defined:

$$x_{ij} = \begin{cases} 1, & \text{if the arc } (i,j) \text{is used in the route} \\ 0, & \text{otherwise} \end{cases}$$

y_{ij}: the location of the arc (i,j) in the tour

$$\boldsymbol{min\ z = \sum_{i=0}^{n}\sum_{j=0}^{n} c_{ij}x_{ij}} \qquad (\mathbf{2.14})$$

subject to:

$$\sum_{i=0}^{n} \boldsymbol{x_{ij} = 1} \qquad (\boldsymbol{j = 1, \dots, n}) \qquad (\mathbf{2.15})$$

$$\sum_{j=0}^{n} \boldsymbol{x_{ij} = 1} \qquad (\boldsymbol{i = 1, \dots, n}) \qquad (\mathbf{2.16})$$

$$\sum_{i=0}^{n} \boldsymbol{x_{i0} = m} \qquad (\mathbf{2.17})$$

$$\sum_{j=0}^{n} \boldsymbol{x_{0j} = m} \qquad (\mathbf{2.18})$$

$$\sum_{j=0}^{n} \boldsymbol{y_{ij}} - \sum_{j=0}^{n} \boldsymbol{y_{ji} = 1} \qquad (\boldsymbol{i = 1, \dots, n}) \qquad (\mathbf{2.19})$$

$$\boldsymbol{y_{ij} \leq (n - m + 1)x_{ij}} \qquad (\boldsymbol{2 \leq i \neq j \leq n}) \qquad (\mathbf{2.20})$$

$$\boldsymbol{x_{ij} \in \{0,1\},\ y_{ij} \geq 0} \qquad (\boldsymbol{\forall i,j \in A}) \qquad (\mathbf{2.21})$$

In this formulation, the objective function (2.14) minimizes the total travel costs. Constraints (2.15) and (2.16) ensure that at each node only arrives and depart one path, while (2.17) and (2.18) verify that exactly m paths depart and return to the node 0 (depot). Constraints (2.19) are the flow restrictions for each node i. Constraints (2.20) establish that if the arc (i,j) is in one path, then it will be at most in the $n - m + 1$ position, where the value of $n - m + 1$ force to have exactly m paths.

All the above formulations can be modified to model the $P|\ STsd|TCT$ problem. The adaptation process is shown later in Chapter 3.

2.4 Scheduling with learning and deterioration effects.

Time variations due to frequent repetition of operations are well known in scheduling literature as learning or deterioration effects. To the best of our knowledge, the concept

of learning effect in scheduling problems was introduced by Biskup (1999). After that, a large number of works has published the effects of learning and/or deterioration on processing times in scheduling problems.

In general, the theory of learning effects states that the time needed to produce a single unit continuously decreases with the processing of additional units and that due to the declining processing times the costs per unit also decline (Biskup, 1999). Also, there are many situations in which a job that is processed later in a machine sequence consumes more time than the same job when it is processed earlier (Wu and Lee, 2008), this is known as deterioration effect.

1 In the position-dependent approach, the time needed to produce a unit decreases (increases) as the number of repetitions of job (Biskup, 1999). The processing time $\boldsymbol{p_{jr}}$ of job j in position r is calculated as $p_{jr} = p_j f(a, r)$, where p_j is the processing time without learning effect (normal processing time), f is a decreasing (increasing) function with respect to r and a is a constant learning factor.

Some related researches are: Chen, Wu, and Lee (2006) and Wang, Wang, and Ji (2012), where the learning and deterioration effects over the processing times are defined by a function of their starting times. A more recent published work is Ji, Tang, Zhang, and Cheng (2016). The authors considered the minimization of the total completion time (TCT) and the makespan on a parallel-machine scheduling problem with deteriorating jobs and DeJongs learning effect (DeJong, 1957) for overcoming the drawback associated with the log-linear learning model. They showed that minimizing the TCT is polynomially solvable, while the minimization of the makespan is NP - hard.

Under group technology considerations, Xingong, Yong, and Shikun (2016) proposed a model for addressing deteriorating and learning effects in a single machine environment. They showed that the total completion time problem can be solved in polynomial time.

The processing times of the already processed jobs are important when human interactions are significant during the processing of the jobs. For those situations it might be more appropriate to consider a time-dependent learning/deterioration effect, due to the learning rate of the operator or the deterioration rate in the cases when deterioration affect the operator performance.

In the time-dependent approach of the processing times, the time needed to produce a unit decreases (increases) depending on the sum of the processing times of the all already scheduled jobs (Kuo and Yang, 2006). The processing time p_{jr} is calculated as $p_{jr} = p_j f(a, \sum_{k=1}^{r} p_{[k]})$, where $p_{[k]}$ means the normal processing time of job in position k and f decreases (increases) as the sum $\sum_{k=1}^{r} p_{[k]}$ increases.

Kuo and Yang (2006), introduced the time-dependent learning effect. They considered that the factor, affecting the normal processing time of the job, is modified by the inclusion of the sum of the processing times of the all already scheduled jobs in the following way:

$$p_{jr} = p_j \left(1 - \sum_{k=1}^{r} p_{[k]}\right)^a$$

Gawiejnowicz (2008) presented a detailed survey of scheduling problems involving time-dependent learning effects on processing times of the jobs.

Another kind of learning effect over processing times is the so-called past-sequence-dependent (p-s-d) setup times introduced by Koulamas and Kyparisis (2008). In the p-s-d setup time approach, the processing time p_{jr} is obtained by the normal processing time plus a value that depends on the sum of the processing times of the all already scheduled jobs that is, $p_{jr} = s_{[r]} + p_j$, where $s_{[0]} = 0$ and $s_{[r]} = b^r \sum_{k=1}^{r} p_{[k]}$ for $r = 2,3,\dots,n$ and b is a constant. The value $s_{[r]}$ could be interpreted as a setup time that depends upon the sum of the processing times of the all already scheduled jobs.

Wang and Wang (2013) and Lee (2014) studied a single machine scheduling problem with p-s-d setups and general effects of deterioration and learning that is, the actual job processing time is a general function of the processing times of the already processed jobs and its scheduled position. They showed that the problems of minimizing the makespan, the total lateness, and the total completion time are polynomially solvable. Salehi Mir and Rezaeian (2016) addressed the problem of scheduling on identical parallel machines with past-sequence-dependent (p-s-d) setup times and effects of deterioration and learning to minimize the total completion time of all jobs. They proposed two metaheuristic algorithms, based on artificial immune system and ant colony optimization, showing the second one the better performance of both.

For a recent review on learning and/or deterioration effects for the past sequence-dependent setup times, an interested reader is referred to Allahverdi (2015).

All the published works mentioned above do not consider setup times for the machines or that the setup times depend only on the job that is going to be processed and therefore they can be included in the job processing times. It is known that in several practical applications the setup times must be explicitly considered, for example in food processing, chemical, printing or metal processing industries, among others (Allahverdi et al., 1999).

On one hand, the explicit consideration of sequence dependent setup times in scheduling problems substantially increases the problem complexity and changes the structure of the solution process, making the adaptations of the existing methods for problems without setup times (Avalos-Rosales et al., 2018). Alternatively, it is known that tremendous savings can be achieved when setup times/costs have been explicitly included in scheduling decisions in various real world industrial/service environments (Allahverdi and Soroush, 2008).

2.5 Metaheuristic Algorithms

To solve the $P|\,STsd|TCT$ problem for large instances we designed and implemented a hybrid metaheuristic algorithm. The proposed algorithm is a hybridization of three well known metaheuristic algorithms: Greedy Randomized Adaptive Search Procedure (GRASP), Variable Neighborhood Search (VNS) and Iterated Greedy Algorithm (IGA).

The VNS is embedded into a multi-start strategy which is GRASP, to give more diversity to the hybrid algorithm and the shaking procedure in VNS is performed by applying systematically an IGA. Next we give a brief description of each of these metaheuristics.

2.5.1 GRASP

Greedy Randomized Adaptive Search Procedure is a multistart algorithm that was first introduced by Feo and Resende (1989). The GRASP consists of a constructive phase and an improvement phase. The best local optimum found over all GRASP iterations is saved as the best found solution (Feo and Resende, 1995).

Solution construction consists of inclusion of one element at a time in an iterative way to a partial solution under construction until a solution has been completed. At each iteration the constructive phase, the selection of the next element to be included in the partial solution is determined by the evaluation of all feasible candidate elements (candidate list) depending on the greedy evaluation function. A Restricted Candidate List (RCL) is formed with the best elements from candidate list (according to a parameter $\alpha, 0 \leq \alpha \leq 1$) and the element to be incorporated into the partial solution is randomly selected from RCL. When the selected element is incorporated to the partial solution, the candidate list is updated. This process is repeated until all the elements have been included in the solution.

The solution obtained in the constructive phase is used as starting solution for the improvement phase. In early versions of GRASP the improvement phase was made up of a local search. Later, metaheuristics like tabu search, iterated local search, VND or VNS, have been implemented as an improvement phase. For more details about GRASP see the survey by Resende and Ribeiro (2010). A generic pseudo-code for GRASP is shown in Algorithm 2.1.

Many works have used the GRASP to solve optimization problems. In the third survey by Allahverdi (2015) it can be found some works that applied the GRASP to solve the parallel scheduling problems with sequence dependent setup times. Kampke, Arroyo and Santos (2009) minimizes an objective function that combines the total completion time and the total number of resources assigned to the parallel unrelated machines with a reactive GRASP and incorporate a path relinking technique. Park and Seo (2013) deal with a transporter scheduling problem of ship assembly block operations management as a parallel machine scheduling with sequence-dependent setup times and precedence constraints due to their easy transformation of one on another. The objective is to maximize the workload balance among transporters. They develop a metaheuristic based on GRASP and conclude that this metaheuristic is promising for transporter scheduling problems. Armentano and de França Filho, (2007) minimize the total tardiness relative to the job due dates and the machine environment is uniform parallel machines. They incorporate adaptive memory principles into their GRASP to solve this problem.

```
// GRASP
Data: Problem data, stopping condition, α
Initialization: f(S_b) ← +∞
while (the stopping condition is not met) do
    (S, f(S)) ← ConstructivePhase(Problem data, α)
    (S', f(S')) ← ImprovementPhase(S, f(S), Problem data)
    if (f(S_b) > f(S')) then
        (S_b, f(S_b)) ← (S', f(S'))
    end
end
Result: (S_b, f(S_b))
```

Algorithm 2.1 Generic pseudo-code for GRASP

More recently, Bierwirth and Kuhpfahl (2017) has built a competitive algorithm based on GRASP that outperforms the state of the art algorithms for the job shop problem, and Molina-Sánchez and González-Neira (2016) incorporated two utility functions called Weighted Modified Due Date (WMDD) and apparent tardiness cost to solve the permutation flow shop.

2.5.2 VNS

The Variable Neighborhood Search (VNS) is based on the idea of a systematic change of neighborhood both in a descent phase to find a local optimum and in a perturbation phase to get out of the corresponding valley (Mladenović and Hansen, 1997).

According to Hansen, Mladenović, and Pérez (2008), VNS is based on three facts:

1) A local minimum with respect to one neighborhood structure is not necessarily a local minimum for another neighborhood structure.
2) A global minimum is a local minimum with respect to all possible neighborhood structures.
3) For many problems, local minima with respect to one or several neighborhoods are relatively close to each other.

The General Variable Neighborhood Search (GVNS) consists of three phases, shaking, local search and move or not. In the shaking, a neighbor solution from the incumbent solution is randomly chosen. After, the local search step is applied to this neighbor and the "move or not" step is performed as follows: if the neighbor solution found in the local search is better than the incumbent solution, then, this neighbor solution becomes the incumbent solution and the search continue in the first neighborhood; if the neighbor is not better than the incumbent, the search advances to the next neighborhood. This procedure is repeated until a stopping criterion is met. A generic pseudo-code for GVNS is shown in Algorithm 2.2.

```
// GVNS
Data: Problem data, a set of neighborhood structures for shaking
      N_k, (k = 1, 2, ..., K_max), a set of neighborhood structures for
      improvement N_l, (l = 1, 2, ..., L_max), an initial solution S and a
      stopping condition
S_b ← Apply an improvement procedure to S
while (the stopping condition is not met) do
    k ← 1
    while (k ≤ K_max) do
        Shaking: Generate a point S' at random from the k^th neighborhood
          N_k(S_b)
        l ← 1
        while (l ≤ L_max) do
            Improvement: Find the best neighbor S'' in N_l(S')
            if (f(S') > f(S'')) then S' ← S'', f(S') ← f(S''), l ← 1
            else l ← l + 1
        end
        if (f(S_b) > f(S')) then S_b ← S', f(S_b) ← f(S'), k ← 1
        else k ← k + 1
    end
end
Result: (S_b, f(S_b))
```

Algorithm 2.2 Generic pseudo-code for GVNS

To solve the parallel machines scheduling problem with sequence dependent setup times by applying VNS we have found two works that apply the VNS metaheuristic. Behnamian and Fatemi Ghomi (2011) presented a min-max bi-objective procedure for minimizing the makespan and the sum of the earliness and tardiness of jobs in due window machine scheduling problems, simultaneously. Driessel and Mönch, (2009) developed a procedure based on VNS to minimize the total weighted tardiness considering ready times of the jobs and precedence constraints.

Other work that deals with the parallel scheduling problem with sequence dependent setup times using VNS is Paula et al. (2007). The performance of their VNS algorithm is compared with three versions of a greedy randomized adaptive search procedure algorithm.

2.6 Conclusions

In this chapter we have presented some works related to the parallel scheduling problem with sequence dependent setup times and the scheduling problems with learning and deterioration effects. Also, we gave some highlights of the two metaheuristic procedures proposed in this research to solve the addressed problems.

Chapter 3 Mathematical formulations for the parallel scheduling problem with sequence dependent setup times.

3.1 Introduction

The problem addressed in this chapter is classified as $P|STsd|TCT$ using the classification scheme explained in Chapter 2.1, where P in the alpha field stands for Parallel machine environment, *STsd* is for sequence dependent Setup Times, and *TCT* is the acronym for Total Completion Time, that is the performance measure.

From the literature review, we observed that this problem has received very little attention in the specialized literature and that there are no reported mathematical formulations capable of solving medium-sized data instances. For this reason, in this chapter we addressed five mixed integer formulations of $P|STsd|TCT$ problem. First three models are obtained by adapting the existing ones of related problems and the last two are developed based on the problem as a variant of the Time Dependent multiple Traveling Salesmen Problem (TDmTSP).

First, we modify two classical formulations for the multiple Travelling Salesmen Problem (mTSP) adapting them to parallel scheduling problem with sequence dependent setup times. The first formulation is formulated from the classical two-index formulations for the mTSP (Bektas, 2006) by expressing the objective function and the sub-tour elimination constraints in terms of the TCT. The second formulation is obtained from a flow-based formulation to the mTSP (Gavish & Graves, 1978) by modifying the objective function to evaluate the TCT. To obtain the third one, we adapt a three-index formulation for minimizing the makespan in an unrelated parallel machine scheduling problem with sequence dependent setup times ($R|STsd|Cmax$). This formulation is an adapted version of the best formulation reported in the literature for the problem (Avalos-Rosales, Angel-Bello, and Alvarez, 2015). Finally, we propose two new formulations based on time dependent travelling salesman problem. These formulations are obtained as generalizations of time dependent formulations to the minimum latency problem (Angel-Bello, Alvarez, and García, 2013).

3.2 Problem formulation

The characteristics and the parameters involved in the problem addressed is presented below:

- There are m identical parallel machines, without preemption or availability restrictions.
- There are n independent jobs to be scheduled in the machines. All jobs are available at time zero.
- Each job j has an associated processing time p_j.
- There are machine setup times s_{ij} for processing a job j just after job i, with $s_{ij} \neq s_{ji}$, in general. There is a setup time s_{0j} for processing the first job on each machine.

The problem consists of assigning n jobs to m machines and determining the order in which the jobs should be processed on the machines in such a way that the sum of the jobs' completion times is minimized, this is, to minimize the *TCT*.

The problem can be defined on a complete graph $G = (V, A)$, where $V = \{0, 1, 2, \dots, n\}$ is the nodes set and A is the arcs set. The node 0 represents the initial state of the machines (dummy job) and the nodes in the set $I = \{1, 2, \dots, n\}$ correspond to the jobs. For each pair of nodes $\{i, j\}$ in V there are two arcs $\{(i, j), (j, i)\} \in A$ that have associated the setup times s_{ij} and s_{ji}, respectively. Each node $j \in V$ has associated a processing time, p_j. Since 0 represents the dummy job, the corresponding processing time is set to 0 (i.e., $p_0 = 0$). Using the setup times s_{ij} and the processing times p_j, we associate to each arc $(i, j) \in A$ the sum of the time required to prepare the machine and to process the job j just after the job i, this is the value $t_{ij} = s_{ij} + p_j, (i \in V, j \in I)$.

Let $P_r = \{0, 1_r, 2_r, \dots, k_r\}$ denote a sequence with $k_r + 1$ jobs in machine r containing the dummy job 0 in the position zero of P_r. We use the notation $[i_r]$ to represent the i-th job in the sequence r. Then, the value $t[i][j]$ is the sum of the time required to prepare the machine and to process the job in the j-th position just after the job in the i-th position. The completion time $C[i_r]$ of the job in the position i_r is calculated as $C[i_r] = \sum_{j=1}^{i_r} t[j-1][j]$. Note that, on the graph G it represents the length of the path from node 0 to node $[i_r]$. Then, the TCT of the sequence P_r is calculated as $TCT(P_r) = \sum_{j=1_r}^{k_r} C[j] = C[1_r] + C[2_r] + \cdots + C[k_r]$.

Thus, the problem is formulated as to find m disjoint simple paths in G starting at source node 0 that together cover all the nodes in I and minimize the objective function, that can be stated as:

$$\boldsymbol{min\ z = \sum_{r=1}^{m} TCT(P_r) = \sum_{r=1}^{m} \sum_{j=1_r}^{k_r} C[j]} \qquad (\mathbf{3.1})$$

That is, the problem is to find m disjoint job sequences (a sequence for each machine) with dummy jobs at the position zero that together contain all the jobs and minimize the sum of the jobs' completion times.

3.3 Formulations derived from $mTSP$ and $R|\ STsd|\ Cmax$

3.3.1 Assignment-based integer programming formulation for TCT

The first formulation is adapted from a formulation for the $P|\ STsd|\ Cmax$ problem that is obtained directly from the two-index formulation of the mTSP presented in Bektas (2006). Our fundamental modification to the $P|\ STsd|\ Cmax$ formulation is to express the objective function and the sub-tour elimination constraints in terms of the completion times.

To do that, we define the following variables:

$$x_{ij} = \begin{cases} 1, & \text{if the job j is processed just after job i} \\ 0, & \text{otherwise} \end{cases}$$

C_i: the completion time of job i

Then, the formulation for the $P|\,STsd|\,TCT$ problem is as follows:

Model 1

Minimize $\boldsymbol{z}$, where:

$$\boldsymbol{min\ z = \sum_{i=1}^{n} C_i} \qquad (\mathbf{3.2})$$

Subject to:

$$\boldsymbol{\sum_{j=1}^{n} x_{0j} = m} \qquad (\mathbf{3.3})$$

$$\boldsymbol{\sum_{j=1}^{n} x_{j0} = m} \qquad (\mathbf{3.4})$$

$$\boldsymbol{\sum_{i=0, i\neq j}^{n} x_{ij} = 1 \qquad (j = 1, \dots, n)} \qquad (\mathbf{3.5})$$

$$\boldsymbol{\sum_{j=0, j\neq i}^{n} x_{ij} = 1 \qquad (i = 1, \dots, n)} \qquad (\mathbf{3.6})$$

$$\boldsymbol{t_{0i}x_{0i} \leq C_i \leq t_{0i}x_{0i} + T(1 - x_{0i}) \qquad (i = 1, 2, .., n)} \qquad (\mathbf{3.7})$$

$$\boldsymbol{C_i - C_j + (T + t_{ij})x_{ij} + (T - t_{ji})x_{ji} \leq T \qquad (i, j = 1, 2, .., n; j \neq i)} \qquad (\mathbf{3.8})$$

$$\boldsymbol{x_{ij} \epsilon \{0, 1\} \ \ \forall (i, j) \in A\,, C_i \geq 0 \quad (i = 1, \dots, n)} \qquad (\mathbf{3.9})$$

Constraints (3.3) and (3.4) establish that there must be $\boldsymbol{m}$ initial and $\boldsymbol{m}$ final jobs scheduled according to the number of machines. Constraints (3.5) and (3.6) are the assignment constrains and they guarantee that each job has a single predecessor and a single successor in the sequences. Constraints (3.7) initialize the value of C_i equal to t_{0i} when the job i is the first job in a sequence and otherwise they are redundant. Constraints (3.8) are the sub-tour elimination constraints in terms of the completion times. They calculate the value of $C_j = C_i + t_{ij}$ when $x_{ij} = 1, x_{ji} = 0$ or $C_i = C_j + t_{ji}$ when $x_{ij} = 0, x_{ji} = 1$. If $x_{ij} = 0, x_{ji} = 0$ they are redundant. Constraints (3.9) define the nature of the variables.

