Parallel machine scheduling problem with a single server

Abdelhak El Idrissi

Thèse de Doctorat en Cotutelle Internationale N°Ordre: 20/18

Pour obtenir le grade de :

Docteur de l'UNIVERSITÉ POLYTECHNIQUE HAUTS-DE-FRANCE

ET L'INSA HAUTS-DE-FRANCE

Discipline. Informatique

Et Docteur de l'ECOLE MOHAMMADIA D'INGENIEURS

Discipline. Mathématiques appliquées: Recherche opérationnelle et informatique

Présentée et soutenue par Abdelhak El idrissi.

Le 15/12/2020, à Rabat

Equipes de recherche, Laboratoires :
LAMIH – UMR CNRS 8201, France.
MOAD6, MASI, Maroc.

Ecoles doctorales :
École Doctorale Sciences Pour l'Ingénieur Université Lille Nord-de-France - 072.
CEDOC « Sciences de l'Ingénieurs » de l'EMI, Maroc.

Parallel machine scheduling problem with a single server:

The case of an arbitrary number of machines

Rapporteurs
- Chu, Feng. Professeur des Universités. Université d'Evry Val d'Essonne, France.
- Kacem, Imed. Professeur des Universités. Université de Lorraine, France.
- Lebbar, Maria. Professeur Habilité. Ecole Nationale Supérieure des Mines de Rabat, Maroc.

Examinateurs
- Dhaenens, Clarisse. Professeur des Universités, et Vice présidente recherche: Sciences et Technologies. Université de Lille, France.
- El Alami, Jamila. Professeur de l'Enseignement Supérieur, et Directrice du CNRST, Maroc. (*Présidente du jury*)
- El Hachemi, Nizar. Professeur Habilité. Ecole Mohammadia d'Ingénieurs, Maroc.

Directeurs de thèse
- Duvivier, David. Professeur des Universités. Université Polytechnique Hauts-de-France, France.
- Benbrahim, Mohammed. Professeur de l'Enseignement Supérieur. Ecole Mohammadia d'Ingénieurs, Maroc.

Invité: Benmansour, Rachid. Professeur Assistant. INSEA, Maroc.

Thèse de Doctorat en Cotutelle Internationale N°Ordre: 20/18

Pour obtenir le grade de :

Docteur de l'UNIVERSITÉ POLYTECHNIQUE HAUTS-DE-FRANCE

ET L'INSA HAUTS-DE-FRANCE

Discipline. Informatique

Et Docteur de l'ECOLE MOHAMMADIA D'INGENIEURS

Discipline. Mathématiques appliquées: Recherche opérationnelle et informatique

Présentée et soutenue par Abdelhak El idrissi.

Le 15/12/2020, à Rabat.

Equipes de recherche, Laboratoires :
LAMIH – UMR CNRS 8201, France.
MOAD6, MASI, Maroc.

Ecoles doctorales :
École Doctorale Sciences Pour l'Ingénieur Université Lille Nord-de-France - 072.
CEDOC « Sciences de l'Ingénieurs » de l'EMI, Maroc.

Ordonnancement sur des machines parallèles avec un serveur partagé: le cas d'un nombre arbitraire de machines

Rapporteurs
- Chu, Feng. Professeur des Universités. Université d'Evry Val d'Essonne, France.
- Kacem, Imed. Professeur des Universités. Université de Lorraine, France.
- Lebbar, Maria. Professeur Habilité. Ecole Nationale Supérieure des Mines de Rabat, Maroc.

Examinateurs
- Dhaenens, Clarisse. Professeur des Universités, et Vice présidente recherche: Sciences et Technologies. Université de Lille, France.
- El Alami, Jamila. Professeur de l'Enseignement Supérieur, et Directrice du CNRST, Maroc. (*Présidente du jury*)
- El Hachemi, Nizar. Professeur Habilité. Ecole Mohammadia d'Ingénieurs, Maroc.

Directeurs de thèse
- Duvivier, David. Professeur des Universités. Université Polytechnique Hauts-de-France, France.
- Benbrahim, Mohammed. Professeur de l'Enseignement Supérieur. Ecole Mohammadia d'Ingénieurs, Maroc.