3.3.2 The single commodity flow formulation for TCT

Before giving the second formulation, it is represented an expression to evaluate the TCT of a sequence of jobs in terms of the position they occupy in the sequence. The

completion time of the jobs for each position in any sequence P_r with k jobs is given by:

$$\begin{aligned}
&\boldsymbol{C_{[1]} = t_{[0][1]}}\\
&\boldsymbol{C_{[2]} = t_{[0][1]} + t_{[1][2]}}\\
&\quad \ldots\\
&\boldsymbol{C_{[k-1]} = t_{[0][1]} + t_{[1][2]} + \cdots + t_{[k-2][k-1]}}\\
&\boldsymbol{C_{[k]} = t_{[0][1]} + t_{[1][2]} + \cdots + t_{[k-2][k-1]} + t_{[k-1][k]}}
\end{aligned}$$

Adding the above expressions, we obtain the TCT for the sequence P_r which can be expressed as:

$$\boldsymbol{TCT(P_r) = \sum_{i=1}^{k} C_{[i]} = kt_{[0][1]} + (k-1)t_{[1][2]} + \cdots + 2t_{[k-2][k-1]} + t_{[k-1][k]}}$$

This is:

$$\boldsymbol{TCT(P_r) = \sum_{i=1}^{k} (k - i + 1)t_{[i-1][i]}} \qquad (\mathbf{3.10})$$

From equation (3.10) it can be seen that the contribution to the TCT of the job in the position q is equal to $(k - q + 1)t_{[q-1][q]}$, that is, it is equal to the number of jobs in the sequence from the job in the position q multiplied by the value of time $t_{[q-1][q]} = s_{[q-1][q]} + p_{[q]}$ (the setup time of the machine to process the job $[q]$ just after the job $[q-1]$ plus the processing time of job $[q]$). Using (3.10) the objective function (3.1) can be expressed as:

$$\mathbf{min}\ \boldsymbol{z = \sum_{r=1}^{m} TCT(P_r) = \sum_{r=1}^{m} \sum_{j=1_r}^{k_r} C[j] = \sum_{r=1}^{m} \sum_{j=1_r}^{k_r} (k_r - j + 1)t_{[j-1][j]}} \qquad (\mathbf{3.11})$$

The second model for the $P|\,STsd|\,TCT$ problem is adapted from the single commodity flow formulation of Gavish and Graves (1978) for mTSP. This formulation uses the binary variables x_{ij} defined for the first formulation and, new integer variables:

$$f_{ij} = \begin{cases} \text{number of jobs in a machine after job i,} & \text{if } x_{ij} = 1 \\ 0, & \text{if } x_{ij} = 0 \end{cases}$$

Using these variables, we define the next valid formulation for the $P|\,STsd|\,TCT$ problem.

Model 2

$$\boldsymbol{min\ z = \sum_{i=0}^{n} \sum_{j=1, j\neq i}^{n} t_{ij} f_{ij}} \qquad (\mathbf{3.12})$$

Subject to:

$$\boldsymbol{\sum_{j=1}^{n} x_{0j} = m} \qquad (\mathbf{3.13})$$

$$\sum_{j=1}^{n} x_{i0} = m \qquad (3.14)$$

$$\sum_{i=0, i\neq j}^{n} x_{ij} = 1 \qquad (j = 1, \dots, n) \qquad (3.15)$$

$$\sum_{j=0, j\neq i}^{n} x_{ij} = 1 \qquad (i = 1, \dots, n) \qquad (3.16)$$

$$f_{0j} \leq (n - m + 1)x_{0j} \qquad (j = 1, 2, .., n) \qquad (3.17)$$

$$f_{ij} \leq (n - m)x_{ij} \qquad (i, j = 1, 2, .., n; j \neq i) \qquad (3.18)$$

$$\sum_{j=1}^{n} f_{0j} = n \qquad (3.19)$$

$$f_{0j} + \sum_{i=1, i\neq j}^{n} (f_{ij} - f_{ji}) = 1 \qquad (j = 1, \dots, n) \qquad (3.20)$$

$$x_{ij} \geq f_{ij} \qquad (i, j = 1, 2, .., n; j \neq i) \qquad (3.21)$$

$$x_{ij} \in \{0, 1\} \ (i, j = 1, 2, .., n; j \neq i); \ f_{ij} \geq 0 \ (i = 0, 1, .., n; j = 1, 2, .., n; j \neq i) \qquad (3.22)$$

Constraints (3.13)-(3.16) are the same that constraints (3.3)-(3.6) in the first formulation. Constraints (3.17) and (3.18) force f_{ij} to be equal to zero when $x_{ij} = 0$ and they provide an upper bound for these variables when $x_{ij} = 1$. Here, $n - m + 1$ is the maximum number of jobs that may be processed by a machine, excluding the dummy jobs. Constraint (3.19) ensures that all the jobs are processed. Constraints (3.17), (3.18) and (3.20) are the sub-tours elimination constraints and allow calculating the positions of jobs on the sequences. Constraints (3.21) are valid inequalities proposed by Godinho, Gouveia, and Magnanti (2008) for the $mTSP$. Finally, constraints (3.22) define the nature of the variables. Note that although the variables f_{ij} are defined as integer variables, they can be considered as real variables in this formulation because the sense of the objective function, the binary nature of the variables x_{ij} and constraints (3.17) and (3.20) force them to be integer variables. In addition, these facts enable the variables f_{ij} to measure the number of jobs in a sequence after job i. Therefore, in the expression (3.12) used to evaluate the TCT, the variable f_{ij} play the same role as the coefficients $(k - j + 1)$ in the expression (3.10).

3.3.3 Assignation and flow-based formulation for TCT

The third formulation for $P|\,STsd|\,TCT$ is an adapted version of the best formulation reported in the literature for the $R|\,STsd|\,Cmax$ problem (Avalos-Rosales, Angel-Bello, and Alvarez, 2015). This formulation uses three-index arc binary variables x_{ijk} and the two-index assignment binary variables y_{ik}.

$$x_{ijk} = \begin{cases} 1, & \text{if the job } j \text{ is processed just after job } i \text{ in machine } k \\ 0, & \text{otherwise} \end{cases}$$

$$y_{ik} = \begin{cases} 1, & \text{if the job } j \text{ is processed in machine } k \\ 0, & \text{otherwise} \end{cases}$$

C_i: is the completion time of job i

To obtain a valid formulation for our problem, we do not consider the constraints related to C_{max} from the original formulation. Using the defined variable, the third model for $P|\ STsd|\ TCT$ can be formulated as follows:

Model 3

$$\boldsymbol{min\ z = \sum_{i=1}^{n} C_i} \qquad (\mathbf{3.23})$$

Subject to:

$$\sum_{k=1}^{m} \boldsymbol{y_{ik} = 1} \qquad (\boldsymbol{i = 1, 2, \dots, n}) \qquad (\mathbf{3.24})$$

$$\sum_{j=0, j\neq i}^{n} \boldsymbol{x_{ijk} = y_{ik}} \qquad (\boldsymbol{i = 1, 2, \dots, n; k = 1, 2, \dots, m}) \qquad (\mathbf{3.25})$$

$$\sum_{i=0, i\neq j}^{n} \boldsymbol{x_{ijk} = y_{jk}} \qquad (\boldsymbol{j = 1, 2, \dots, n; k = 1, 2, \dots, m}) \qquad (\mathbf{3.26})$$

$$\sum_{j=1}^{n} \boldsymbol{x_{0jk} \leq 1} \qquad (\boldsymbol{k = 1, 2, \dots, m}) \qquad (\mathbf{3.27})$$

$$\boldsymbol{C_j - C_i + T(1 - x_{ijk}) \geq x_{ijk} + y_{jk}}$$
$$(\boldsymbol{i = 0, 1, \dots, n;\ j = 1, 2, \dots, n; j \neq i; k = 1, 2, \dots, m}) \qquad (\mathbf{3.28})$$

$$\boldsymbol{C_0 = 0} \qquad (\mathbf{3.29})$$

$$\boldsymbol{x_{ijk} \in \{0, 1\}}, \quad \boldsymbol{y_{ik} \geq 0,\ C_i \geq 0} \quad (\boldsymbol{i, j = 0, 1, \dots, n;\ j \neq i; k = 1, 2, \dots, m}) \qquad (\mathbf{3.30})$$

Constraints (3.24) ensure that each job is assigned exactly to one machine. Constraints (3.25) guarantee that every job has exactly one successor in the assigned machine. Constraints (3.26) establish that every job has exactly one predecessor in the assigned machine. The predecessor and successor can be either the dummy job or any of the remaining jobs. Constraints (3.27) ensure that one job, at most, is scheduled as the first on each machine after the dummy job. Constraints (3.28) break sub-tours and provide a right processing order allowing the calculation of the completion times of the jobs. The restriction (3.29) sets the completion times of the dummy jobs to zero. Finally, constraints (3.30) define the nature of the variables. Note that in previous formulation the variables $\boldsymbol{y_{ik}}$ are stated as non-negative real variables even when they were defined as binary variables. This is because constraints (3.25) and (3.26) force them to be binary variables. Constraints (3.28) and (3.29) together allow to calculate the objective function which is to minimize the value of TCT.

3.4 Time dependent formulations

From expressions (3.10) and (3.11), it can be observed that the sequencing problem is a particular case of a time-dependent mTSP, given that the contribution of each job to the objective function depends on its position in the machine sequence. This property will

be used for providing two time-dependent mixed integer models for the underlying problem, which generalize formulations for the Minimum Latency Problem (MLP) discussed in (Angel-Bello, Alvarez, & García, 2013). The first formulation that we propose is adapted from the k-Travelling Repairmen Problem (k-TRP) formulation (Nucamendi-Guillén, Martínez-Salazar, Angel-Bello, and Moreno-Vega, 2016). The second one can also be seen as a generalization of the formulation proposed by Picard and Queyranne (1978) for the minimization of the TCT in a single machine scheduling problem with sequence dependent setup times.

3.4.1 Time dependent formulation based on assignation and flow

To get the following formulation let us define the following decision variables.

$$x_i^k = \begin{cases} 1, & \text{if there are } (k-1) \text{ jobs after job i in the sequence} \\ 0, & \text{otherwise} \end{cases}$$

$$y_{ij}^k = \begin{cases} 1, & \text{if } j \text{ is sequenced just after job } i \text{ in any machine,} \\ & \text{and there are } k \text{ remaininig jobs in the sequence} \\ 0, & \text{otherwise} \end{cases}$$

The parameter $N = n - m + 1$ is the maximum number of jobs that can be processed on a machine.

Using these variables, the first-time dependent model is presented below.

Model 4

$$\mathbf{min}\ z = \sum_{j=1}^{n} c_{0j} \sum_{k=1}^{N} k y_{0j}^k + \sum_{i=1}^{n} \sum_{j=1, j\neq 1}^{n} c_{ij} \sum_{k=1}^{N-1} k y_{ij}^k \qquad (3.31)$$

Subject to:

$$\sum_{k=1}^{N} x_i^k = 1 \qquad (\forall i \epsilon I) \qquad (3.32)$$

$$\sum_{i=1}^{n} x_i^1 = m \qquad (3.33)$$

$$\sum_{k=1}^{N} \sum_{j=1}^{n} y_{0j}^k = m \qquad (3.34)$$

$$\sum_{j=1, j\neq i}^{n} y_{ij}^k = x_i^{k+1} \qquad (\forall i \epsilon I; k = 1, 2, \dots, N-1) \qquad (3.35)$$

$$y_{0j}^k + \sum_{i=1, i\neq j}^{n} y_{ij}^k = x_j^k \qquad (\forall i \epsilon I; k = 1, 2, \dots, N-1) \qquad (3.36)$$

$$y_{0j}^N = x_j^N \qquad (\forall j \epsilon I) \qquad (3.37)$$

$$x_i^k \in \{0,1\} \qquad (\forall i \in I; k = 1,2,\dots,N) \qquad (3.38)$$

$$y_{0j}^k \geq 0 \qquad (\forall i \in I; k = 1,2,\dots,N)$$

$$y_{ij}^k \geq 0 \qquad (\forall i \in I; j \neq i;\ k = 1,2,\dots,N-1)$$

The objective function (3.31) minimizes the total completion time. When $y_{ij}^k = 1$ there are k nodes after node i in the sequence and for this reason the variables y_{ij}^k have been multiplied by k in (3.31), making the objective function to correspond with the formula (3.11) and thus, being able to evaluate the TCT of any feasible solution. Constraints (3.32) guarantee that each job is processed by a single machine. Constraint (3.33) ensures that every machine has assigned at least one job, while constraint (3.34) ensures that exactly m machines are used. These two constraints together ensure that there are an initial and a final job on each machine. Constraints (3.35) guarantee that every job has exactly one successor in the assigned machine. This successor can be any other job. Constraints (3.36) establish that every job has exactly one predecessor in the assigned machine. This predecessor can be the dummy job or any other job. Constraints (3.37) are used when there is a sequence with exactly N jobs, in other case they are redundant. Constrains (3.35), (3.36) and (3.37) are the connectivity constraints and guarantee the continuity of the sequences. Finally, constraints (3.38) establish the nature of the variables.

To the better understanding of the next time-dependent mathematical model, it is presented a multi-level network shown in Figure 3.1

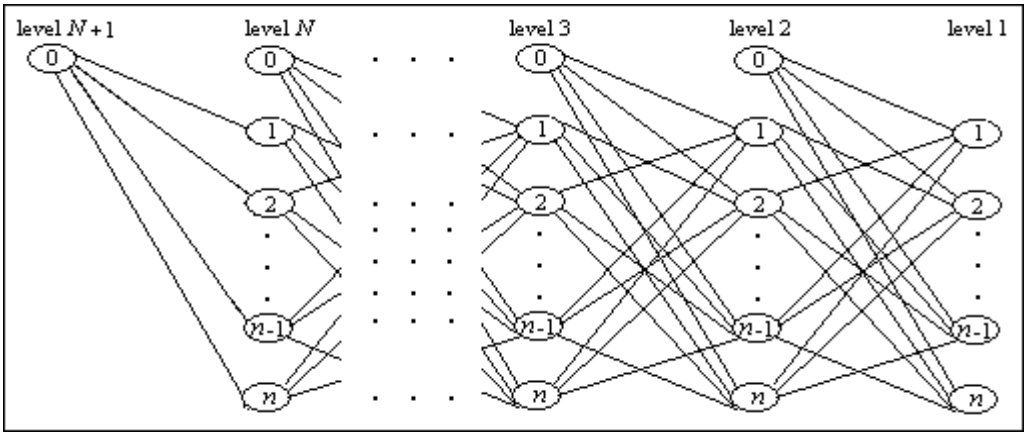

Figure 3.1 The multilevel network

In Figure 3.1, each node represents a job and each level represents the position of the job i in the sequence. Then, a sequence is represented by a path that links the dummy job in any level to the jobs in the lower levels until level 1. The nodes of a same level cannot be connected due to two or more jobs cannot be processed at the same time on a same machine and as well there is no arcs between same nodes in different levels due to a job cannot be processed two or more times.

Level 1 consists of a copy of nodes associated with jobs, levels $2, 3, \dots, N$ are composed of a copy of nodes associated with every job plus a dummy job representing the machines' initial states, while the $N + 1$ level contains a copy of node 0 associated with the initial state of machines. Each sequence is represented in this network by a path that

starts at node 0 (associated with the initial states of machines), visits nodes in lower levels and ends at a node in level 1. It is said that a node is active at a certain level when it is visited by a path at that level.

The contribution of each arc (i,j) used in the solution to the TCT depends on the levels that this arc connects in the network. If arc (i,j) connects level $k+1$ with level k, then its contribution to the objective function is kt_{ij}.

Using the multilevel network of Figure 3.1, the underlaying problem consists on finding m disjoint paths on the multi-level network. Each path starts from a node 0 in any level from level 2 to $N+1$ and ending at nodes in level 1 in such a way that the sum of the lengths of the paths is minimized. For example, a path can start from node 0 at level 3, and connect it with the node 2 at level 2 and connect this one with the node 1 at level 1 to finish the sequence. This path follows the order 0-2-1, and contains 2 jobs. The arc (0,2) connects level 3 with level 2 and this implies that there are 2 more nodes in the path, after node 0.

3.4.2 Time dependent formulation based on flow

The next model can be obtained from the previous one by expressing all the constraints in terms of the previously defined binary variables y_{ij}^k or also from the multi-level network (shown in Figure 3.1) redefining the binary variables y_{ij}^k for this network.

$$y_{ij}^k = \begin{cases} 1, & \text{if the arc } (i,j) \text{ is used to link the node i in level } k+1 \\ & \text{with node } j \text{ in level } k \\ 0, & \text{otherwise} \end{cases}$$

The second time-dependent model is shown below.

Model 5

$$\mathbf{min}\ z = \sum_{j=1}^{n} c_{0j} \sum_{k=1}^{N} k y_{0j}^k + \sum_{i=1}^{n} \sum_{j=1, j\neq 1}^{n} c_{ij} \sum_{k=1}^{N-1} k y_{ij}^k \qquad (3.39)$$

Subject to:

$$\sum_{k=1}^{N} y_{0j}^k + \sum_{k=1}^{N-1} \sum_{i=1, i\neq j}^{n} y_{ij}^k = 1 \qquad (\forall j \epsilon\, I) \qquad (3.40)$$

$$\sum_{j=1}^{N} y_{0j}^1 + \sum_{i=1}^{n} \sum_{j=1, j\neq i}^{n} y_{ij}^1 = m \qquad (\forall i \epsilon\, I) \qquad (3.41)$$

$$\sum_{k=1}^{N} \sum_{j=1}^{n} y_{0j}^k = m \qquad (3.42)$$

$$y_{0j}^{k+1} + \sum_{i=1, i\neq j}^{n} y_{ij}^{k+1} = \sum_{i=1, i\neq j}^{n} y_{ji}^k \qquad (\forall i \epsilon I; k = 1, 2, \dots, N-2) \qquad (3.43)$$

$$y_{0j}^N = \sum_{i=1, i\neq j}^{n} y_{ji}^{N-1} \qquad (\forall j \epsilon I) \qquad (3.44)$$

$$y_{0j}^{k} \in \{0,1\} \qquad (\forall i \in I; k = 1,2,\dots,N) \qquad (3.45)$$
$$y_{ij}^{k} \in \{0,1\} \qquad (\forall i \in I; j \neq i;\ k = 1,2,\dots,N-1)$$

The objective function (3.39) uses the same expression to evaluate the TCT that the previous model. Constraints (3.40) are similar to (3.32) and they ensure that each job is processed by a single machine. Constraints (3.41) and (3.42) are equivalent to constraints (3.33) and (3.34) and together they ensure that there are an initial and a final job on each machine. Constraints (3.43) are the flow conservation constraints and they guarantee the continuity of the paths. Constraints (3.44) are used when there are sequences that contain exactly N jobs, otherwise they are redundant. Finally, constraints (3.45) establish the nature of the variables.

From now on we will refer to the previous formulations to as Model1, Model2, Model3, Model4 and Model5 according to the order in which they were presented. A summary with the number of binary variables, real variables and constraints contained in each formulation is shown in table 3.1

Table 3.1 Summary of variables and constraints

	Binary variables	Real variables	Constraints
Model1	$n^2 + n$	n	$n^2 + 2n + 2$
Model2	$n^2 + n$	n^2	$n^2 + 2n + 3$
Model3	$(n^2 + n)m$	$nm + 1$	$n^2 + 2nm + n + m + 1$
Model4	$n(n - m + 1)$	$n^2(n - m + 1)$	$2n(n - m + 1) + 2$
Model5	$n^2(n - m) + n$	-	$n(n - m + 1) + 2$

3.5 Computational experiments and Comparisons

In order to analyze the effectiveness of the proposed formulations we conducted two types of experiments. First, we compare the results obtained by all the formulation for small instances. Second, for larger instances we compare results obtained by the two time-dependent formulations regarding the size of instances that they can solve optimality and the computational time to reach the optimal solution.

All experiments described in this section were performed on an Intel Core 2 Duo CPU at 3.00 GHz and 3.21 GB of RAM under Windows OS. The formulations are implemented in C++ using concert technology of professional solver CPLEX 12.6.

To conduct the experiments, we took two groups of instances from literature and generated one more group. The first group involves small instances proposed by Vallada and Ruiz (2011). In these instances, combinations of number of jobs $n = \{6,8,10,12\}$ and number of machines $m = \{2,3,4,5\}$ were considered. The setup times s_{ij} and the processing times p_j were generated using the uniform distribution. For setup times there are three intervals: S_1: [1-49], S_2: [1-99] and S_3: [1-124] and for processing times only one interval which is [1-99]. There are 10 replicates for each possible combination of number of machines, number of jobs and range of setup times, obtaining a total of 480 small instances.

The second group was generated by Avalos Rosales (2014) using the same instance configuration described in Vallada and Ruiz (2011) with $n = \{15,20,25,30\}$ and $m = \{2,4,6,8\}$ and the same intervals to generate setup and processing times. Finally, we complement the set of instances by generating a third group with $n = \{40, 50,60\}$ and the same parameters for setup and processing times used in Avalos Rosales (2014). There are 5 replicates for each possible combination of number of machines, number of jobs and range of setup times, obtaining a total of 420 medium instances.

Note that the instances took from literature were generated for unrelated parallel machines. Then, since we are working with identical machines, we have taken the data of the first machine in each instance.

3.5.1 Comparison of the presented formulations on small instances

The purpose of this first experiment is to evaluate the behavior of all the formulations regarding to the size of solved instances and the CPU time to reach the optimal solutions. To perform this assessment, we will use small instances grouped according to size (n, m) and the setup time range (S).

In table 3.21, columns 1 and 2 refer to the size of the instances in terms of number of jobs and number of machines, respectively, while column 3 indicates the setup time ranges. Entries in columns 4, 6, 7, 9 and 10 exhibit the CPU time (in seconds) elapsed by the solver using each model, respectively. Columns 5 and 8 display the number of optimal solutions found by model1 and model3, respectively. For the other models these values are not shown because the solver using these formulations was able to reach all the optimal solutions. Values of the CPU times in table 3.1 are averaged over 10 instances and the solver was allowed to run a maximum time of one hour (3600 sec.) for each instance.

Table 3.2 Comparison between the five proposed formulations

			Model1		*Model2*	*Model3*		*Model4*	*Model5*
n	*m*	*S*	*CPU time (s)*	*nbOpt*	*CPU time (s)*	*CPU time (s)*	*nbOpt*	*CPU time (s)*	*CPU time (s)*
6	2	*S1*	0.29	10	0.41	0.39	10	0.16	0.14
		S2	0.37	10	0.41	0.42	10	0.16	0.15
		S3	0.30	10	0.34	0.32	10	0.15	0.15
	3	*S1*	0.27	10	0.33	0.36	10	0.15	0.15
		S2	0.32	10	0.28	0.45	10	0.15	0.15
		S3	0.29	10	0.30	0.39	10	0.15	0.15
	4	*S1*	0.25	10	0.28	0.44	10	0.15	0.14
		S2	0.27	10	0.27	0.37	10	0.13	0.14
		S3	0.25	10	0.26	0.34	10	0.14	0.15
	5	*S1*	0.21	10	0.23	0.27	10	0.14	0.14
		S2	0.21	10	0.22	0.26	10	0.14	0.15
		S3	0.19	10	0.22	0.26	10	0.13	0.15
8	2	*S1*	3.48	10	0.82	8.82	10	0.20	0.20
		S2	2.07	10	0.62	5.81	10	0.18	0.18
		S3	1.53	10	0.60	3.76	10	0.19	0.17

	3	*S1*	1.12	10	0.56	3.37	10	0.20	0.17
		S2	1.09	10	0.42	2.67	10	0.19	0.17
		S3	0.69	10	0.51	2.17	10	0.19	0.16
	4	*S1*	0.45	10	0.49	1.55	10	0.17	0.15
		S2	0.48	10	0.31	0.70	10	0.17	0.15
		S3	0.41	10	0.35	0.63	10	0.17	0.15
	5	*S1*	0.39	10	0.32	1.22	10	0.16	0.16
		S2	0.31	10	0.30	1.11	10	0.16	0.16
		S3	0.33	10	0.28	1.02	10	0.16	0.15
10	*2*	*S1*	133.31	10	3.63	435.99	10	0.21	0.17
		S2	48.84	10	1.42	178.96	10	0.21	0.16
		S3	28.05	10	1.18	110.53	10	0.22	0.17
	3	*S1*	12.21	10	1.85	79.60	10	0.22	0.18
		S2	10.52	10	1.16	63.38	10	0.23	0.23
		S3	2.72	10	0.66	10.81	10	0.21	0.17
	4	*S1*	3.61	10	0.78	7.55	10	0.19	0.16
		S2	2.44	10	0.69	4.85	10	0.18	0.16
		S3	1.22	10	0.55	3.38	10	0.20	0.18
	5	*S1*	1.78	10	0.60	3.59	10	0.20	0.16
		S2	0.97	10	0.51	2.55	10	0.20	0.16
		S3	0.64	10	0.39	2.05	10	0.20	0.16
12	*2*	*S1*	3564.90	1	196.75	-	0	0.31	0.27
		S2	1240.83	2	17.97	4268.82	1	0.26	0.22
		S3	1232.67	9	7.05	2896.41	3	0.30	0.25
	3	*S1*	1216.24	7	10.56	2005.77	2	0.23	0.22
		S2	809.55	9	6.08	858.97	4	0.27	0.24
		S3	36.94	10	1.70	881.58	10	0.27	0.25
	4	*S1*	280.83	10	4.00	1295.10	10	0.22	0.19
		S2	28.12	10	1.92	199.54	10	0.23	0.21
		S3	3.26	10	1.14	45.80	10	0.23	0.22
	5	*S1*	9.76	10	1.68	40.72	10	0.24	0.22
		S2	1.61	10	0.81	12.58	10	0.22	0.19
		S3	1.03	10	0.65	9.92	10	0.22	0.17

From table 3.2 it can be observed that Model2, Model4 and Model5 obtained optimal solutions for all instances up to several orders of magnitude faster than Model1 and Model3 and that these last two models begin to fail in instances of 12 jobs. Note that model2 was able to obtain all the optimal solutions but spending significantly more computational time than model4 and model5. Another fact that is observed is that as the number of machines and the range of setup times decrease, instances are more difficult to solve for Model1, Model2 and Model3 while this fact does not seem to affect Model4 and Model5. The latter is expected because when there are fewer machines the job sequences are longer. In summary, we can conclude that the winners in this experiment are the time-dependent based formulation model4 and model5.

3.5.2 Comparison between time-dependent formulations.

Taking into account that both time-dependent formulations reached optimal solutions for all instances and that they spent significantly less CPU time than model2, in the second experiment we compare these two formulations regarding the CPU time needed to reach the optimal solutions.

To carry out the next experiment we group the instances according to the values of n and m. We do not show the results according the different setup time ranges we did not find significant differences in the computational time for the different ranges.

In Table 3.3 only CPU times are shown because both formulations reached all the optimal solutions. Columns 1 and 2 refer to the size of instances. Columns from 3 to 6 show the CPU times (in seconds) spent by the solver to achieve optimal solution. Columns 3 and 4 are associated with Model4 while Columns 5 and 6 with Model5. Values in columns 3 and 5 are related to the instance that consumed more computational time to reach the optimum solution while values in columns 4 and 6 are to the instance that was solved faster within the 15-instance group.

Table 3.3 Comparison between time-dependent formulations

		Model4		*Model5*	
n	*m*	*Max CPU time(s)*	*Min CPU time(s)*	*Max CPU time(s)*	*Min CPU time(s)*
15	2	0.989	0.218	0.965	0.252
	4	0.533	0.220	0.530	0.207
	6	0.432	0.180	0.382	0.197
	8	0.212	0.156	0.264	0.193
20	2	1.985	0.407	2.723	0.489
	4	0.780	0.304	1.090	0.399
	6	0.554	0.261	0.746	0.368
	8	0.363	0.210	0.642	0.317
25	2	6.862	0.877	5.571	1.135
	4	2.500	0.516	4.688	0.784
	6	1.120	0.420	1.603	0.724
	8	0.802	0.339	1.387	0.631
30	2	161.905	1.572	22.267	2.045
	4	4.575	1.106	6.854	1.616
	6	2.197	0.748	3.777	1.365
	8	1.825	0.587	2.270	1.198
40	2	1036.240	67.582	147.227	22.032
	4	295.470	4.401	35.676	6.105
	6	9.218	2.578	11.422	3.265
	8	5.188	1.359	10.563	2.390
50	2	12579.500	359.672	897.584	32.938
	4	1886.590	31.159	212.862	29.715
	6	334.063	11.109	27.500	14.078
	8	22.406	10.281	39.235	12.890
60	2	49411.9*	8048.04	7212.420	846.912
	4	6827.35	481.74	766.593	370.487
	6	2829.81	79.615	355.856	84.222
	8	1031.44	45.229	127.082	49.571

From this table it can be seen that formulations have a similar performance up to 30 jobs. From 40-job instances Model5 begins to have better performance than Model4, with a significant difference for the hardest instances, i.e. for instances with fewer machines. The symbol "*" means that the solver using model 4 could not solve 4 of the

15 instances with 60 jobs and 2 machines. For unresolved instances the maximum gap was 0.7552%.

3.6 Chapter conclusions

In this chapter we studied a scheduling problem for minimizing total completion time in identical parallel machines with sequence dependent setup times. We derived three mixed integer formulation from known formulation for the $mTSP$ and for $R|STsd|Cmax$ problem and presented two time-dependent formulations for the addressed problem.

In computational experiments we showed that time-dependent based formulations performed much better than the others formulations in terms of computational time and quality of the results. To the best of authors' knowledge, the time-dependent formulations for this problem have not been presented previously in the operations research literature.

Both time-dependent based formulations, implemented in a commercial solver, could solve instances up to 60 jobs in a reasonable computation time. Particularly, the model5 showed better performance when there are many jobs and few machines.

As future researches, on the one hand, it would be interesting to investigate the scope of the models, that is, to investigate how large may be the instances that they can optimally solve, and on the other hand, in order to accelerate the solution process could be considered to obtain valid inequalities and to develop heuristic methods to provide good initial solutions to formulations.

Chapter 4 A hybrid metaheuristic algorithm for the $P|STsd|TCT$ problem

4.1 Introduction

In parallel machine scheduling literature, we found that many authors have studied problems those are close to the one that we address in this research. For instance, problems with same characteristics as the minimization of the total weighted completion time or other related objective functions with extra features or the minimization of the TCT with additional constraints like job release dates, precedence constraints or machine eligibility constraints.

However, for the specific problem of minimizing the TCT in an identical parallel machine scheduling problem with sequence dependent setup, we only found two works reported in the literature. Baez, Angel-Bello and Alvarez (2016) proposed two time-dependent mathematical formulations that solved to optimality instances up to 60 jobs. They compared the time-dependent formulations with a modified formulation based on the classical m-Travelling Salesman Problem, showing that the time-dependent formulaions have a better performance than the modified formulation, consuming less computational time and solving to optimality more than five times larger instances. Morales, Acosta and Socarrás (2016) developed a first metaheuristic algorithm for the addressed problem. They implemented an Iterated Greedy Algorithm that used as diversification generator a destructive-constructive process and as improvement phase a composited local search procedure based on intra-machine and inter-machine moves by means of relocation and interchange of jobs.

Taking into account that the mathematical models are developed in Chapter 3 can solve optimality limited-size instances. In this work we propose a hybrid metaheuristic algorithm (HMA) which is composed of $GRASP$ and VNS as the improvement procedure. In addition, in the shaking phase of VNS, it uses the destructive-constructive strategy of the Iterated Greedy Algorithms. The proposed algorithm outperforms the results obtained by the current state of the art methodology.

4.2 The proposed metaheuristic solution approach

In this section we describe the proposed metaheuristic algorithm. It is a hybrid algorithm based on $GRASP$ that uses in the improvement phase a $GVNS$ with random selection of neighborhood and with a systematic application of an iterated greedy in shaking phase.

Each $GRASP$ iteration is composed of a constructive phase and an improvement phase. The best local optimum found over all $GRASP$ iterations is saved as the best found solution (Feo & Resende, 1995). In this chapter we consider a hybrid local search scheme, that is, the basic local search scheme has been extended in order to explore multiple neighborhoods. The rationale behind this is, since a global minimum is a local minimum with respect to all neighborhoods, it should be easier to find global minimum if more neighborhoods are explored.
A pseudocode for the proposed hybrid metaheuristic algorithm is showed in Algorithm 4.1.

```
// Hybrid Metaheuristic Algorithm
Data: Problem data, K_max, L_max, neighborhood structures for improvement
      N_l, (l = 1, 2, ..., L_max), and a stopping conditions
1  f(S^(*)) ← +∞
2  while (GRASP stopping condition is not met) do
3  |  Obtain an initial solution S^(0) through a GRASP construction
4  |  S^(best) ← RVND(S^(0), N_l, L_max): Apply the improvement procedure to
   |  S^(0)
5  |  Save (S^(best), f(S^(best)))
6  |  while (GVNS stopping condition is not met) do
7  |  |  k ← 1
8  |  |  while (k ≤ K_max) do
9  |  |  |  S ← Shaking(S^(best), k): Generate a solution S through a
   |  |  |  random disturbance of the solution S^(best)
10 |  |  |  S' ← RVND(S, N_l, L_max): Apply the improvement procedure
   |  |  |  to S
11 |  |  |  if (f(S^(best)) > f(S')) then S^(best) ← S', f(S^(best)) ← f(S'),
   |  |  |  k ← 1
12 |  |  |  else k ← k + 1
13 |  |  end
14 |  end
15 |  if (f(S^(*)) > f(S^(best))) then S^(*) ← S^(best), f(S^(*)) ← f(S^(best))
16 end
   Result: (S^(*), f(S^(*)))
```

Algorithm 4.1 Pseudo-code for the proposed Hybrid Metaheuristic Algorithm

The HMA receives inputs of the problem data, the maximum number of iterations (K_{max}) that a solution is perturbed without improvement, the neighborhood structures of the solution (N_l, $l = 1,2,\dots,K_{max}$) used in the Random Variable Neighborhood Descent (RVND) method and the stopping criteria (maximum number of $GRASP$ iterations and maximum number of $GVNS$ iterations). The objective function value is initialized as $+\infty$ in line 1 of the algorithm and the procedure goes into a loop in line 2 that finishes in line 16 reporting the best global found solution $S^{(*)}$ and its objective function value $f(S^{(*)})$.

An initial solution $S^{(0)}$ is generated using a GRASP in line 3 and it is improved in line 4 by a VND procedure with random selection of neighborhoods (RVND). The improved solution and its objective value are saved as the best solution $S^{(best)}$ and the best solution value $f(S^{(best)})$ of the current GRASP iteration. Then, procedure goes into the GVNS loop in line 6 receiving as input $S^{(best)}$ and $f(S^{(best)})$. In line 7 the value of k is initialized (the k value is associated with the percentage of destruction of the given solution) and the procedure goes into a loop in line 8. In line 9 the current solution is partially destroyed depending on the value of k and then it is reconstructed and then apply the improvement procedure (line 10). If the solution found is better than $S^{(best)}$ then $S^{(best)}$ and its objective function value are updated and the k value is reset to 1 (line 11). If not, the value of k is increased by a unit (line 11). Note that line 9-12 within

the loop represent the systematic application of a iterated greedy strategy. Finally, in line 15 the best global solution and its value are updated.

4.2.1 Constructive Phase

In the implementation Sa denotes the set of unassigned jobs and by $SP = \{SP_1, SP_2, \ldots, SP_m\}$ a partial solution under construction. SP_r is the partial sequence under construction associated to machine $r, (r = 1,2,\ldots,m)$. A partial sequence that has k occupied positions, refers to the positions 1 to k. The position 0 is always occupied by the dummy job 0. At the beginning of the constructive phase $SP_r = \{0\}, (r = 1,2,\ldots,m)$ and $Sa = \{1,2,\ldots,n\}$.

A pseudo-code for the constructive procedure implemented for the constructive phase is shown in Algorithm 4.2.

```
   // Hybrid Algorithm:  Constructive Procedure
   Data: Problem data, α
 1 Initialization: Sa = {1, 2, ··· , n}, SP_r ← {0}(r = 1, 2, ..., m), TCT ← 0
 2 Initialize the m sequences with jobs j ∈ Sa having the lowest values of t_0j
 3 Update Sa and TCT
 4 while (Sa ≠ ∅) do
 5 |   Choose the machine r with less span
 6 |   foreach i ∈ Sa do
 7 |   |   Determine the best insertion point best_q_i in SP_r and its insertion
   |   |    value Δ_i
 8 |   end
 9 |   Compute Δ_min = min{Δ_i/∀i ∈ Sa}
10 |   Compute Δ_max = max{Δ_i/∀i ∈ Sa}
11 |   RCL ← {j ∈ Sa : Δ_j ≤ Δ_min + α(Δ_max − Δ_min)}
12 |   Select l randomly from RCL
13 |   Insert l in position best_q_l of SP_r
14 |   Sa ← Sa\{l}
15 |   Update TCT
16 end
17 Sol ← SP
   Result: (Sol, TCT)
```

Algorithm 4.2 Pseudo-code for our implementation of the constructive phase.

The constructive phase receives the problem data and the value of $\boldsymbol{\alpha}$, then the set $\boldsymbol{Sa}$ of unassigned jobs, the partial sequences SP_r and TCT are initialized (line 1). The m jobs with lowest values of t_{0j} are selected and they are assigned to the machines, one per machine (line 2). Then, Sa and TCT are updated. In line 4 the procedure enters to a loop that finishes when all the jobs have been assigned to machines. First, the machine with the shortest span (the completion time of the last job assigned to the machine) is selected (line 5). Then, for each unassigned job, its best insertion point in the selected machine and the corresponding insertion value are calculated (lines 6-8). The best insertion point is the position in the machine sequence where the machine TCT is less increased and the insertion value is the variation in the machine TCT if the job would be inserted in that position. In the lines 9 to 11, the jobs that will be included in the restricted candidate list RCL are determined. A job is randomly selected from RCL

(line 12) and it is inserted in the selected machine in its best insertion point (line 13). TCT and Sa are updated.

The output of the procedure is a feasible solution Sol and its objective value TCT.

The greedy function, used in lines 6 to 8 for evaluating the candidate jobs to be inserted in the partial solution, is based on a process of searching for the best insertion point.

Let's analyze the insertion of a new job into a sequence under construction. It is easy to implement that local moves in any sequence affect the contribution of several jobs to the objective function. Suppose we have to insert a new job j into a position q of a sequence under construction with k jobs

$$\boldsymbol{P = \{0, [1], [2], \dots, [q-1], [q], [q+1], \dots, [k]\}}$$

On the one hand, if we use the Formula 3.1 of Chapter 3 for a sequence of k jobs $TCT(P) = \sum_{j=1}^{k} C[j] = C[1] + C[2] + \cdots + C[k]$, where $C[i] = \sum_{j=1}^{i} t[j-1][j] = C[i-1] + t[i-1][i]$, it is easy to see that the insertion of a new job affects the completion times of all jobs in sequence after the insertion point.

On the other hand, if we use the Formula 3.10 of Chapter 3 to evaluate the TCT of a sequence P ($TCT(P) = kt_{[0][1]} + (k-1)t_{[1][2]} + \cdots + 2t_{[k-2][k-1]} + t_{[k-1][k]}$), then the insertion of a new job affects the contributions of all jobs that are before the insertion point.

Since this process is very time-consuming, we propose an efficient insertion strategy to implement this approach using the formula 3.10 in Chapter 3. Given a partial sequence SP_r with k occupied positions and the set of unassigned jobs Sa, find the best insertion point in the sequence under construction for each job $i \in Sa$, we propose the following procedure.