Invité: Benmansour, Rachid. Professeur Assistant. INSEA, Maroc.

Abstract

Scheduling is a discipline of combinatorial optimization. It aims to allocate a limited set of resources over time to a set of jobs to be scheduled, with the aim of optimizing one or more criteria (project duration, advances/delays, availability of machines, etc.). Scheduling on parallel machines has been widely studied in the literature because of its practical and theoretical interest. Indeed, this type of configuration is well suited to model several real problems such as scheduling in operating rooms, queue management and scheduling of requests to processors, etc. However, a great number of works consider that the jobs to be scheduled are ready without prior preprocessing or setup. Therefore, under certain conditions, this assumption can lead to non optimal results, particularly if the processing time is substantial. This may result in a shortfall and/or waste of time. To overcome this, we are interested in the scheduling problem with a single server that performs these setup operations. The problem is studied in the literature only for specific cases, namely the case of two machines and the case of equal processing times.

In this thesis, we consider the problem of scheduling independent jobs on an arbitrary number of identical parallel machines with a single server. In this problem, each job requires a setup or a loading operation before its execution on a machine. This operation is performed by a single server which can be a robot or an operator. This problem has several industrial applications, namely: in semi-automatic production, container terminal and biomass logistics. Firstly, we consider the objective function of minimizing the total execution time (makespan), where all jobs are available at the beginning of the scheduling. Several mathematical models are presented for the general case of this problem, and a mathematical model is proposed for the regular case based on mathematical properties. Due to the complexity of the problem ($\mathcal{NP}$-hard), a lower bound, two greedy heuristics and a metaheuristic based on variable neighborhood search are proposed. Secondly, we propose a generalization of the problem, integrating the release and due dates for all the jobs. The objective function involves the minimization of the maximum lateness. In order to solve this new extension of the problem, we suggest a mathematical model

for solving small-sized instances. For large-sized instances of the problem, we design two metaheuristics based on a general variable neighborhood search and a hybrid of greedy randomized adaptive search procedure with variable neighborhood descent. Our approaches show their efficiency on benchmark instances existing in the literature.

Keywords: Scheduling - Parallel machines - Setup-time - Single server - Exact optimization methods - Metaheuristics

Résumé

L'ordonnancement est une discipline de l'optimisation combinatoire. Elle vise à allouer dans le temps un ensemble limité de ressources à des tâches à réaliser, avec pour objectif d'optimiser un ou plusieurs critères (durée du projet, avances/retards, disponibilité des machines, etc). L'ordonnancement sur machines parallèles a été beaucoup étudié dans la littérature du fait de son intérêt pratique et théorique. En effet, ce type de configuration est bien adapté pour modéliser plusieurs problèmes réels tels que l'ordonnancement dans les salles opératoires, la gestion des files d'attente et l'ordonnancement des requêtes aux processeurs, etc. Cependant, la plupart des travaux considèrent que les tâches à réaliser sont prêtes à être exécutées sans prétraitement préalable. Sous certaines conditions, cette hypothèse conduit à des résultats non optimaux, particulièrement si le temps de traitement est conséquent. Ceci peut se traduire par un manque à gagner et/ou par du temps perdu. Pour pallier à cela, nous nous intéressons au problème d'ordonnancement avec un serveur partagé qui réalise ces opérations de prétraitement. Le problème est étudié dans la littérature seulement pour des cas spécifiques à savoir le cas de deux machines et le cas des durées opératoires identiques.

Dans cette thèse, nous nous intéressons au problème d'ordonnancement de tâches sur un nombre arbitraire de machines parallèles et identiques avec un serveur partagé. Dans ce problème, chaque tâche nécessite une opération de chargement ou de prétraitement avant son exécution sur une machine. Cette opération est réalisée par un serveur unique qui peut être un robot ou un opérateur. Ce problème est présent dans plusieurs secteurs industriels, à savoir : en production semi-automatique, logistique portuaire et logistique de biomasse. Nous considérons dans un premier temps la fonction objectif de minimisation du temps total (makespan) avec des tâches disponibles au début de l'ordonnancement. Plusieurs modèles mathématiques sont présentés pour le cas général de ce problème et un modèle mathématique est proposé pour le cas régulier en se basant sur des propriétés mathématiques. En raison de la complexité du problème ($\mathcal{NP}$-difficile), une borne inférieure, deux heuristiques gloutonnes et une métaheuristique basée sur la recherche à

voisinage variable sont proposées. Dans un deuxième temps, nous proposons une généralisation du problème, intégrant les dates de disponibilité et d'échéance pour toutes les tâches. La fonction objectif étudiée est la minimisation du plus grand retard algébrique. Pour résoudre cette nouvelle extension du problème, nous suggérons un modèle mathématique pour la résolution des instances de petite taille. Afin de résoudre les instances de grande taille du problème, nous proposons deux métaheuristiques basées sur une recherche généralisée à voisinage variable et une procédure gloutonne de recherche adaptative. Nos approches montrent leur efficacité sur des jeux instances existant dans la littérature.

Mots-clefs : Ordonnancement - Machines parallèles - Temps de préparation - Serveur partagé - Méthodes d'optimisation exactes - Métaheuristiques

Contents

List of Figures

List of Tables

List of Algorithms

Introduction

"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 tasks over given time periods and its goal is to optimize one or more objectives." (Pinedo [75]). Scheduling problems may be classified to five main categories, namely: single machine scheduling, parallel machine scheduling, flow shop scheduling, job shop scheduling and open shop scheduling (Backer and Trietsch [11], Lohmer and Lasch [73], and Pinedo [75]). In flow shops, job shops and open shops, each task (job) has to undergo a series of operations. Those operations are the same for all jobs in flow shops, not necessarily the same for each job in job shops, and are open in open shops. Furthermore, in both single machine and parallel machines, jobs consist of one operation.

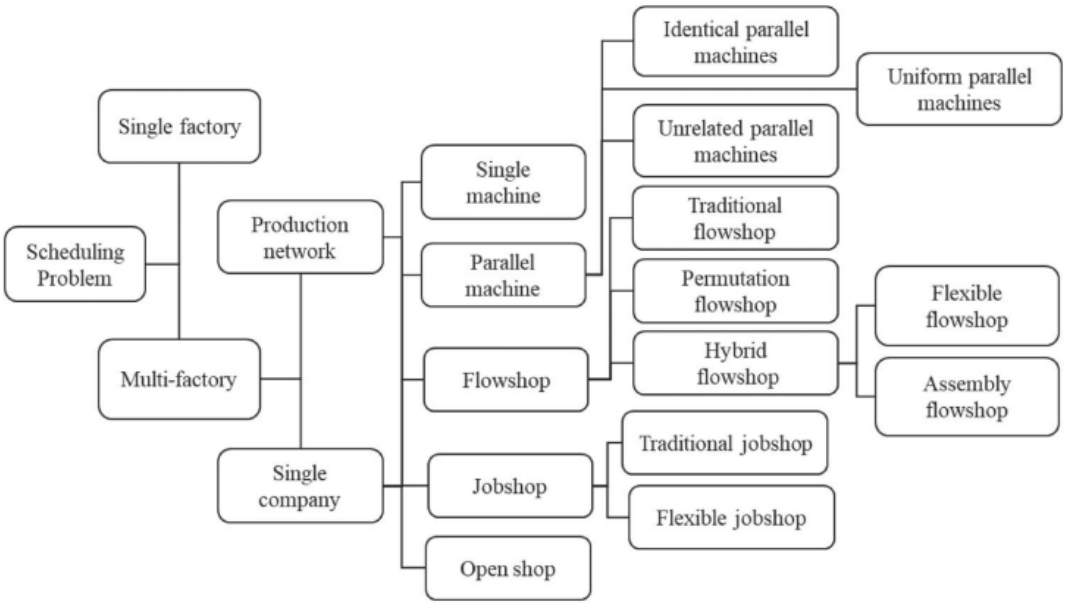

Figure 1: Classification of multi-factory scheduling problems [73]

Figure 1 illustrates different scheduling environments found in the recent literature.