First, the insertion values Δ_{iq} in positions $q, (q = 1,2,\dots,k+1)$ of SP_r are calculated starting from the first position ($q = 1$) using the next expression:

$$\Delta_{iq} = \begin{cases} \boldsymbol{(k+1)t_{o1} + k(t_{i[1]} - t_{01})}, & \boldsymbol{if\ q = 1} \\ \boldsymbol{\sum_{l=1}^{q-1} t_{[l-1][l]} + (k-q+2)t_{[q-1]i} + (k-q+1)\left(t_{i[q]} - t_{[q-1][q]}\right)}, & \boldsymbol{if\ 2 \le q \le k} \\ \boldsymbol{\sum_{l=1}^{k} t_{[l-1][l]} + t_{[k]i}} & \boldsymbol{if\ q = k+1} \end{cases} \quad (4.1)$$

Then the best insertion value Δ_i and the best insertion point $best_{q_i}$ in SP_r for job i are calculated as:

$$\Delta_i = \min_{1 \le q \le k+1} \{\Delta_{iq}\} \quad (4.2)$$

$$best_{q_i} = \arg\min_{1 \le q \le k+1} \{\Delta_{iq}\} \quad (4.3)$$

Note that the value of the sum $\sum_{l=1}^{q-1} t_{[l-1][l]}$ increases as the value of q increases, then using the formula (4.1) in an iterative way starting from $q = 1$, it is guaranteed to perform $O(1)$ elementary operations to evaluate each insertion point.

4.2 Improvement phase

Improvement phase is the second phase of the VNS, which is composed of lines 6 to 14 of the pseudocode shown in Algorithm 4.1. Next, we will then describe its main components referring to the lines of that Algorithm.

As shaking procedure (line 9) we use a destructive-constructive process where a partial destruction procedure and a reconstruction procedure are applied to a given solution. The percentage of the solution destruction is related to the number of the actual iteration $k, (k = \{1, \ldots, k_{max}\})$.

In Algorithm 4.3, a pseudo-code for the shaking procedure is shown.

```
   // Improvement phase:  Shaking(S_b, k)
   Data: S_b, k
1  (SP, Sa) ← PartialDestruction(S_b, k)
2  (S_0, f(S_0)) ← Reconstruction(SP, Sa)
   Result: (S_0, f(S_0))
```

Algorithm 4.3 Pseudo-code for shaking procedure.

The shaking procedure receives a solution and the value of k to determine the percentage of destruction. First, $\lfloor 10k\% \rfloor$ of jobs are randomly eliminated from the solution and the remaining jobs are arranged to the beginning of the sequences. Then, the solution is rebuilt using a method similar to the constructive procedure described above. The difference is that a totally greedy strategy is now used, that is, the job to be inserted is that with the lowest insertion value.

Algorithm 4.4 shows a pseudo-code for the proposed partial destruction procedure. This consists of removing a number (r) of jobs from a given solution. At the beginning, all the sequences are candidate to be chosen for the destruction procedure (i.e., they are active sequences). A sequence is deactivated when it no longer contains jobs. In line 2, the number of removed jobs is calculated. Note that the value of k must be less than 10. In the line 3, the procedure goes into a loop where jobs are removed from active machines one by one, until r jobs are removed. They are placed into the set of unassigned nodes. Then, the remaining jobs in the sequences are sequenced at the beginning and they are assigned to partial solution SP in the same order.

```
// Shaking:  PartialDestruction(S_b, k)
Data: S_b, k, n
Activate all the sequences in S_b
r ← ⌊0.1 * k * n⌋
while (r > 0) do
    Randomly select an active sequence in S_b
    Randomly choose a job in the selected sequence and place it in Sa
    if the selected sequence no longer contains jobs, put it inactive
    r ← r − 1
end
Pack the remaining jobs in each sequence of S_b to the beginning of the
  sequences and assign them to SP in the same order
Result: (SP, Sa)
```

Algorithm 4.4 Pseudo-code for partial destruction procedure.

The reconstruction procedure receives a partial solution SP and a set of unassigned jobs Sa. First, it is chosen the machine sequence SP_r with lowest value of TCT, then for each job in Sa it calculates the best insertion point in SP_r. The job less increasing the TCT of SP_r is deleted from Sa, it is inserted in its best insertion point in SP_r and the TCT of SP_r is updated. This process is repeated until all jobs in Sa have been assigned to the machines. The best insertion point is found using the formulas (4.1), (4.2) and (4.3).

In the implementation of GVNS for the improvement phase of the hybrid algorithm, it is used RVND (line 10 in Algorithm 4.1) instead of VND for improving a solution. It is also used it for improving the starting solution (line 4, Algorithm 4.1). We investigated the appropriate execution order of the neighborhoods and experimentally concluded that a stochastic order for exploring the neighborhoods yields better solutions than a deterministic order. For this reason, we decided to implement a RVND procedure inside GVNS.
Algorithm 4.5 shows a sketch of our implementation of RVND. In this procedure, each local search is repeats until it reaches a local minimum.

```
// Improvement Phase:  RVND Procedure
Data: Problem Data, current solution and set of neighborhoods
Activate all the neighborhoods
while (There are active neighborhoods) do
    Randomly select an active neighborhood
    Apply a local search to the current solution using the selected
      neighborhood
    if (The local search improved the current solution) then
        Update the current solution and its objective function
        Activate all the neighborhoods
    end
    Deactivate the selected neighborhood
end
Result: The current solution and its objective function
```

Algorithm 4.5 Sketch for our implementation of the RVND procedure.

In line 1, all neighborhoods are activated so that all candidates to be selected for the local search. In line 2 the procedure enters a loop that culminates when there are no longer active neighborhoods. An active neighborhood is selected (line 3) and a local search is applied to the current solution until a local minimum is reach (line 4). If found neighbor solution is better than the current solution, it is updated and all the neighborhoods are activated (lines 5-8). As the local search repeats until reaching a local minimum, it makes no sense to explore this neighborhood until the solution is modified again. For this reason, the neighborhood is deactivated in line 9 regardless of whether or not it has improved the current solution.

4.2.1 Local searches and strategies for their implementation

In the proposed RVND procedure we use four local searches based on intra-machine and inter-machine movements:

- Exchange move between two non-adjacent jobs in the same sequence: Two jobs i and j in non-adjacent positions i and j interchange their positions.

$$P = \{0, [1], [2], \dots, [i-1], [\boldsymbol{i}], [i+1], \dots, [j-1], [\boldsymbol{j}], [j+1], \dots, [k]\}$$

$$\downarrow$$

$$P = \{0, [1], [2], \dots, [i-1], [\boldsymbol{j}], [i+1], \dots, [j-1], [\boldsymbol{i}], [j+1], \dots, [k]\}$$

Fixing the position i, its sub-neighborhood is defined by all the jobs $[j]$ in positions $j \geq i+2$ that can interchange their positions with job $[i]$. Since in this case only jobs $[i]$ and $[j]$ change their positions, using the formula (3.10) it is possible to obtain the following expression 4.4 to evaluate the variation Δ_{ij} in the TCT of the given sequence.

$$\Delta_{\boldsymbol{i,j}} = (\boldsymbol{k-i+1})(\boldsymbol{t_{[i-1][j]} - t_{[i-1][i]}}) + (\boldsymbol{k-i})(\boldsymbol{t_{[j][i+1]} - t_{[i][i+1]}}) + (\boldsymbol{k-j+1})(\boldsymbol{t_{[j-1][i]} - t_{[j-1][j]}}) + (\boldsymbol{k-j})(\boldsymbol{t_{[i][j+1]} - t_{[j][j+1]}}) \quad (\boldsymbol{4.4})$$

- Relocation move of a job to a different position in the same sequence: The job $[i]$ is removed from its position i and inserted into another position j.

$$\boldsymbol{P} = \{\boldsymbol{0}, [\boldsymbol{1}], [\boldsymbol{2}], \dots, [\boldsymbol{i-1}], [\boldsymbol{i}], [\boldsymbol{i+1}], \dots, [\boldsymbol{j-1}], [\boldsymbol{j}], [\boldsymbol{j+1}], \dots, [\boldsymbol{k}]\}$$

$$\downarrow$$

$$\boldsymbol{P} = \{\boldsymbol{0}, [\boldsymbol{1}], [\boldsymbol{2}], \dots, [\boldsymbol{i-1}], [\boldsymbol{i+1}], \dots, [\boldsymbol{j-1}], [\boldsymbol{i}], [\boldsymbol{j}], [\boldsymbol{j+1}], \dots, [\boldsymbol{k}]\}$$

The sub-neighborhood of job $[i]$ is composed of all the positions j in which it can be inserted. In this case, formulas can also be obtained to evaluate the movement through an iterative process to obtain an efficient procedure to explore the sub-neighborhood.

An efficient way, used in this work, to explore the sub-neighborhood is through successive swaps between jobs in adjacent positions and calculate the move values in an incremental way. This idea was taken from Schiavinotto and Stützle (2004), who successfully implemented a similar strategy in the linear ordering problem context. The

formula (4.5) is used to evaluate the swap movements between jobs in adjacent positions.

$$\Delta_{i,i+1}= (\boldsymbol{k-i+1})\left(\boldsymbol{t_{[i-1][i+1]}-t_{[i-1][i]}}\right)+(\boldsymbol{k-i})\left(\boldsymbol{t_{[i+1][i]}-t_{[i][i+1]}}\right)+(\boldsymbol{k-i+1})\left(\boldsymbol{t_{[i][i+2]}-t_{[i+1][i+2]}}\right) \quad (\boldsymbol{4.5})$$

- Exchange move between two jobs in different sequences: Two jobs $[i]$ and $[j]$ in positions i and j of different sequences interchange their positions.

$$P_1 = \{0, [1_1], [2_1], \dots, [(i-1)_1], [\boldsymbol{i_1}], [(i+1)_1], \dots, [k_1]\}$$
$$P_2 = \{0, [1_2], [2_2], \dots, [(j-1)_2], [\boldsymbol{j_2}], [(j+1)_2], \dots, [k_2]\}$$

$$P_1 = \{0, [1_1], [2_1], \dots, [(i-1)_1], [\boldsymbol{j_2}], [(i+1)_1], \dots, [k_1]\}$$
$$P_2 = \{0, [1_2], [2_2], \dots, [(j-1)_2], [\boldsymbol{i_1}], [\boldsymbol{(j+1)_2}], \dots, [\boldsymbol{k_2}]\}$$

Fixing the position i in the first sequence, the sub-neighborhood of job $[i]$ is defined by all the jobs $[j]$ in the second sequence that can interchange their positions with job $[i]$.

Suppose that the first machine contains k_1 jobs and the second machine contains k_2 jobs. Then, as only jobs $[i]$ and $[j]$ change their positions, is obtained a simple expression (4.6) to evaluate the variation Δ_{ij} in the TCT of the solution.

$$\Delta_{i,j}= (\boldsymbol{k_1-i+1})\left(\boldsymbol{t_{[i-1][j]}-t_{[i-1][i]}}\right)+(\boldsymbol{k_1-i})\left(\boldsymbol{t_{[j][i+1]}-t_{[i][i+1]}}\right)+(\boldsymbol{k_2-j-1})\left(\boldsymbol{t_{[j-1][i]}-t_{[j-1][j]}}\right)+(\boldsymbol{k_2-j})\left(\boldsymbol{t_{[i][j+1]}-t_{[j][i+1]}}\right) \quad (\boldsymbol{4.6})$$

- Relocation move of a job to a different sequence: The job $[i]$ is removed from its position i in a sequence and inserted into position j of another sequence.

$$P_1 = \{0, [1_1], [2_1], \dots, [(i-1)_1], [\boldsymbol{i_1}], [(i+1)_1], \dots, [k_1]\}$$
$$P_2 = \{0, [1_2], [2_2], \dots, [(j-1)_2], [j_2], [(j+1)_2], \dots, [k_2]\}$$

$$P_1 = \{0, [1_1], [2_1], \dots, [(i-1)_1], [(i+1)_1], \dots, [k_1]\}$$
$$P_2 = \{0, [1_2], [2_2], \dots, [(j-1)_2], [\boldsymbol{i_1}], [j_2], [(j+1)_2], \dots, [k_2]\}$$

When the job $[i]$ is extracted from the first sequence, the variation Δ_i in TCT is calculated using the expression (4.7). The positions i in the sequence are chosen in a sequential way starting by $i = 1$ in order to guarantee $O(1)$ elementary operations in this evaluation.

$$\Delta_i= \begin{cases} (\boldsymbol{k_1-1})\left(\boldsymbol{t_{0[i+1]}-t_{[i][i+1]}}\right)-\boldsymbol{k_1 t_{0[i]}} & \boldsymbol{if\ i=1} \\ (\boldsymbol{k_1-i})\left(\boldsymbol{t_{[i-1][i+1]}-t_{[i][i+1]}}\right)-(\boldsymbol{k_1-i+1})\boldsymbol{t_{[i-1][i]}}-\sum_{l=1}^{i-1}\boldsymbol{t_{[l-1][l],}} & \boldsymbol{if\ 2\le i\le k_1-1} \\ -\sum_{l=1}^{i}\boldsymbol{t_{[l-1][l],}} & \boldsymbol{if\ i=k_1} \end{cases} \quad (\boldsymbol{4.7})$$

Fixing the position i in the first sequence, the sub-neighborhood of job $[i]$ is defined by all the positions j in the second sequence where it can be inserted. The positions j in the second sequence are explored in a sequential way starting by the first position and variation Δ_j in the TCT of the second sequence is calculated by expression (4.1).

The value of a move, that is, the variation Δ_{ij} in the solution TCT when the job $[i]$ is removed from a sequence and it is inserted in position j of another sequence is calculated as $\Delta_{ij} = \Delta_i + \Delta_j$.

In addition, the implementation of the RVND algorithm includes a memory structure with the purpose of reducing the computational time. The reasoning behind this is to avoid re-exploring sequences that did not reach any improvement during the last execution of the considered neighborhood structure.

4.3 Computational experiments and Comparisons

The goals of the computational addressed conducted in this thesis work are the following:

- Corroborating the effectiveness of providing to a VNS algorithm several different initial solutions obtained through a constructive GRASP.
- Comparing the results yielded by the proposed hybrid metaheuristic algorithm with the optimal solutions found for small and medium-sized instances.
- Assessing the performance of the hybrid algorithm, comparing its results with those published in the literature.

The experiments were performed on an Intel® Core(TM) i5-2410M CPU @ 2.30GHz and 4.00GB of RAM processor. The proposed algorithm was coded in C++ and, to obtain the optimal solutions, the mathematical model was solved using the Concert Technology of Professional Solver Cplex 12.6.

For the computational experiments we used two sets of instances. The first set contains the instances generated by Avalos-Rosales et al. (2015) the procedure described in Vallada and Ruiz (2011) with $n = \{20, 30, 40, 50, 60\}$ and $m = \{2, 4, 6, 8\}$. The processing times (p_j) were uniformly generated between [1-99], and the setup times were uniformly generated in the ranges [1-49], [1-99] and [1-124], denoted by S1, S2 and S3, respectively. In a similar way, we generated the second group of instances with 70 and 80 jobs. For each combination of number of machines, number of jobs and range of setup times, there are 5 replicates for each set of instances.

Taking into account that the instances taken from the literature are for unrelated parallel machines and in this work, we are considering identical machines, we have taken the data of the first machine in each instance.

The stopping criterion in GRASP (number of iterations) and the parameter α to restrict the candidate list in the constructive procedure were statistically set to $n^2$and 0.3, respectively. The hybrid algorithm was run one time.

4.3.1 Benefits of hybridizing GRASP with VNS

This section is devoted to corroborate the good performance of the combination of GRASP and Variable Neighborhood Search (VNS) to solve the problem tackled in this work. That is, the results obtained using VNS starting from an initial solution constructed in a greedy way were compared with the results obtained with the proposed hybrid algorithm. Table 4.1 shows this comparison. Column 1 indicate the number of machines, while column 2 indicate the number of jobs. Column 3 display the objective values obtained by the VNS, while column 4 display the objective values obtained by the hybrid algorithm. In column 5 the improvement percentage ($Improv\%$) of the proposed algorithm (HMA) in relation to VNS is shown. It is calculated as

$$\boldsymbol{Improv\%} = \mathbf{100} * \frac{\boldsymbol{VNS_{val}} - \boldsymbol{HMA_{val}}}{\boldsymbol{VNS_{val}}} \qquad (\mathbf{4.8})$$

Where, VNS_{val} is the best solution value reported by the VNS algorithm, and HMA_{val} is the best solution value obtained by the proposed Hybrid Metaheuristic Algorithm.

The experimental results of VNS and HMA are summarized in Table 4.1, in which, the first column (m) represents the number of machines and the second column (n) stands for the number of jobs. The objective values obtained with both metaheuristics are presented in next two columns and the percentage improvement is represented in las column using the formula (4.8). The values displayed have been averaged over 15 instances with the same number of jobs and the same number of machines.

***Table 4. 1** Comparison between HMA and VNS*

		Objective values		*Improvement*
m	*n*	*VNS*	*HMA*	*(%)*
2	20	5205.87	5105.13	1.82
	30	11783.53	11347.13	3.73
	40	18386.67	17687.6	3.73
	50	29967.33	28838.8	3.74
	60	41437.07	39881.27	3.74
	70	53206.4	51210.53	3.71
	80	70183.93	67409.2	3.89
4	20	3439.93	3406.87	0.97
	30	6201.53	6029.2	2.73
	40	9990.87	9633.6	3.51
	50	16103.4	15453.47	3.95
	60	21913.13	21065.8	3.84
	70	27964	26935.4	3.66
	80	36566.4	35267.67	3.5
6	20	2548.27	2522.27	0.99
	30	4707.38	4560.44	2.91
	40	7237.73	7001.6	3.23
	50	11402.8	11016.93	3.36
	60	15471.8	14890.93	3.68

	70	19637.67	18914.2	3.65
	80	2509.07	24560.73	3.25
8	20	2058.93	2040.8	0.9
	30	3865.2	3783.87	2.09
	40	5916.6	5729.8	3.08
	50	9119.07	8857.27	2.75
	60	12248.33	11884.4	2.91
	70	15455.73	14918	3.44
	80	19969.73	19253.67	3.51

In order to assess the statistical differences between the hybrid algorithm, HMA, proposed here and VNS, the Wilcoxon non-parametric test for comparing two samples has been used. As the computed p-value is lower than the significance level $\alpha = 0.05$, the null hypothesis is rejected. Therefore, we conclude that the two algorithms are statistically different and the proposed hybrid metaheuristic outperforms VNS for the problem at hand.

4.3.2 Comparing with optimal solutions

Tables 4.2 and 4.3 shows, for the first set of instances, the comparison between the results obtained by the hybrid algorithm and the optimal solutions found using the mathematical model described in Chapter 3. Column 1 indicate the number of machines and column 2 the number of jobs (n) while column 3 indicate the setup time range (S). Column 4 display the CPU time elapsed by the solver (Model) and by the proposed hybrid metaheuristic algorithm (HMA), respectively. Columns 5 and 6 shows the relative gap of the results obtained by the proposed heuristic algorithm with respect to the optimal solutions. Results in columns 4 to 7 are averaged over 5 instances. For each instance, the relative gap is computed as

$$\boldsymbol{gap\%} = \mathbf{100} * \frac{\boldsymbol{Optimal_{val} - HMA_{val}}}{\boldsymbol{Optimal_{val}}} \qquad (\mathbf{4.9})$$

where $Optimal_{val}$ is the optimal solution value reported by the solver, and HMA_{val} is the best solution value obtained by the proposed algorithm.