The study of parallel machine scheduling problems is important from both a theoretical and a practical point of view. From a theoretical point of view it is a special case of the flexible flow shop, and a generalization of the single machine. Indeed, many properties of the parallel machine scheduling problem can be derived from the single machine scheduling problem. In addition, many optimization problems can be modeled as a parallel machine scheduling problem such as: traveling salesman problem and vehicle routing problem. From a practical point of view, resources in parallel can be found in many real world optimization problems (e.g., trucks scheduling in logistics, machines scheduling in production and crane-constrained vehicle scheduling in container terminal). Moreover, there are three main configurations of machines in parallel, namely: identical, unrelated or uniform (Brucker [24]). However, on identical parallel machines, the processing time of each job is the same for all machines. On unrelated parallel machines, the processing time of jobs depends on machines. While on uniform parallel machines, the processing time of each job is proportional for all machines.

Among the parallel machine scheduling problems, the parallel machine scheduling problem with a single server (PSS problem) has received much attention over the last two decades. In the PSS problem, the server is in charge of the setup operation of jobs. This setup operation can be defined as the time required to prepare the necessary resource (e.g., machines, people) to perform a task (e.g., job, operation) [5, 14, 47, 62]. Indeed, in the classical parallel machine scheduling problem, it is assumed that the jobs are to be executed without prior setup. However, this assumption is not always satisfied in practice where industrial systems are more flexible (e.g., flexible manufacturing system). Under certain conditions, this assumption can lead also to a shortfall and/or waste of time. It can be noticed that the vast majority of existing scheduling literature neglects this fact [5]. In addition, there are two types of setup times: sequence-independent and sequence-dependent. Sequence-independant which means that the setup time depends solely on the job to be processed, regardless of its preceding job. On the other hand, in the sequence-dependent type, setup time depends on both the job and its preceding job. In some environments, there may be different families or batches of jobs which involve setup times among the jobs within a family and setup times among the jobs families [18]. The family setup time can be also sequence-independent or sequence-dependent.

The PSS problem has many industrial applications. In network computing, the network server sets up the workstations by loading the required software. In production applications, the setting up of machines involves simultaneous use of a common resource which might be a robot or a human operator attending each setup [14]. In automated material handling systems, robotic cells or in the semiconductor industry [62] it is necessary to

share a common server, for example a robot, by a number of machines to carry out machine setups. Then job processing is executed automatically and independently by the individual machines.

In this thesis, the PSS problem is studied. In addition, setup time is considered as sequence-independent. Following the standard $\alpha|\beta|\gamma$ classification scheme for scheduling problems known as the Graham triplet (Graham et al. [43]), the characteristics of the PSS problem are defined in Table 1. In this notation, α represents the machines environment and the number of servers. For example, $(P, S1)$ corresponds to identical parallel machines with one server in charge of jobs setup. β represents the server and jobs types. For example, (r_j, M_j) corresponds to scheduling jobs that are released over time with machine eligibility restriction. Finally, γ describes the objective in the PSS problem. As example, $P, S1|p_j, s_j|C_{max}$ represents the parallel machine scheduling problem with a single server to minimize the makespan. This thesis expects to contribute in the literature of scheduling optimization. The major contributions are:

- First contribution is about the problem $P, S1|p_j, s_j|C_{max}$. We propose new approaches for its exact resolution.
- Second contribution is about the approximate resolution of the problem $P, S1|p_j, s_j|C_{max}$ by greedy heuristics and Variable Neighborhood Search (VNS) metaheuristic.
- Third contribution is about the exact and approximate resolution of the problem $P, S1|r_j|L_{max}$.

This thesis is structured in four main chapters. Chapter 1 surveys several papers on the parallel machine scheduling problem involving a single server and its variants that have appeared since 1996. The review is focused on the static and dynamic version of the problem. In the static case, all the information about the problem (i.e., number of jobs, number of machines, processing times and setup times) is available before the scheduling starts. While, in the dynamic case, all jobs are presented one by one according to an input sequence. Additionally, we survey some variants of the PSS problem, namely: scheduling with time restriction, scheduling reentrant jobs, scheduling with a single server and loading/unloading operations and scheduling with multiple servers. The contributions for solving the problem $P, S1|p_j, s_j|C_{max}$ by mixed integer programming formulations are presented in Chapter 2. First, several formulations for the general case of the problem are presented. We also propose valid inequalities to improve those formulations. Then, a

Table 1: The abbreviations used in triple notations $\alpha|\beta|\gamma$.