***Table 4. 2** Comparison of HMA with optimal solutions for 2 and 4 machines.*

			Exact	*GRASP-VNS*	
m	*n*	*S*	*CPU time avg. (s)*	*CPU time avg. (s)*	*Gap(%) Avg*
2	20	*S1*	1.2452	0.1474	0.0000
		S2	1.558	0.1506	0.0000
		S3	0.9966	0.1376	0.0000
	30	*S1*	4.1556	0.7428	0.0417
		S2	5.7202	0.7344	0.4020
		S3	9.7134	0.7512	0.2985
	40	*S1*	38.7090	2.3724	0.1826

		S2	79.0626	2.2996	1.0319
		S3	65.9272	2.3206	1.3818
	50	*S1*	264.6488	5.8488	0.6781
		S2	495.0716	5.4884	1.6836
		S3	414.6172	5.7984	2.2219
	60	*S1*	1990.3574	12.2584	1.3669
		S2	1735.8160	12.4860	2.7307
		S3	3095.3104	12.2986	3.1924
4	20	*S1*	0.7966	0.1592	0.0000
		S2	0.7312	0.164	0.0000
		S3	0.6066	0.1568	0.0000
	30	*S1*	2.8056	0.782	0.0233
		S2	4.9328	0.7954	0.0217
		S3	4.4154	0.8154	0.4767
	40	*S1*	10.9486	2.4908	0.4909
		S2	15.5674	2.4484	1.1894
		S3	14.8664	2.5692	0.9717
	50	*S1*	59.8036	6.2078	1.0346
		S2	106.7918	5.9144	1.8820
		S3	150.3252	6.2	2.3391
	60	*S1*	435.2174	12.7238	1.4991
		S2	268.6436	12.8032	2.7891
		S3	520.8954	12.363	2.9586

***Table 4. 3** Comparison of HMA with optimal solutions for 6 and 8 machines*

			Exact	*GRASP-VNS*	
m	*n*	*S*	*CPU time avg. (s)*	*CPU time avg (s)*	*Gap(%) Avg*
6	20	*S1*	0.4834	0.1714	0.0000
		S2	0.4930	0.1632	0.0000
		S3	0.4182	0.1658	0.0000
	30	*S1*	1.9396	0.7886	0.1150
		S2	2.4296	0.8038	0.0368
		S3	2.1300	0.8042	0.3345
	40	*S1*	7.0128	2.4804	0.5836
		S2	7.2686	2.4228	0.8618
		S3	6.4342	2.5294	1.4935
	50	*S1*	16.8280	6.1636	0.9250
		S2	18.4654	6.0264	1.9028
		S3	21.9032	6.1806	2.0629
	60	*S1*	128.7410	12.8940	1.4603
		S2	177.5666	12.4320	2.5590
		S3	224.3990	12.7454	2.8931
8	20	*S1*	0.4156	0.1888	0.0000

	S2	0.3546	0.1850	0.0000
	S3	0.3536	0.1898	0.0000
30	*S1*	1.5400	0.8064	0.0703
	S2	1.5842	0.8104	0.1026
	S3	1.5634	0.8350	0.0706
40	*S1*	5.4998	2.4788	0.4117
	S2	8.9498	2.2996	0.6888
	S3	5.5682	2.5446	1.2025
50	*S1*	18.7624	6.0826	1.0218
	S2	14.3340	5.9290	1.7616
	S3	14.5778	6.2482	1.8145
60	*S1*	55.0242	12.742	1.4804
	S2	88.4128	13.1630	2.4190
	S3	59.4282	12.9518	2.8943

First of all, from Table 4.2 and 4.3, it is observed that the Gap values are below 3.19 in all cases and increase as the number of jobs increases. However, it is not noticed any tendency with respect to the increment of the number of machines, yielding the highest gap values for 4 and 6 machines. Moreover, given a number of machines and jobs, the gap increases as the setup (S1 to S3) increases, especially for more than 40 jobs. Finally, notice that for each range, the gaps are kept very close regardless of the number of machines, especially for a higher number of jobs (50 and 60).

If we pay attention to the CPU time, it increases as the size of the instance increases, especially when the number of jobs grows. For the mathematical model, the CPU time decreases as the number of machines increases. This means that long sequences in the machines are more difficult to deal for the model. Furthermore, for higher number of jobs, the CPU time increases as the setup time range increase. For the hybrid heuristic, the time also increases with the number of jobs, but contrary to the model occurs. The shorter CPU time correspond to a small number of machines with longer sequences. In any case, the variability is very small with respect to the number of machines.

With the purpose of evaluating the performance of the algorithm in larger instances we tried to solve to optimality, using the time dependent formulation based on flow (model5), the instances with 70 and 80 jobs. However, the solver could not find any feasible solution for instances with 80 jobs.

In Tables 4.4 and 4.5 we show the comparison between the results obtained by the hybrid algorithm HMA and the optimal solutions or best integer solutions found by the solver for 70-jobs instances. For both tables, column 1 indicates the setup range (S), while columns 2 indicate the number of machines (m). Columns 3, 4 and 5 refer to the model. Specifically, columns 3 show the objective value, columns 4 the CPU time in seconds, while columns 5 the gap reported by the solver. The symbol (-) in columns 4 means that the solver could not reach the optimal solutions and stopped reporting “out of memory”. Columns 6, 7, 8 refer to the hybrid algorithm (HMA). Specifically, columns 6 indicate the value of the objective function; columns 7 indicate the elapsed CPU time in seconds, while columns 8 display the gap of HMA related to the best solution found by the solver.

Table 4. 4 Comparison of HMA with best known solutions for 70-job instances (2 and 4 machines)

		Model			*HMA*		
m	*S*	*Obj. Val.*	*Time (s)*	*Gap*	*Obj. Val.*	*Time (s)*	*Gap*
2	*S1*	40999	(-)	1.33	40958	23.17	-0.10
		45037	(-)	0.63	45648	23.90	1.36
		47313	2820.02	0.00	47735	24.34	0.89
		49947	(-)	1.05	49834	23.90	-0.23
		52095	(-)	0.76	52587	23.12	0.94
	S2	50705	(-)	0.24	51531	22.16	1.63
		50908	(-)	0.59	51871	22.42	1.89
		49947	(-)	1.48	50206	21.91	0.52
		53089	(-)	0.88	53760	23.32	1.26
		56208	(-)	1.07	57107	22.19	1.60
	S3	54798	(-)	0.68	55592	25.17	1.45
		49193	(-)	0.77	50883	22.80	3.44
		46169	(-)	2.09	46750	23.74	1.26
		55716	(-)	0.83	56564	24.55	1.52
		46629	(-)	2.36	47161	23.92	1.14
4	*S1*	21285	(-)	0.20	21639	24.83	1.66
		23582	879.38	0.00	23626	25.66	0.19
		24932	(-)	0.11	25143	25.31	0.85
		25953	(-)	0.53	26083	25.45	0.50
		26978	1182.25	0.00	27428	23.66	1.67
	S2	26772	1191.39	0.00	27368	23.91	2.23
		26512	968.86	0.00	26954	22.91	1.67
		26037	(-)	0.32	26451	24.46	1.59
		27785	(-)	0.52	28251	25.28	1.68
		29220	710.57	0.00	30145	23.34	3.17
	S3	28745	996.79	0.00	29324	26.26	2.01
		26003	(-)	0.78	26606	24.15	2.32
		24427	(-)	1.71	24504	24.09	0.32
		29143	(-)	0.74	29644	24.95	1.72
		24315	2714.33	0.00	24732	27.49	1.71

Table 4. 5 Comparison of HMA with best known solutions for 70-job instances (6 and 8 machines)

		Model			*HMA*		
m	*S*	*Obj. Val.*	*Time (s)*	*Gap*	*Obj. Val.*	*Time (s)*	*Gap*
6	*S1*	14948	(-)	0.12	15127	24.41	1.20
		16605	815.13	0.00	16792	25.15	1.13
		17502	(-)	0.16	17652	24.85	0.86
		18149	97.94	0.00	18401	23.65	1.39
		18737	506.96	0.00	18895	23.69	0.84
	S2	18827	875.52	0.00	19208	22.99	2.02
		18589	543,621	0.00	19035	26.10	2.40
		17502	(-)	0.17	18622	25.04	6.40
		19526	(-)	0.29	19770	23.9	1.25
		20527	957.41	0.00	20933	23.65	1.98
	S3	20243	757.81	0.00	20560	26.95	1.57
		18212	636,341	0.00	18527	25.79	1.73
		17077	965.80	0.00	17494	25.02	2.44
		20245	606.28	0.00	20720	24.83	2.35
		17196	794.99	0.00	17487	28	1.69
8	*S1*	11839	91.57	0.00	11940	23.90	0.85
		13119	105.62	0.00	13271	24.52	1.16
		13821	207.24	0.00	13865	24.61	0.32
		14326	236.38	0.00	14418	24.67	0.64
		14685	124.48	0.00	14786	23.22	0.69
	S2	14873	289.35	0.00	15015	23.14	0.95
		14675	271.94	0.00	14939	24.75	1.80
		14590	84.44	0.00	14778	25.62	1.29
		15417	727.74	0.00	15582	23.83	1.07
		16241	493.92	0.00	16465	22.87	1.38
	S3	16003	96.30	0.00	16303	26.99	1.87
		14437	(-)	0.34	14736	25.11	2.07
		13618	559.13	0.00	13965	24.78	2.55
		15936	105.34	0.00	16213	24.76	1.74
		13702	(-)	0.35	14122	26.99	3.07

From both tables, it can be conclude that even though we have no guarantee that the solver could find always the optimal solutions, the gaps reported by the solver are quite small, indicating that the solutions reported by the solver are really close to the optimal solutions, in the case that they are not already the optimal ones.

As we have seen in the comparisons, the results obtained with the proposed hybrid algorithm are very good, since the deviation from optimal or almost optimal solutions are very small in all cases. Moreover, in two instances with 2 machines and setups in the range S1, the algorithm obtains better solutions than those found by the solver.

4.3.3 Comparing with the state of the art

As we mentioned in chapter 2, Morales et al., (2015) developed an IGA for the same problem addressed here. For this reason, we present a comparison between two algorithms. To ensure fair comparisons, both metaheuristic algorithms were run on the same computer for each instance. The computer code for IGA was kindly provided by their authors.

Table 4.6 and 4.7 show the results of this comparison. The structure of the tables: column 1 indicates the number of jobs (n), column 2 the setup range (S). The columns 3 and 6 show the CPU time in seconds for the IGA for 2, 4, 6 and 9 machines respectively, and in the same way the columns 4 and 7 for the HMA. We present the percentage improvement (Improv%) of our proposed algorithm in relation to IGA in the columns 5 and 8.

$$\boldsymbol{Improv\%} = \mathbf{100} * \frac{\boldsymbol{IGA_{val}} - \boldsymbol{HMA_{val}}}{\boldsymbol{IGA_{val}}} \qquad (\mathbf{4.10})$$

where IGA_{val} is the best solution value reported by the Iterated Greedy algorithm, and HMA_{val} is the best solution value obtained by the proposed HMA.

Table 4. 6 Comparison with IGA (2 and 4 machines)

		m=2			m=4		
		CPU Time (s)		Improv	CPU Time (s)		Improv
n	S	IGA	HMA	%	IGA	HMA	%
20	S1	0.12	0.15	0.00	0.17	0.16	0.04
	S2	0.12	0.15	0.00	0.17	0.16	0.00
	S3	0.12	0.14	0.00	0.16	0.16	0.00
30	S1	0.63	0.74	0.19	0.79	0.78	0.30
	S2	0.62	0.73	0.58	0.78	0.80	0.72
	S3	0.61	0.75	1.01	0.77	0.82	0.69
40	S1	2.10	2.37	0.45	2.64	2.49	0.55
	S2	2.03	2.30	0.84	2.51	2.45	0.80
	S3	2.01	2.32	1.13	2.47	2.57	1.08
50	S1	5.41	5.85	0.98	6.47	6.21	0.57
	S2	5.23	5.49	1.91	6.25	5.91	1.37
	S3	5.10	5.80	1.91	6.18	6.20	1.49
60	S1	12.07	12.26	0.60	14.37	12.72	0.69
	S2	11.63	12.49	1.73	13.63	12.80	1.30
	S3	11.38	12.30	1.98	13.22	12.36	1.64
70	S1	18.95	23.69	2.20	22.37	24.98	2.10
	S2	18.61	22.40	3.37	21.33	23.98	2.89
	S3	18.23	24.04	3.73	21.28	25.39	3.63
80	S1	35.25	40.97	2.10	41.10	43.09	2.21
	S2	33.68	40.77	3.91	40.44	41.40	2.88
	S3	31.99	42.23	3.93	39.31	42.08	4.26

Table 4. 7 Comparison with IGA (6 and 8 machines)

		m=6			m=8		
		CPU Time *(s)*		Improv	CPU Time *(s)*		Improv
n	S	IGA	HMA	%	IGA	HMA	%
20	S1	0.22	0.17	0.00	0.30	0.19	0.00
	S2	0.21	0.16	0.00	0.31	0.19	0.00
	S3	0.22	0.17	0.00	0.31	0.19	0.00
30	S1	0.97	0.79	0.10	1.17	0.81	0.03
	S2	0.93	0.80	0.27	1.13	0.81	0.04
	S3	0.94	0.80	0.31	1.12	0.84	0.29
40	S1	3.00	2.48	0.36	3.48	2.48	0.53
	S2	2.84	2.42	0.81	3.27	2.30	0.34
	S3	2.86	2.53	0.79	3.29	2.54	0.41
50	S1	7.01	6.16	0.38	7.99	6.08	0.40
	S2	6.82	6.03	1.06	7.84	5.93	0.78
	S3	6.90	6.18	1.11	7.79	6.25	0.92
60	S1	15.17	12.89	0.64	17.16	12.74	0.46
	S2	14.73	12.43	1.38	16.63	13.16	0.62
	S3	14.43	12.75	1.30	16.33	12.95	0.96
70	S1	23.6	24.35	0.60	25.64	24.19	1.90
	S2	22.72	24.34	2.70	26.90	24.04	3.13
	S3	23.25	26.12	2.71	27.96	25.73	2.35
80	S1	43.04	43.06	2.12	49.32	42.06	1.57
	S2	41.80	41.27	3.28	47.72	41.07	2.99
	S3	44.56	41.75	3.42	43.32	43.26	3.48

On analyzing the solution quality, both the algorithms perform similarity for the small instances. As the number of jobs increases, the HMA yields better results than IGA. This difference becomes bigger for the same number of jobs as the setup time increases. For the same number of jobs, the largest improvements correspond to longer sequences with fewer machines.

In order to assess the statistical differences between both algorithms, the Wilcoxon non-parametric test for comparing two samples has been used. In the first test carried out in this section, the whole set of data has been considered. As the computed p-value is lower than the significance level $\alpha = 0.05$, the null hypothesis is rejected. Therefore, we conclude that the two algorithms are statistically different, and the proposed hybrid metaheuristic outperforms the best algorithm from the literature. Given the fact that the differences between both algorithms are not large, we have also conducted statistical tests for each instance size, i.e. number of jobs n, from n = 20 up to n = 80. In the case of n = 20, the computed p-value is greater than the significance level $\alpha = 0.05$. Therefore, the null hypothesis cannot be rejected, meaning that there are not statistical differences between the samples corresponding to IGA and HMA for n = 20. However,

for n = 30 up to n = 80, the Wilcoxon non-parametric test for comparing two samples finds statistical differences between both algorithms, obtaining p-values lower than the significance level $\alpha = 0.05$. Therefore, we may conclude that HMA statistically outperforms IGA.

Finally, it is concluded that for the small instances with 2 machines, the CPU time required by IGA is slightly smaller than the time required by HMA. However, HMA is faster as the number of machines increases.

4.7 Chapter conclusions

In this chapter we propose a hybrid algorithm that combines the GRASP and VNS metaheuristics for the parallel machine scheduling problem with sequence dependent setup times and the goal of minimize the TCT. The basic structure of the algorithm is divided in two phases, constructive and improvement. For the construction of initial solutions, several procedures were tested, and the best results were achieved by the constructive procedure described in Algorithm 4.2. Then, the improvement phase is implemented by means of a GVNS, with a destructive and reconstructive procedure as shaking process.

The computational results achieved in this work, corroborated through statistical tests, show the effectiveness of the proposed hybrid algorithm when compared with optimal solutions or best integer solutions found using a mathematical model, and with results of the best heuristic from the literature. We have proved that the hybrid algorithm statistically outperforms the state of the art for the problem.

Chapter 5 Study of the learning and deterioration effects on sequence dependent setup times in single and parallel machine scheduling problems

5.1. Introduction

In this chapter we deal with single and parallel machine scheduling problem with sequence dependent setup times for minimizing the sum of completion time of all jobs. In these problems we consider that the setup time of the machine and the jobs processing time can be affected by the learning or fatigue of the operators in carrying out the activities.

Time variations due to frequent repetition of operations are known in scheduling literature as learning or deterioration effects. From the literature review, we find that in all published works on problems with learning and deterioration effects, the setup times of the machines have been ignored or have been considered independent of the order in which the jobs are processed.

Most of the published papers, addressing the scheduling problems with learning and/or deterioration effects, assumed these effects over the processing times of the jobs (Biskup, 2008). The researches found in the literature can be grouped into two main categories: position-dependent and time-dependent effects. In both categories the processing time of a job is affected by a factor depending the jobs processed before it. In the first one, the learning/deterioration factor depends on the number of the all already scheduled jobs, while in the second one, it depends on the sum of the processing times of the all already scheduled jobs.

Another kind of learning/deterioration effect over processing times is past-sequence-dependent (p-s-d) setup times introduced by Koulamas and Kyparisis (2008). In the p-s-d setup time approach, the processing time p_{jr} of job j scheduled in position r is obtained as the normal processing time p_j plus a value that depends on the sum of the processing times of all already scheduled jobs, that is, $p_{jr} = s_{[r]} + p_j$, where $s_{[1]} = 0$, $s_{[r]} = b^{r-1} \sum_{k=1}^{r-1} p_{[k]}$ for $r = 1, 2, \cdots, n$ and b is constant associated with the learning/deterioration factor. They interpreted the value $s_{[r]}$ as a setup time that depends on the sum of the processing time of the all already scheduled jobs. For more details about scheduling problems with p-s-d setup times refer the recent survey done by Allahverdi (2015).

All published works that have studied the types of learning/deterioration effects described above considered that the setup time depends only on the job about to start.

It is true that in some practical applications the setup (or changeover) time may be ignored considering them as part of the processing time of the jobs. However, there also exist several applications in which they must be explicitly considered, since otherwise the costs and times would rise considerably (Allahverdi et al., 1999).

In this chapter, we consider a manufacturing environment where jobs are processed automatically and the machine settings between different types of jobs should be

executed manually. Here, the impact of human factor becomes significant for the whole production time. Therefore, from the repeated application of operations, the setup time may decrease because of some kind of learning, but they also may increase due to the fatigue of the operators (deterioration).

A first effort in the address of these kind of problems is presented in Expósito-Izquierdo et al.,(2019). They consider an identical parallel machine environment and addressed the learning and the deterioration effects on the setup times. In this work, first, an elite set of high-quality and diverse solutions ignoring the learning and the deterioration effects is generated through a modification of the hybrid algorithm proposed in Chapter 4. To assess how robust the solutions obtained are in the presence of deterioration and learning effects on setup times, a multi-agent simulation approach is applied.

In this chapter, first, we addressed the scheduling problems for a single machine in the manufacturing environment described above and propose four mixed integer formulations for each kind of effect. Then, we generalize these formulations for scheduling problems on identical parallel machines. All the formulations are assessed using test data instances.

To the best of our knowledge, this is the first time that:

- Propose mathematical formulations for scheduling problems where the learning/deterioration effects on sequence dependent setup times are considered.
- Conduct a study to determine how solutions are affected by learning or deterioration effects when setup times are sequence dependent.

5.2. Formulations of the problems

Consider a set of n independent jobs to be processed on the machines. We addressed two machine settings: a single machine and m identical parallel machines. Each job j has an associated processing time p_j and there are machine setup times s_{ij} for processing job j just after job i. In general, $s_{ij} \neq s_{ji}$. All the machines are at an initial state 0 (dummy job 0), and there is a setup time s_{0j} for processing the first job on each machine. All the jobs are available at time zero and each job should be continually processed on the same machine, i.e., preemption is not allowed.

As in the previous chapters, a sequence of k jobs for a given machine is represented by $P = \{0, [1], [2], \dots, [r-1], [r], [r+1], \dots, [k]\}$ where $[r]$ denotes the job in the position r in the sequence P.

We assume that the setup times are affected by learning effect or by deterioration effect, which depend on the number of setups that have already been done in the machine. Then, the total time required for processing the job j in position r just after job i in position $r-1$ is defined as:

$$\boldsymbol{t_{ij}^r = s_{ij}^r + p_j = f(b, r, k) s_{ij} + p_j} \qquad (\mathbf{5.1})$$

where $b \in (0,1)$ is a parameter associated with the learning/deterioration rate and k is the number of jobs in the sequence. When the values of b and k are fixed, $y = f(r)$ is

a monotonic non-increasing function for learning or a monotonic non-decreasing function for deterioration.

The objective is to find a schedule that minimizes the sum of the job completion times on the available machines; i.e., the Total Completion Time (TCT).

Given the function f and a sequence P, the TCT of P can be calculated as:

$$\boldsymbol{TCT(P) = kt^{1}_{0[1]} + (k-1)t^{2}_{[1][2]} + \cdots + 2t^{k-1}_{[k-2][k-1]} + t^{k}_{[k-1][k]}} \qquad \boldsymbol{(5.3)}$$

where the values of $t^{r}_{[i-1][i]}$ $(r = 1,2, \ldots, k)$ are calculated using expression (5.1).

For the single machine scheduling problem, the objective is to find a sequence with the n jobs that minimizes the TCT. For the parallel machine scheduling problem, the objective is to find m disjoint sequences Pr $(r = 1, 2, \ldots, m)$ in such a way that the following function is minimized

$$\boldsymbol{z = \sum_{r=1}^{m} TCT(P_r)} \qquad \boldsymbol{(5.4)}$$

In this research, we use the following functions (5.5) and (5.6) to model the learning effect and the deterioration effect, respectively.

$$\boldsymbol{f(b, r, k) = b^{r-1}} \qquad \boldsymbol{(5.5)}$$
$$\boldsymbol{f(b, r, k) = b^{k-r}} \qquad \boldsymbol{(5.6)}$$

where $k = n$ for the single machine problems.

The values of the t^{r}_{ij} are calculated depending on the considered effect. That is,

$$\boldsymbol{t^{r}_{ij} = b^{r-1}s_{ij} + p_j} \qquad \boldsymbol{(5.7)}$$

for the learning effect, and

$$\boldsymbol{t^{r}_{ij} = b^{k-r}s_{ij} + p_j} \qquad \boldsymbol{(5.8)}$$

for the deterioration effect.

5.2.1. Mathematical formulations for the single machine scheduling problems with learning and deterioration effects.

In this section we adapt a formulation developed in Angel Bello et al. (2013) for the Minimum Latency Problem (MLP). In that paper, they developed two improved formulations for the MLP and assessed them using routing and scheduling instances. We select the second formulation (Model B) because it had a better performance on the scheduling instances.

To adapt that formulation to the problem with learning and deterioration effects, let us define the following decision variables.

$$y^{r}_{ij} = \begin{cases} 1, & \text{there are } n-r \text{ jobs in the sequence after job } i \\ 0, & \text{otherwise} \end{cases}$$

Then the adapted formulation is shown below.

$$\min z = n\sum_{i=1}^{n} t_{0i}^{1} \sum_{j=1,j\neq i}^{n} y_{ij}^{1} + \sum_{r=1}^{n-1}\sum_{i=1}^{n}\sum_{j=1,j\neq i}^{n} (n-r)t_{ij}^{r}y_{ij}^{r} \qquad (5.9)$$

subject to:

$$\sum_{i=1}^{n}\sum_{j=1,j\neq i}^{n} y_{ij}^{1} = 1 \qquad (5.10)$$

$$\sum_{j=1,j\neq i}^{n} (y_{ij}^{r} - y_{ji}^{r-1}) = 0 \qquad (i = 1,2,\dots,n; r = 2,3,\dots,n-1) \qquad (5.11)$$

$$\sum_{i=1}^{n}\sum_{j=1,j\neq i}^{n} y_{ji}^{n-1} = 1 \qquad (5.12)$$

$$\sum_{j=1,j\neq i}^{n} y_{ij}^{1} + \sum_{r=1}^{n-1}\sum_{j=1,j\neq i}^{n} y_{ji}^{r} = 1 \qquad (i = 1,2,\dots,n) \qquad (5.13)$$

$$y_{ij}^{r} \in \{0,1\} \qquad (i = 1,2,\dots,n; j = 1,2,\dots,n; j \neq i; r = 2,3,\dots,n-1) \qquad (5.14)$$

The objective function (5.9) is to minimize the TCT of a given sequence. The variables y_{ij}^{1} are multiplied by n in (5.9) because $y_{ij}^{1} = 1$ means that the job i occupies the position 1 in the sequence. In general, $y_{ij}^{r} = 1$ means that the job i occupies the position $r-1$ and the job j occupies the position r in the sequence. This fact is equivalent to that after job i there are $n-r$ jobs in the sequence and hence the variables y_{ij}^{r} are multiplied by $n-r$.

Constraint (5.10) guarantees that a single job occupies position 1 in the sequence, while constraint (5.12) guarantees that a single job occupies the last position in the sequence. Constraints (5.11) are the flow conservation restrictions and they establish the sequence continuity. Constraints (5.13) force each job to occupy only one position in the sequence. Finally, constraints (5.14) establish the binary nature of the y_{ij}^{r} variables.

5.2.2. Mathematical formulations for the parallel machine scheduling problems with learning and deterioration effects.

For the parallel machine scheduling problems, we generalize the model 3.5 developed in Chapter 3. In a parallel machine setting it is not known in advance how many jobs will be assigned to each machine, we only have an upper bound $N = n - m + 1$ that represents the maximum number of jobs that can be assigned to a machine. For that reason, in model5 the definition of variables y_{ij}^{r} had to be modified. In this case, y_{ij}^{r} is a binary variable that is equal to 1 if and only if there are r jobs after job i in the sequence to which the job i belongs. Note that now $y_{ij}^{1} = 1$ means that job j occupies the last position in some sequence. This fact caused that developing the formulation for the deterioration effect is practically straight, while for the learning effect it is much more complicated.

In fact, for the deterioration effect, the objective function coefficients for the variables y_{ij}^1 are $s_{ij} + p_j$, for y_{ij}^2 are $2(bs_{ij} + p_j)$ and, in general, for y_{ij}^r are $r(b^{r-1}s_{ij} + p_j)$, then the objective function can be written as:

$$\min z = \sum_{j=1}^{n}\sum_{r=1}^{N} r(b^{r-1}s_{0j} + p_j)y_{0j}^r + \sum_{i=1}^{n}\sum_{j=1,j\neq 1}^{n}\sum_{r=1}^{N-1} r(b^{r-1}s_{ij} + p_j)y_{ij}^r \qquad (5.15)$$

In the learning effect situation, $y_{ij}^1 = 1$ also means that job j occupies the last position in some sequence and the variables y_{ij}^1 should be multiplied in the objective function by $b^{k-1}s_{ij} + p_j$, where k is the number of jobs in the sequence to which the job j belongs. However, as the values of k for each sequence are not known until the problem is solved, we have to substitute the value of k by N to obtain the objective function coefficients. Then, the objective functions can be written as follow:

$$\min z = \sum_{j=1}^{n}\sum_{r=1}^{N} r(b^{N-r}s_{0j} + p_j)y_{0j}^r + \sum_{i=1}^{n}\sum_{j=1,j\neq 1}^{n}\sum_{r=1}^{N-1} r(b^{N-r}s_{ij} + p_j)y_{ij}^r \qquad (5.16)$$

The fact that the variables in the objective function are not multiplied by the correct coefficients could affect the optimal solution and it should be verified in computational experimentation.

For both problems, the set of constraints is the same as for model 3.5 and we rewrite it below

$$\sum_{r=1}^{N} y_{0j}^r + \sum_{r=1}^{N-1}\sum_{i=1,i\neq j}^{n} y_{ij}^r = 1 \qquad (i = 1, 2, \dots, n) \qquad (5.17)$$

$$\sum_{j=1}^{N} y_{0j}^1 + \sum_{i=1}^{n}\sum_{j=1,j\neq i}^{n} y_{ij}^1 = m \qquad (i = 1, 2, \dots, n) \qquad (5.18)$$

$$\sum_{r=1}^{N}\sum_{j=1}^{n} y_{0j}^r = m \qquad (5.19)$$

$$y_{0j}^{r+1} + \sum_{i=1,i\neq j}^{n} y_{ij}^{r+1} = \sum_{i=1,i\neq j}^{n} y_{ji}^r \qquad (i = 1, 2, \dots, n; r = 1, 2, \dots, N-2) \qquad (5.20)$$

$$y_{0j}^N = \sum_{i=1,i\neq j}^{n} y_{ji}^{N-1} \qquad (i = 1, 2, \dots, n) \qquad (5.21)$$

$$y_{0j}^r \in \{0, 1\} \qquad (i = 1, 2, \dots, n; r = 1, 2, \dots, N) \qquad (5.22)$$

$$y_{ij}^r \in \{0, 1\} \qquad (i, j = 1, 2, \dots, n; j \neq i;\ r = 1, 2, \dots, N-1)$$

Constraints (5.17) ensure that each job is processed by a single machine. Constraints (5.18) and (5.19) together ensure that there are an initial and a final job on each machine. Constraints (5.20) are the flow conservation constraints and they guarantee the continuity of the sequences. Constraints (5.21) are used when there are sequences containing exactly N jobs, otherwise they are redundant. Finally, constraints (5.22) establish the binary nature of the variables.

Considering the nature of effect and the machine environment we have 4 models, that are summarized in Table 5.1.

Table 5. 1 Proposed models

	Learning effect	Deterioration effect
Single Machine Problem	Model 1	Model 2
Parallel Machine Problem	Model 3	Model 4

5.3. Computational experiments

The objective of the experiments is to show the advantages of including the learning effect or the deterioration effect in the production programming process. To assess the impact of the learning/deterioration effect on the quality of the solutions, we first take the optimal solution without considering the learning/deterioration effect and evaluate it for each value of the learning/deterioration factor b. Then the obtained objective function values are compared with the values of the optimal solution with learning or deterioration effect for the corresponding b value. That is, the gap value for each b value is calculated as:

$$\boldsymbol{Gap(b)\% = 100 * \frac{SolVal_{witoutEffect}(b) - OptVal(b)}{OptVal(b)}} \qquad (\mathbf{5.23})$$

where $OptVal(b)$ is the optimal solution value obtained for a given value of b and $SolVal_{withoutEffect}(b)$ is the value of the solution without learning/deterioration effect when it is evaluated for the same b value. Note that, these gaps represent a measure of how good a solution is (obtained without considering the learning/deterioration effects) when it is evaluated for the different values of the learning/deterioration factor.

The experiments for single machine environment were performed on an Intel Core 2 Duo CPU at 3.00 GHz and 3.21 GB of RAM under Windows OS. The formulations were implemented in C++ using the Gurobi Optimizer 8.1 solver. While the experiments for the parallel machine environment were performed on an Intel Core i5-5200U CPU at 2.20 GHz and 8 GB of RAM Windows 10 Enterprise 64 bits, also were implemented in C++ using the Gurobi Optimizer 8.1 solver.

To carry out the computational experiments we use a subset of instances used in Chapters 3 and 4. Specifically, we use the instances with $n = \{15,20,25,30\}$ for the single machine problems and the instances with $n = \{20,30,40,50,60\}$, $m = \{2,4,6,8\}$ for the parallel machine problems. We take instances of the three setup ranges S_1, S_2, and S_3. The values of the learning/deterioration factor b vary between 0.1 and 0.9 with a step of 0.1. In addition, we set $b = 1$ to obtain the optimal solutions without learning/deterioration effect.

5.3.1 Single machine

The Table 5.2 shows the gaps for the single machine problems with learning effects and with deterioration effect. In Table 5.2, column 1 (n) indicates the number of jobs, while column 2 (m) shows the different values of b. Columns 3 to 5 are associated to model 1 while columns 6 to 8 are related to Model 2. These columns show the gap values for

setup ranges S_1, S_2, and S_3, respectively. The gap values are calculated using the expression (5.23), the obtained values are grouped in subset of 20 values according to number of jobs (n), setup range (S) and level of learning factor (b) and then the average of each group is calculated. The gaps shown in columns 3 to 8 of Table 5.2 are averaged over 20 instances.

***Table 5. 2** Gaps of optimal solutions without learning/deterioration effect regarding learning and deterioration levels for the single machine problems.*

		Gap (%) for $m = 1$					
		Learning effect			***Deterioration effect***		
n	***b***	***S_1***	***S_2***	***S_3***	***S_1***	***S_2***	***S_3***
15	0.1	6.1103	11.339	9.4763	6.8019	12.5440	12.7163
	0.2	5.7949	10.691	8.8852	6.7091	12.3841	12.4956
	0.3	5.4789	9.9187	8.1433	6.5977	12.1735	12.1770
	0.4	5.1246	9.0211	7.1655	6.4317	11.9149	11.7583
	0.5	4.7052	8.0269	6.1133	6.1151	11.5098	11.1551
	0.6	4.0679	6.7245	4.9972	5.5805	10.8178	10.1572
	0.7	3.1224	4.9213	3.7362	4.6928	9.5542	8.6373
	0.8	2.0394	2.8022	2.0618	3.2233	7.1155	6.2352
	0.9	0.8367	0.8288	0.5609	1.1919	2.9711	2.5838
	1	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
20	0.1	6.4371	8.8005	10.8290	6.8348	10.5417	12.3654
	0.2	6.1681	8.3507	10.3060	6.7931	10.4496	12.2257
	0.3	5.8133	7.9582	9.7336	6.7405	10.3302	12.1039
	0.4	5.4159	7.5540	9.1023	6.6667	10.1620	11.9066
	0.5	5.0203	7.0093	8.2665	6.5207	9.9132	11.5929
	0.6	4.5118	6.1175	7.0491	6.2235	9.4573	10.9642
	0.7	3.7495	4.8691	5.5179	5.6995	8.6112	9.75419
	0.8	2.5826	3.1717	3.6202	4.6171	6.8644	7.4269
	0.9	1.0175	1.0446	1.3362	2.2317	3.2501	3.1813
	1	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
25	0.1	4.7195	9.2083	10.9102	5.0703	10.0593	11.7052
	0.2	4.5470	8.9036	10.6019	5.0438	10.0068	11.6461
	0.3	4.3534	8.5460	10.2314	5.0034	9.9529	11.5582
	0.4	4.1317	8.1890	9.7719	4.9503	9.8646	11.4203
	0.5	3.8501	7.6828	9.1469	4.8598	9.7080	11.1913
	0.6	3.4658	6.9167	8.2511	4.6989	9.4203	10.7742
	0.7	2.9227	5.6916	7.0393	4.3941	8.7735	10.0016
	0.8	2.0447	3.9916	5.2446	3.6531	7.2732	8.2702
	0.9	0.7964	1.8524	2.4622	2.0458	3.8755	4.4306
	1	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
30	0.1	4.5748	7.5836	8.6234	5.0546	8.3731	9.4431
	0.2	4.4401	7.3500	8.3754	5.0360	8.3436	9.4079
	0.3	4.3107	7.1037	8.1304	5.0133	8.3032	9.3651
	0.4	4.1544	6.8147	7.8632	4.9809	8.2367	9.2962
	0.5	3.9541	6.4499	7.5415	4.9179	8.1243	9.1871
	0.6	3.6682	5.9232	7.0117	4.8007	7.9376	8.9802
	0.7	3.2200	5.0920	6.0973	4.5437	7.5644	8.5153
	0.8	2.5077	3.7320	4.5605	3.9407	6.5939	7.4593
	0.9	1.1902	1.7558	2.0321	2.3221	3.9631	4.4050
	1	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000

From the results shown in this table we can conclude that the behaviors of gaps are similar for the two types of effects, having slightly higher values when the deterioration effect is considered. For both effects, it is possible to observe that the gaps increase as the values of b decrease and the solutions without learning effect is degraded when the

range of variation of the setup times is greater than the range of variation of the processing times (S_3) for all levels of the learning factor. It can also be seen that the values of the gaps decrease as the n value decreases.

In summary, taking into account the values of gaps, we can conclude that the quality of the solution without learning and deterioration effects is not very low, but if we want better quality solutions, we should consider the learning or deterioration factors in the process of production programming.

The following set of graphics shows the performance of Model 1 (left column) and Model 2 (right column) regarding the CPU time spent to find the optimal solution according to the number of jobs and ***b*** values. Notice that that for all the instances the optimal solution was reached in shorter times to 13 seconds.

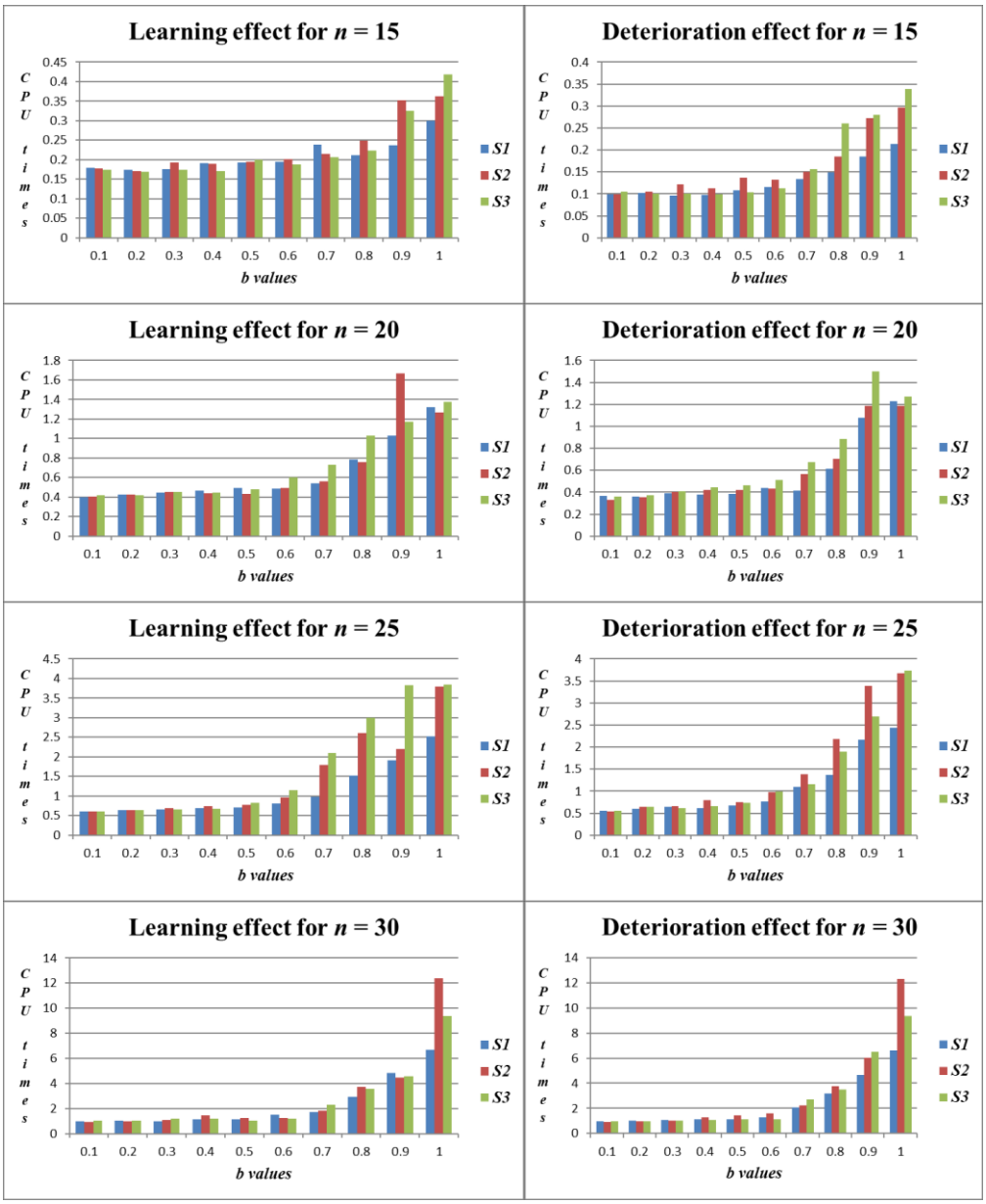

***Figure 5. 1** Behavior of the CPU time according to the number of jobs, setup ranges and learning/deteriration levels.*

5.3.2 Parallel machines

Next, we present in Tables 5.3 to 5.6 the results corresponding to the experiments carried out for the parallel machine environment. These tables have the same structure as Table 5.2.