	Notation	Explanation
α	P	identical parallel machines scheduling
	PD	dedicated parallel machines scheduling
	R	unrelated parallel machines scheduling
	$S1$	Scheduling with a single server
	Sk	Scheduling with k servers
β	p_j	Processing times
	s_j	Setup times
	r_j	Release dates
	d_j	Due dates
	M_j	Machine eligibility restrictions
	$prmp$	Scheduling with preemption
	ST_{sd}	Sequence-dependent setup time
	$fixed - seq$	Job processing sequences are fixed
γ	C_{max}	Makespan
	L_{max}	Maximum lateness
	IT	Total machine idle time
	$\sum C_j$	Total completion time
	$\sum w_j C_j$	Total weighted completion time
	$\sum T_j$	Total tardiness
	$\sum w_j T_j$	Total weighted tardiness
	$\sum U_j$	Number of tardy jobs
	$\sum w_j U_j$	Weighted number of tardy jobs

theoretical study of the regular case of the problem and a mathematical formulation for its exact resolution are suggested. Thereafter, extensive computational experiments are reported and the performance of the mathematical formulations is discussed. On the experimental results, one of our proposed formulations turned out to be the new state-of-the art formulation. Chapter 3 describes our approximate approaches for solving the problem $P, S1|p_j, s_j|C_{max}$. First, two greedy heuristics with a complexity of $O(n^2)$ are proposed to minimize respectively the server waiting time and the machine idle time. These heuristics generalize those proposed in [3, 55]. Then, we present a basic VNS with three neighborhood structures. The obtained results, on benchmark instances from the literature, show the effectiveness and efficiency of the proposed approaches compared with some existing algorithms in the literature. In the last chapter, we address the problem $P, S1|r_j|L_{max}$. In the literature, only a limited number of works considered it. Among them, Hall et al. [44] showed that the problem $P2, S1|p_j, s_j = 1|L_{max}$ is unary $\mathcal{NP}$-hard. Note that the Early Due Date rule can solve optimally the problem $P, S1|p_j = 1, s_j|L_{max}$ in $O(n \log n)$. In ad-

dition, Brucker et al. [25] showed that the problem $P2, S1|r_j = 1|L_{max}$ is unary $\mathcal{NP}$-hard. To solve the problem $P, S1|r_j|L_{max}$, we present a mixed integer programming formulation based on network variables, a constructive heuristic and two metaheuristics based on General Variable Neighborhood Search (GVNS) and Greedy Randomized Adaptive Search Procedure (GRASP) with Variable Neighborhood Descent (VND). The computational results on a set of randomly generated instances showed that GVNS and GRASP outperformed the proposed methods. Finally, the main results of this thesis are summarized at the end along with new perspectives on the future work to be carried out.

Note that all publications (i.e., international conferences with selection committee and international journal) relating to this thesis are available via the webpage https://www.uphf.fr/LAMIH/en/membre?id=elidrissi_abdelhak.

Chapter 1

The parallel machine scheduling problem with a single server: a survey

Contents

1.1 Introduction

The objective of this chapter is to draw a succinct picture of current research in the field of the PSS problem. We discuss within a structured framework the main problem

attributes and research directions in the field of parallel machine scheduling with a single server as of 2020, pinpointing current industrial applications, complexity results and some variants. The remainder of this chapter is organized as follows: Industrial applications of the studied problem are discussed in section 1.2. Some complexity results are presented in section 1.3. In section 1.4 the literature on the static version of the problem (i.e., the situation where the information about the problem is available before the scheduling starts) is reviewed, while in section 1.5, the literature on the dynamic version of the problem (i.e., the situation where the jobs are presented one by one according to an input sequence) is reviewed. Some variants are discussed in section 1.6. Finally, section 1.7 ends the chapter with conclusions and some perspectives.