***Table 5. 3** Gaps of optimal solutions without learning/deterioration effect regarding learning and deterioration levels for the parallel machine problems with m=2.*

		Gap (%) for $m = 2$					
		Learning effect			*Deterioration effect*		
n	***b***	S_1	S_2	S_3	S_1	S_2	S_3
15	0.1	65.8988	117.2618	148.6121	33.3592	46.1247	63.9454
	0.2	57.3466	107.7226	136.0523	27.1017	45.2606	58.8916
	0.3	56.1139	106.0798	131.4117	25.7830	41.2723	52.4629
	0.4	56.0213	105.8713	105.8713	23.3960	35.1890	42.4357
	0.5	55.6336	107.0921	115.6978	19.7490	31.6181	36.9881
	0.6	49.1218	73.7468	110.9999	15.9645	25.9787	30.1112
	0.7	37.9911	65.3583	82.9784	12.9309	16.5068	23.3729
	0.8	27.2916	49.2469	58.8745	6.4501	12.6451	17.1958
	0.9	12.4260	21.1823	28.3144	2.8846	5.3367	7.5704
	1	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
20	0.1	29.6845	68.6843	84.6869	21.6347	54.0124	68.2221
	0.2	26.7749	63.6888	81.9099	20.6611	53.5653	65.6686
	0.3	23.2158	57.5764	74.8439	19.3790	50.9691	64.6102
	0.4	22.2430	56.7814	72.4027	19.0240	41.3628	55.3676
	0.5	22.2271	56.6599	72.2982	17.8828	32.3678	46.7920
	0.6	22.1307	50.9155	66.7791	16.0312	28.5667	36.8245
	0.7	21.0984	37.5212	47.6667	12.9085	21.0555	26.9554
	0.8	15.6957	24.6936	28.6338	8.1278	12.6731	15.4265
	0.9	8.1613	12.3744	15.0748	4.1578	6.0318	6.3887
	1	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
25	0.1	29.6696	81.7450	91.3666	31.9133	65.2400	77.8600
	0.2	33.0030	79.3130	98.6707	24.5021	62.3570	66.6925
	0.3	28.1908	70.6129	79.4901	24.3533	57.4285	64.3315
	0.4	26.2815	67.6637	73.4072	24.0074	54.5968	63.2773
	0.5	25.1486	66.5166	71.0886	22.3531	43.0495	54.5477
	0.6	25.1133	62.6190	67.4367	17.1659	37.5331	39.9620
	0.7	21.8149	52.9193	55.0600	11.4410	26.2520	29.4972
	0.8	16.4713	34.4326	38.7127	8.4438	17.8491	18.3941
	0.9	8.8682	17.0180	18.9507	4.6076	8.7182	9.3480
	1	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
30	0.1	34.3000	74.1058	89.8460	41.0862	61.0444	78.2396
	0.2	34.5677	78.7809	89.6706	35.0853	58.7093	69.3125
	0.3	33.5672	71.3780	82.9029	34.6978	56.3957	66.1241
	0.4	29.3800	63.8376	76.0297	25.4373	52.4543	64.9755

	0.5	28.2502	60.9628	72.2329	23.9223	47.1869	59.1499
	0.6	27.8092	60.8676	71.9826	18.7011	39.8884	45.5328
	0.7	24.4392	55.0765	62.0507	14.8001	25.8324	30.8978
	0.8	19.3490	33.6296	41.9418	10.9380	16.3639	18.3497
	0.9	8.5936	14.8366	18.1533	4.8082	7.4153	8.6056
	1	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000

***Table 5. 4** Gaps of optimal solutions without learning/deterioration effect regarding learning and deterioration levels for the parallel machine problems with m=4.*

Gap (%) for $m = 4$							
		Learning effect			*Deterioration effect*		
n	***b***	S_1	S_2	S_3	S_1	S_2	S_3
15	0.1	40.4618	103.2302	176.3518	12.4581	30.7029	46.2300
	0.2	32.7598	80.97617	130.6997	11.4328	26.5718	42.6540
	0.3	32.7425	80.9713	130.5875	9.5744	23.5422	37.3594
	0.4	32.7285	80.9200	130.4706	8.3646	19.9142	34.0003
	0.5	32.6416	80.6098	129.7798	6.4721	17.5761	27.8860
	0.6	30.0632	71.7061	123.5713	5.5512	11.2310	24.4680
	0.7	28.8458	62.8895	105.5198	4.0505	8.2168	19.7222
	0.8	21.1277	41.6441	69.1655	2.8349	5.4365	12.4457
	0.9	11.2809	20.4470	29.9021	1.5011	2.8247	4.0382
	1	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
20	0.1	26.6037	62.6921	79.4990	18.0655	40.5917	48.7050
	0.2	30.8962	60.8647	75.2225	17.6185	35.9406	44.1168
	0.3	22.2829	47.8193	55.1943	15.3851	30.7262	38.8545
	0.4	20.9139	46.3679	54.4119	13.6839	27.5846	33.8624
	0.5	20.9097	46.3542	54.3939	10.9735	21.4982	26.7768
	0.6	20.8708	46.2294	54.2313	9.4616	18.3370	21.7896
	0.7	20.0615	38.9195	50.2908	6.8382	11.6825	17.9493
	0.8	15.5362	29.8057	39.4851	4.5725	7.75103	8.6549
	0.9	9.2875	14.6785	15.4480	2.3104	3.5232	3.34533
	1	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
25	0.1	32.0063	78.8046	86.4863	18.8724	45.9892	51.8287
	0.2	33.4189	71.5805	87.6163	18.5183	42.4285	47.9878
	0.3	30.8917	67.3468	79.0594	17.1827	37.3564	44.4969
	0.4	21.2725	47.7717	54.6499	15.5603	31.6937	37.1436
	0.5	20.2155	47.7523	54.6353	13.1534	28.5494	33.4884
	0.6	20.2102	47.7347	54.6095	10.7361	25.5196	25.3628
	0.7	20.1455	47.5204	54.3116	8.81491	18.5784	18.3022
	0.8	18.7261	39.6577	42.0300	5.2076	11.3142	11.9980

	0.9	11.3925	20.4205	19.5406	2.4506	5.7076	4.2022
	1	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
30	0.1	35.1008	79.2002	92.7947	21.1632	45.6224	51.3321
	0.2	35.5886	78.8567	89.0508	20.6299	43.1693	50.2063
	0.3	35.0116	78.3900	87.4945	19.7945	38.8619	43.7551
	0.4	28.9291	62.0848	67.1610	17.8032	34.5329	35.0348
	0.5	21.6587	48.4375	55.1640	15.7991	28.1976	26.3221
	0.6	21.6578	48.0011	55.1552	13.1217	19.9469	22.0825
	0.7	21.6385	47.9437	55.0781	9.6380	14.4200	18.2064
	0.8	20.3540	38.1562	39.7184	6.5157	8.9205	11.3348
	0.9	12.3513	19.0112	21.5025	3.2246	4.6473	5.0021
	1	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000

***Table 5. 5** Gaps of optimal solutions without learning/deterioration effect regarding learning and deterioration levels for the parallel machine problems with m=6.*

Gap (%) for m = 6

		Learning effect			***Deterioration effect***		
n	***b***	***S_1***	***S_2***	***S_3***	***S_1***	***S_2***	***S_3***
15	0.1	49.7343	123.0367	100.2296	17.1926	37.4436	37.11881
	0.2	44.2389	103.0189	93.6742	15.5637	35.2860	33.51412
	0.3	44.2305	102.9935	93.6529	13.6654	28.2435	29.53765
	0.4	44.1693	102.7983	93.4896	11.8256	23.8679	24.98431
	0.5	43.8824	101.8756	92.7110	9.5995	17.0013	18.78135
	0.6	42.8792	98.4092	88.8095	7.5033	11.4890	13.72476
	0.7	38.8980	88.5730	76.5489	5.8341	8.1432	9.122875
	0.8	30.3067	58.9317	54.9975	3.6608	4.9148	4.803582
	0.9	17.7898	26.7920	25.5195	1.7386	2.3516	1.85242
	1	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
20	0.1	31.9935	60.8220	74.0027	18.6296	36.2645	37.8881
	0.2	34.1885	54.8085	64.4170	16.6657	34.0065	33.5824
	0.3	22.5084	45.2000	48.6505	15.4161	29.5191	29.1725
	0.4	22.5077	45.1981	48.6484	13.2256	24.8835	23.6151
	0.5	22.4990	45.1750	48.6240	11.9441	20.5652	19.9242
	0.6	22.2758	44.9923	48.4289	9.7804	17.0327	14.6506
	0.7	21.8833	43.2677	45.4381	6.3988	11.0739	10.5056
	0.8	19.0285	34.4068	33.1484	4.3334	5.9063	6.4146
	0.9	10.3476	18.1288	18.0198	1.9345	2.7431	2.9016
	1	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
25	0.1	34.5739	77.4411	82.5544	19.8370	41.8353	38.4771
	0.2	36.5830	77.5399	76.2059	18.4703	38.1728	36.0883
	0.3	35.6394	70.2485	63.6900	16.0151	34.6811	29.9502
	0.4	22.3436	49.9653	44.9373	14.4244	29.4165	26.5812
	0.5	22.2663	49.8369	44.9362	12.0316	26.6913	22.3191

	0.6	22.2589	49.8119	44.9157	9.0006	22.1702	16.7435
	0.7	22.1726	49.5290	44.6827	7.4514	15.2087	12.2421
	0.8	20.6150	45.4081	38.8568	4.9241	10.4084	7.4332
	0.9	11.3171	25.2039	21.2901	2.4048	4.4015	3.0274
	1	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
30	0.1	42.3898	65.2722	93.8534	20.5196	32.6962	48.1899
	0.2	43.0621	68.7697	88.0670	18.8980	30.9766	45.1150
	0.3	38.6824	66.3541	93.7071	17.3083	26.8549	38.7134
	0.4	35.6265	55.0134	78.8586	15.7349	21.7282	32.6064
	0.5	22.0939	34.8728	56.4391	13.1781	18.2902	27.0093
	0.6	21.9972	34.8713	56.4247	10.6114	13.6287	19.6891
	0.7	21.9413	34.8369	54.5981	8.1384	10.6295	15.6506
	0.8	21.1410	32.5453	46.85765	5.4018	6.6405	9.1559
	0.9	14.9692	19.91788	23.6965	2.7393	3.4431	4.1513
	1	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000

***Table 5. 6** Gaps of the optimal solutions without learning/deterioration effect regarding learning and deterioration levels for the parallel machine problems with m=8.*

Gap (%) for m = 8

		Learning effect			***Deterioration effect***		
n	***b***	***S_1***	***S_2***	***S_3***	***S_1***	***S_2***	***S_3***
15	0.1	52.7623	104.2262	156.6708	17.4082	32.4524	47.2652
	0.2	52.6186	104.2135	156.6469	15.2727	28.5345	40.5772
	0.3	52.5703	104.0810	156.3926	12.9972	23.5191	33.1791
	0.4	52.3209	103.4011	155.0943	10.5609	18.0583	30.0564
	0.5	51.4585	98.5534	145.5622	9.3000	14.5030	26.8735
	0.6	48.4558	89.0032	121.9466	7.2243	11.7905	21.2234
	0.7	42.0989	73.4573	93.4003	5.6085	8.0415	10.2784
	0.8	29.2866	49.8121	60.8760	3.2127	4.0337	4.2018
	0.9	14.9518	22.0382	27.2192	1.2953	1.6648	1.7997
	1	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
20	0.1	34.7787	56.7727	87.2121	18.1559	37.7481	45.2558
	0.2	27.09493	50.5254	65.7982	16.0685	34.0296	42.2353
	0.3	23.5867	48.1117	65.4784	14.5181	27.7716	34.7221
	0.4	23.5844	48.1046	65.4683	12.7573	24.4758	29.2031
	0.5	23.5633	48.0376	65.3695	11.1994	21.0284	22.3429
	0.6	23.4306	47.6272	64.7564	8.8281	17.8985	17.0213
	0.7	22.8047	45.7704	58.2615	5.8635	12.1982	11.0231
	0.8	19.2400	36.2163	42.5169	3.7966	5.8459	6.5749
	0.9	12.6680	18.2417	20.8710	1.7842	2.6675	2.6858
	1	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
25	0.1	31.7761	65.50086	98.14790	18.8883	36.1262	48.8618
	0.2	29.8942	62.6424	87.9887	16.6552	31.6017	41.3085
	0.3	30.2406	51.5729	82.6668	14.7066	28.3205	38.5461
	0.4	22.1495	42.8867	60.7120	12.4363	24.1303	30.9397
	0.5	22.0632	42.8841	60.7077	10.2518	20.7899	25.0552

	0.6	22.0498	42.8481	60.6485	8.3121	16.5171	18.9872
	0.7	21.9186	34.8061	60.0769	5.5229	11.8317	12.2774
	0.8	19.7894	38.8769	52.7319	3.7454	6.7694	7.7745
	0.9	12.2690	22.5368	25.0521	1.9056	3.2156	3.3734
	1	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
30	0.1	35.1544	61.4239	76.1796	17.3623	28.6382	37.4824
	0.2	34.9034	47.8662	80.0278	16.7950	26.8772	32.7563
	0.3	35.1788	57.7330	85.0620	14.7610	24.3255	30.5952
	0.4	28.9754	46.9167	55.0123	12.9955	21.5691	24.1574
	0.5	19.8248	36.1781	44.7360	11.7649	17.5587	19.5592
	0.6	19.8235	36.1753	44.7318	8.9368	12.2319	14.3471
	0.7	19.7987	36.1200	44.6539	6.4861	9.1863	10.4738
	0.8	19.4747	32.8107	40.3748	3.6400	5.9428	7.0332
	0.9	13.7196	20.0552	24.5484	1.7923	2.8541	3.5046
	1	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000

From the results shown in Tables 5.3 to 5.6 we can conclude that the behaviors of gaps for learning effect are worse than for deterioration effect. This behavior is observable for all combination of (n, m, b, S). For both effects, it is possible to detect that the gaps increase as the values of b decrease and the solutions without learning effect degrade as the time range grows, always having worse values for S_3 for all levels of the learning factor.

In summary, we conclude that in parallel machine problems, when the learning or deterioration effects are not considered in the production programming process, it can lead us to obtain solutions of very poor quality. We also want to emphasize that it is essential to bear in mind the learning effect in the production programming process, if we want to get high quality solutions.

With the figures shown below we intend to reinforce the previous comments. Figures 5.2 and 5.3 are graphic illustrations of the gaps behavior (on average) for each level of learning factor and each setup range. Th gaps calculated by the expression (5.23) have been grouped into subsets. To obtain the column charts in Figure 5.2 the gaps are averaged for each n value, while in Figure 5.3 they are averaged for each m value.

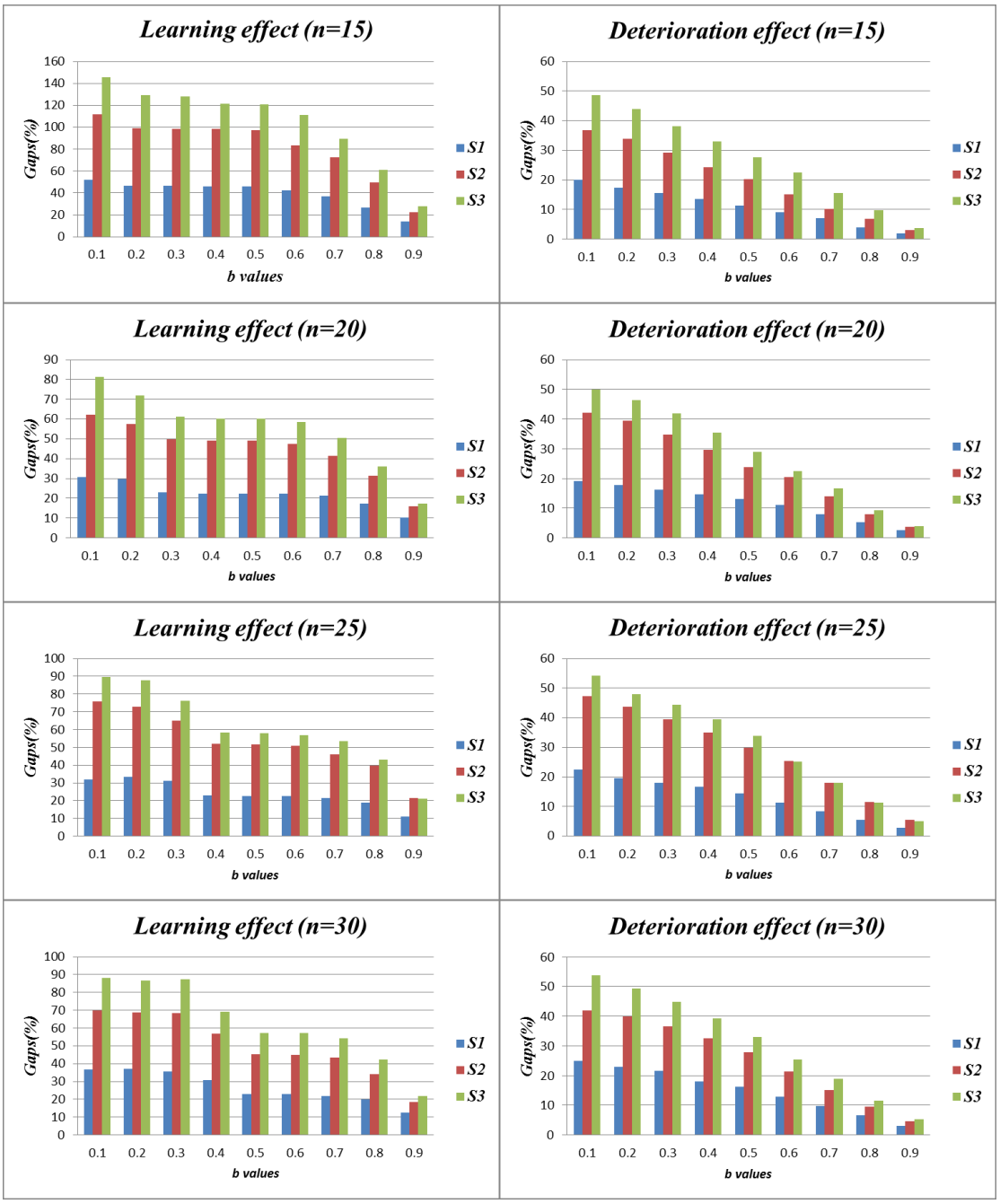

***Figure 5. 2** Averaged gaps grouped by number of jobs according to setup ranges and learning/deterioration levels.*

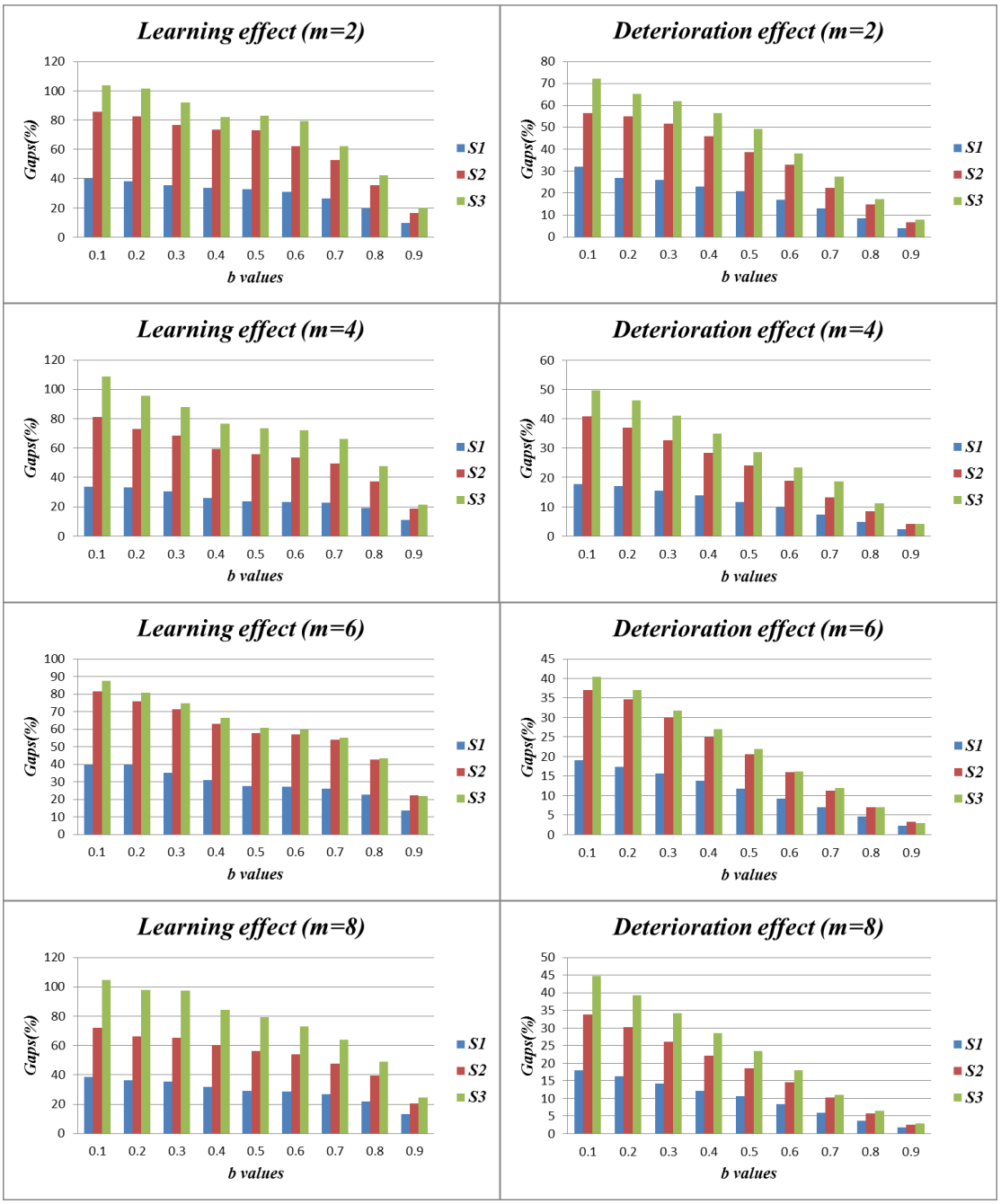

***Figure 5. 3** Averaged gaps grouped by number of machines according to setup ranges and learning/deterioration levels.*

Table 5.7 shows the range of variation for the CPU time spent by Model 3 (with learning effect) and Model 4 (with deterioration effect) solving the 30-job instances with 2, 4, 6 and 8 machines. Column 1 displays the machine number while column 2 indicates the setup time range. Columns 3 to 6 are related to Model 3 while columns 7 to 10 are related to Model 4. Columns 3 and 7 show the minimum values of CPU time while 5 and 9 show the maximum values of CPU time spent by models to obtain the optimal solutions. Columns 4, 6, 8 and 10 exhibit the *b* value for which those maximum and minimum values of CPU time are reached.

Table 5. 7 Maximum and minimum CPU times (in seconds) spent by two models for parallel machine problems solving the 30-job instances.

		Learning effect				*Deterioration effect*			
m	***S***	***Min***	***b***	***Max***	***b***	***Min***	***b***	***Max***	***b***
2	S_1	0.64	0.6	1.921	0.9	0.597	0.3	308.499	0.8
	S_2	0.601	0.6	2.29	0.9	0.695	0.1	41.227	0.9
	S_3	0.65	0.4	1.96	0.9	1.093	0.2	21.37	0.9
4	S_1	0.365	0.6	1.033	0.9	2.864	0.1	24.596	0.8
	S_2	0.354	0.7	0.818	0.9	7.014	0.3	76.337	0.9
	S_3	0.386	0.3	1.059	0.9	14.998	0.8	71.278	0.7
6	S_1	0.283	0.1	0.569	0.9	16.905	0.2	55.322	0.6
	S_2	0.267	0.7	0.592	0.9	7.502	0.5	58.616	0.8
	S_3	0.284	0.8	0.837	0.9	5.304	0.2	23.511	0.5
8	S_1	0.238	0.7	0.65	0.9	2.039	0.7	12.246	0.3
	S_2	0.248	0.6	0.801	0.9	1.606	0.3	8.976	0.7
	S_3	0.247	0.7	0.528	0.9	0.358	0.1	4.978	0.4

From the results in the table it is seen that the CPU times are relatively small and that the largest values are obtained for m = 2. This is somewhat reasonable because having fewer machines the sequences of jobs are longer.

5.4. Conclusions

In this chapter, we studied how total completion time is affected by learning effect or by deterioration effect over sequence-dependent setup times in single and the parallel machine scheduling problem. To model the learning or deterioration effects we used functions depending on the number of machine setups already completed and, on a learning, or deterioration factor.

From the experiments carried out for single machine problems, we can conclude that the solutions obtained without considering the learning/deterioration effects could be considered of acceptable quality for the different levels of *b*, although higher quality solutions could be obtained if these effects on the sequence dependent setup times are considered in the process of production planning.

For parallel machines problems the situation is different. For both types of effects, the solutions obtained without considering learning and deterioration effects are of very poor quality, deteriorating further as the *b* value decreases. For parallel machines problems we recommend to always take into account the learning and deterioration effects in the process of production planning. In addition, it is essential to consider the learning effect if we want to get high quality solutions.

Using the proposed formulations, we are able to solve optimality of up to 30 jobs. Then, interesting research avenues could be the addressed of the scope of formulations for parallel machine problems and the design of heuristic and metaheuristic algorithms to determine quality solutions for larger instances considering learning effect or deterioration effect in the production programming process.

Chapter 6 General conclusions

In this thesis work, we approach two kinds of scheduling problems and in both of them, the goal is to minimize the total completion time. The first problem is the parallel scheduling problem with sequence-dependent setup times. In the second addressed problem, we incorporate learning and deteriorating effects over the sequence-dependent setup times according to the position, and the approach was for a single machine and parallel machines environments.

6.1 Conclusions

From the first problem, the parallel scheduling problem with sequence-dependent setup times, we conclude the following:

- We presented two new time-dependent formulations for the addressed problem.
- We showed that time-dependent based formulations performed much better than the others formulations for the parallel machines scheduling problem with sequence-dependent setup times.
- In the case with many jobs and few machines, model 5 showed a better performance.
- For the same problem, propose a hybrid algorithm that combines the GRASP and Variable Neighborhood Search metaheuristics.
- After tested several orders of the neighborhoods for the improvement phase, the best results were achieved by selecting randomly the order of the neighborhoods sequence.
- Comparing with optimal solutions or best integer solutions found using a mathematical model, and with results of the best heuristic from the literature we show the effectiveness of the proposed hybrid algorithm.

From the second problem, the parallel scheduling problem with sequence-dependent setup times with learning and deteriorating effects, we conclude:

- We presented mixed integer formulations for the environments of a single and parallel machines with learning and with deterioration effects for the case when these effects act just over the setup times.
- The experiments for parallel machines environment obtained the bigger gaps percentages regarding the optimal solution without effects than the experiments for single machine environment.
- The function implemented to describe the deterioration effect presents a widely decreases in setup times for sequences with more processed jobs.

- We noticed that for these two kinds of problems (single and parallel machines), an optimal solution with learning or deterioration effects does not remain to be the optimal solution when these effects are not considered.

6.2 Future works

In this research, the learning and the deterioration effects were considered just over the sequence dependent setup times. Another research line could include these effects over the setup times and over the processing times. Also, the generalization of this addressed to other manufacturing environment with sequence dependent setup times could also be another interesting research direction.

Appendix A. Acronyms Table

Table A. 1 Table of acronyms

TCT	Total Completion Time
Makespan (Cmax)	The completion time of the final job sequenced at a schedule
m-TSP	m-Travelling Salesman Problem
TWT	Total Weighted Tardiness
TWCT	Total Weighted Completion Time
B&B	Branch-and-Bound algorithm
ATSP	Asymmetric Travelling Salesman Problem
VRP	Vehicle Routing Problem
TSP	Travelling Salesman Problem
P-s-d	Past-sequence-dependent
GRASP	Greedy Randomized Adaptive Search Procedure
VNS	Variable Neighborhood Search
VND	Variable Neighborhood Descent
RVND	Random Variable Neighborhood Descent
IGA	Iterated Greedy Algorithm
RCL	Restricted Candidate List
WMDD	Weighted Modified Due Date
HMA	Hybrid Metaheuristic Algorithm

Table A. 2 Table of classification scheme for scheduling problems.

P|STsd|TCT	Parallel machine Scheduling Problem with sequence dependent Setup Times, minimizing TCT
R|STsd|Cmax	Unrelated parallel machine scheduling problem with sequence dependent Setup Times, minimizing the makespan

Bibliography

Allahverdi, A. (2015). The third comprehensive survey on scheduling problems with setup times/costs. *European Journal of Operational Research*, *246*(2), 345–378. https://doi.org/10.1016/j.ejor.2015.04.004

Allahverdi, A., Gupta, J. N. D., & Aldowaisan, T. (1999). A review of scheduling research involving setup considerations. *Omega*, *27*(2), 219–239. https://doi.org/10.1016/S0305-0483(98)00042-5

Allahverdi, A., Ng, C. T., Cheng, T. C. E., & Kovalyov, M. Y. (2008). A survey of scheduling problems with setup times or costs. *European Journal of Operational Research*, *187*(3), 985–1032. https://doi.org/10.1016/j.ejor.2006.06.060

Allahverdi, A., & Soroush, H. M. (2008). The significance of reducing setup times/setup costs. European Journal of Operational Research, 187(3), 978-984.

Angel-Bello, F., Alvarez, A., & García, I. (2013). Two improved formulations for the minimum latency problem. *Applied Mathematical Modelling*, *37*(4), 2257–2266.

Armentano, V. A., & de França Filho, M. F. (2007). Minimizing total tardiness in parallel machine scheduling with setup times: An adaptive memory-based GRASP approach. *European Journal of Operational Research*, *183*(1), 100–114. https://doi.org/10.1016/j.ejor.2006.09.077

Avalos Rosales, O. (2014). *Secuenciación en máquinas paralelas no relacionadas con tiempos de preparación y tareas de mantenimiento preventivo* (PhD Thesis). Universidad Autónoma de Nuevo León.

Avalos-Rosales, O., Angel-Bello, F., & Alvarez, A. (2015). Efficient metaheuristic algorithm and re-formulations for the unrelated parallel machine scheduling problem with sequence and machine-dependent setup times. *The International Journal of Advanced Manufacturing Technology*, *76*(9–12), 1705–1718. https://doi.org/10.1007/s00170-014-6390-6

Baez, S., Angel-Bello, F., & Alvarez, A. (2016). Time-dependent formulations for minimizing total completion time in a parallel machine scheduling problem with dependent setup times. *IFAC-PapersOnLine*, *49*(12), 857–862.

Behnamian, J., & Fatemi Ghomi, S. M. T. (2011). Hybrid flowshop scheduling with machine and resource-dependent processing times. *Applied Mathematical Modelling*, *35*(3), 1107–1123. https://doi.org/10.1016/j.apm.2010.07.057

Bektas, T. (2006). The multiple traveling salesman problem: an overview of formulations and solution procedures. *Omega*, *34*(3), 209–219.

Bettayeb, B., Kacem, I., & Adjallah, K. H. (2008). An improved branch-and-bound algorithm to minimize the weighted flowtime on identical parallel machines with family setup times. *Journal of Systems Science and Systems Engineering*, *17*(4), 446–459. https://doi.org/10.1007/s11518-008-5065-y

Bierwirth, C., & Kuhpfahl, J. (2017). Extended GRASP for the job shop scheduling problem with total weighted tardiness objective. *European Journal of Operational Research*, *261*(3), 835–848. https://doi.org/10.1016/j.ejor.2017.03.030

Biskup, D. (2008). A state-of-the-art review on scheduling with learning effects. *European Journal of Operational Research*, *188*(2), 315–329.

BRIMBERG, J. (1996). A variable neighborhood algorithm for solving the continuous location-allocation problem. *Studies in Locational Analysis*, *10*, 1–12.

Chen, P., Wu, C. C., & Lee, W. C. (2006). A bi-criteria two-machine flowshop scheduling problem with a learning effect. Journal of the Operational Research Society, 57(9), 1113-1125.

Chou, F.-D., Wang, H.-M., & Chang, T.-Y. (2009). Algorithms for the single machine total weighted completion time scheduling problem with release times and sequence-dependent setups. *The International Journal of Advanced Manufacturing Technology*, *43*(7–8), 810. https://doi.org/10.1007/s00170-008-1762-4

Christofides, N., Mingozzi, A., & Toth, P. (1981). Exact algorithms for the vehicle routing problem, based on spanning tree and shortest path relaxations. *Mathematical Programming*, *20*(1), 255–282.

De Jong, J. R. (1957). The effects of increasing skill on cycle time and its consequences for time standards. Ergonomics, 1(1), 51-60.

Driessel, R., & Moench, L. (2009). Scheduling jobs on parallel machines with sequence-dependent setup times, precedence constraints, and ready times using variable neighborhood search. In *2009 International Conference on*

Computers Industrial Engineering (pp. 273–278). https://doi.org/10.1109/ICCIE.2009.5223515

Expósito-Izquierdo, C., Angel-Bello, F., Melián-Batista, B., Alvarez, A., & Báez, S. (2019). A metaheuristic algorithm and simulation to addressed the effect of learning or tiredness on sequence-dependent setup times in a parallel machine scheduling problem. *Expert Systems with Applications*, *117*, 62–74.

Fan, B., & Tang, G. (2006). A column generation for a parallel machine scheduling with sequence-dependent setup times. Tongji Daxue Xuebao/ Journal of Tongji University(Natural Science), 34(5), 680-683.

Feo, T. A., & Resende, M. G. C. (1989). A probabilistic heuristic for a computationally difficult set covering problem. *Operations Research Letters*, *8*(2), 67–71. https://doi.org/10.1016/0167-6377(89)90002-3

Feo, T. A., & Resende, M. G. C. (1995). Greedy Randomized Adaptive Search Procedures. *Journal of Global Optimization*, *6*(2), 109–133. https://doi.org/10.1007/BF01096763

Fowler, J., Horng, S. M., & Cochran, J. K. (2003). A hybridized genetic algorithm to solve parallel machine scheduling problems with sequence dependent setups. International Journal of Industrial Engineering: Theory Applications and Practice, 10(3), 232-243.

Gacias, B., Artigues, C., & Lopez, P. (2010). Parallel machine scheduling with precedence constraints and setup times. *Computers & Operations Research*, *37*(12), 2141–2151. https://doi.org/10.1016/j.cor.2010.03.003

Gavish, B., & Graves, S. C. (1978). The travelling salesman problem and related problems.

Godinho, M. T., Gouveia, L., & Magnanti, T. L. (2008). Combined route capacity and route length models for unit demand vehicle routing problems. *Discrete Optimization*, *5*(2), 350–372.

Graham, R. L., Lawler, E. L., Lenstra, J. K., & Kan, A. H. G. R. (1979). Optimization and Approximation in Deterministic Sequencing and Scheduling: a Survey. In P. L. Hammer, E. L. Johnson, & B. H. Korte (Eds.), *Annals of Discrete Mathematics* (Vol. 5, pp. 287–326). Elsevier. https://doi.org/10.1016/S0167-5060(08)70356-X

Guinet, A. (1991). Textile production systems: a succession of non-identical

parallel processor shops. Journal of the Operational Research Society, 42(8), 655-671.

Hansen, P., Mladenović, N., & Pérez, J. A. M. (2008). Variable neighbourhood search: methods and applications. *4OR*, *6*(4), 319–360. https://doi.org/10.1007/s10288-008-0089-1

Joo, C. M., & Kim, B. S. (2015). Hybrid genetic algorithms with dispatching rules for unrelated parallel machine scheduling with setup time and production availability. Computers & Industrial Engineering, 85, 102-109.

Kampke, E. H., Arroyo, J. E. C., & Santos, A. G. dos. (2009). Reactive GRASP with path relinking for solving parallel machines scheduling problem with resource-assignable sequence dependent setup times. In *2009 World Congress on Nature Biologically Inspired Computing (NaBIC)* (pp. 924–929). https://doi.org/10.1109/NABIC.2009.5393873

Koulamas, C., & Kyparisis, G. J. (2008). Single-machine scheduling problems with past-sequence-dependent setup times. *European Journal of Operational Research*, *187*(3), 1045–1049.

Lee, C. H., Liao, C. J., & Chao, C. W. (2014). Unrelated parallel machine scheduling with dedicated machines and common deadline. Computers & Industrial Engineering, 74, 161-168.

Lee, W. C. (2014). Single-machine scheduling with past-sequence-dependent setup times and general effects of deterioration and learning. Optimization Letters, 8(1), 135-144.

Miller, C. E., Tucker, A. W., & Zemlin, R. A. (1960). Integer Programming Formulation of Traveling Salesman Problems. *J. ACM*, *7*(4), 326–329. https://doi.org/10.1145/321043.321046

Mladenović, N., & Hansen, P. (1997). Variable neighborhood search. *Computers & Operations Research*, *24*(11), 1097–1100. https://doi.org/10.1016/S0305-0548(97)00031-2

Molina-Sánchez, L., & González-Neira, E. (2016). GRASP to minimize total weighted tardiness in a permutation flow shop environment. *International Journal of Industrial Engineering Computations*, *7*(1), 161–176.

Morales, M. F., Acosta, F. A. B., & Socarrás, A. A. (2016). An Iterated Greedy Algorithm for Minimizing the Total Completion Time in a Parallel Machine Scheduling Problem with Dependent Se. Retrieved June 28, 2018, from

http://scholar.googleusercontent.com/scholar?q=cache:EHMKYqLClMcJ:scholar.google.com/+Morales+et+al.+(2016)++scheduling&hl=es&as_sdt=0,5

Mustu, S., & Eren, T. (2018). The single machine scheduling problem with sequence-dependent setup times and a learning effect on processing times. *Applied Soft Computing*, *71*, 291–306. https://doi.org/10.1016/j.asoc.2018.06.051

Nessah, R., Chu, C., & Yalaoui, F. (2007). An exact method for Pm/sds,ri/∑i=1nCi problem. *Computers & Operations Research*, *34*(9), 2840–2848. https://doi.org/10.1016/j.cor.2005.10.017

Nucamendi-Guillén, S., Martínez-Salazar, I., Angel-Bello, F., & Moreno-Vega, J. M. (2016). A mixed integer formulation and an efficient metaheuristic procedure for the k-Travelling Repairmen Problem. *Journal of the Operational Research Society*, *67*(8), 1121–1134.

Pacheco, J., Ángel-Bello, F., & Álvarez, A. (2013). A multi-start tabu search method for a single-machine scheduling problem with periodic maintenance and sequence-dependent set-up times. *Journal of Scheduling*, *16*(6), 661–673. https://doi.org/10.1007/s10951-012-0280-2

Park, C., & Seo, J. (2013). A GRASP approach to transporter scheduling for ship assembly block operations management. *European Journal of Industrial Engineering*, *7*(3), 312–332. https://doi.org/10.1504/EJIE.2013.054133

Paula, D., Rocha, M., Ravetti, M. G., Mateus, G. R., & Pardalos, P. M. (2007). Solving parallel machines scheduling problems with sequence-dependent setup times using variable neighbourhood search. *IMA Journal of Management Mathematics*, *18*(2), 101–115. https://doi.org/10.1093/imaman/dpm016

Picard, J.-C., & Queyranne, M. (1978). The time-dependent traveling salesman problem and its application to the tardiness problem in one-machine scheduling. *Operations Research*, *26*(1), 86–110.

Pinedo, M. L. (2016). *Scheduling: Theory, Algorithms, and Systems*. Springer.

Resende, M. G., & Ribeiro, C. C. (2010). Grasp. Search Methodologies, 2nd ed., EK Burke and G. Kendall, Eds. Springer (to appear).

Ruiz, R., & Andres, C. (2007). Unrelated Parallel Machines Scheduling with Resource-Assignable Sequence Dependent Setup Times, 8.

Schiavinotto, T., & Stützle, T. (2004). The Linear Ordering Problem: Instances, Search Space Analysis and Algorithms. ***Journal of Mathematical Modelling and Algorithms*****, *3*(4), 367–402. https://doi.org/10.1023/B:JMMA.0000049426.06305.d8**

Su, L. H. (2009). Scheduling on identical parallel machines to minimize total completion time with deadline and machine eligibility constraints. The International Journal of Advanced Manufacturing Technology, 40(5-6), 572-581.

Tavakkoli-Moghaddam, R., Taheri, F., & Bazzazi, M. (2008). MULTI-OBJECTIVE UNRELATED PARALLEL MACHINES SCHEDULING WITH SEQUENCE-DEPENDENT SETUP TIMES AND PRECEDENCE CONSTRAINTS, 10.

Vallada, E., & Ruiz, R. (2011). A genetic algorithm for the unrelated parallel machine scheduling problem with sequence dependent setup times. ***European Journal of Operational Research*****, *211*(3), 612–622. https://doi.org/10.1016/j.ejor.2011.01.011**

Wang, J. B., Wang, M. Z., & Ji, P. (2012). Single machine total completion time minimization scheduling with a time-dependent learning effect and deteriorating jobs. International Journal of Systems Science, 43(5), 861-868.

Wang, X.-Y., & Wang, J.-J. (2013). Scheduling problems with past-sequence-dependent setup times and general effects of deterioration and learning. ***Applied Mathematical Modelling*****, *37*(7), 4905–4914. https://doi.org/10.1016/j.apm.2012.09.044**

Weng, M. X., Lu, J., & Ren, H. (2001). Unrelated parallel machine scheduling with setup consideration and a total weighted completion time objective. ***International Journal of Production Economics*****, *70*(3), 215–226. https://doi.org/10.1016/S0925-5273(00)00066-9**

Wu, C.-C., & Lee, W.-C. (2008). Single-machine group-scheduling problems with deteriorating setup times and job-processing times. ***International Journal of Production Economics*****, *115*(1), 128–133. https://doi.org/10.1016/j.ijpe.2008.05.004**

Xingong, Z., Yong, W., & Shikun, B. (2016). Single-machine group scheduling problems with deteriorating and learning effect. International Journal of Systems Science, 47(10), 2402-2410.

Yang, S.-J. (2011). Group scheduling problems with simultaneous considerations of learning and deterioration effects on a single-machine. *Applied Mathematical Modelling*, *35*(8), 4008–4016. https://doi.org/10.1016/j.apm.2011.02.024