1.2 Industrial applications

As mentioned below, the PSS problem has many industrial applications ...

In supply chain optimization. Torjai and Kruzslicz [98] studied a biomass truck scheduling problem that originated in a real-life herbaceous biomass supply chain. The authors considered it as an identical PSS problem. Two objective functions were studied, namely: the minimization of the number of machines and minimization of the idle time of machines. The authors stated that the identical trucks represent the identical parallel machines in charge of delivering biomass from satellite storage locations to a central biorefinery operating a single unloader (the single server). They considered two assumptions regarding the server. The unload operation of the server has a unit time length for each trip and idle periods are not allowed for it.

In container terminals. Bish [20] showed that the multiple-crane-constrained vehicle scheduling and location problem is similar to the PSS problem, where crane loading/unloading operations represent the setup times, crane plays the role of the server, each container corresponds to a job, and vehicles correspond to machines. The objective is to minimize the maximum turnaround time of a ship, which can represent the maximum lateness.

In the plastic injection industry. Bektur and Saraç [14] considered a set of plastic injection machines, which involve a team that must work together to set up (clean, prepare, etc.) orders on machines. The team is considered as a single server. The authors considered it as an unrelated PSS problem. The objective function involved the minimization of the total weighted tardiness.

In the printing industry. Huang et al. [56] considered a set of printing machines who must be set up by a team before printing orders on machines. The authors stated that the setup times are sequence dependent. They considered the problem as a dedicated PSS problem. The objective function involved the minimization of the makespan.

In robotic cells or in the semiconductor industry. It is necessary to share a single server (or robot), by a number of machines to carry out machine setups. Then job processing is executed automatically and independently by the individual machines. (see Kim and Lee [62]).

For more details about industrial applications, we direct the reader to consult the thesis of Schnitzler Daniel [85].

1.3 Complexity results

In this section, we present some complexity results of the PSS problem with different objective functions. The first part is dedicated to the complexity results for the case of two machines, and, the second part is dedicated to the complexity results for the case of m machines.

1.3.1 Complexity results for two machines

For the case of two machines, Kravenchnko and Werner [67] presented complexity results for two objective functions, namely: makespan and total machine idle time. They showed that the problems $P2, S1|p_j, s_j = 1|C_{max}$ and $P2, S1|p_j, s_j = s|IT$ are $\mathcal{NP}$-hard. On the other hand, they showed that the problems $P2, S1|p_j, s_j = 1|C_{max}$, $P2, S1|p_j > s, s_j = s|IT$, $P2, S1|p_j < s, s_j = s|IT$ and $P2, S1|p_j, s_j = 1|IT$ are polynomial solvable. Brucker et al. [25] studied the PSS problem, where four objective functions were examined, namely: makespan, maximum lateness, total completion time and total weighted completion time. They showed that the problems

- $P2, S1|p_j, s_j = 1|C_{max}$
- $P2, S1|p_j = p, s_j|C_{max}$
- $P2, S1|p_j = 1, r_j, s_j|L_{max}$
- $P2, S1|p_j = p, s_j|\sum C_j$

- $P2, S1|p_j, r_j, s_j = 1|\sum C_j$
- $P2, S1|p_j = 1, r_j, s_j|\sum C_j$
- $P2, S1|p_j, s_j = 1|\sum w_j C_j$

are $\mathcal{NP}$-hard. In addition, Hall et al. [44] showed that the problems

- $P2, S1|p_j, s_j = 1|C_{max}$
- $P2, S1|p_j, s_j = s|C_{max}$
- $P2, S1|p_j, s_j = 1|L_{max}$
- $P2, S1|p_j, s_j = s|\sum C_j$
- $P2, S1|p_j, s_j = 1|\sum w_j C_j$
- $P2, S1|p_j, s_j = 1|\sum U_j$
- $P2, S1|p_j = 1, s_j|\sum wjU_j$
- $P2, S1|p_j = 1, s_j|\sum T_j$
- $P2, S1|s_j = 1|\sum T_j$
- $P2, S1|p_j = 1, s_j|\sum w_j T_j$

are $\mathcal{NP}$-hard. For the case of dedicated parallel machines, Glass et al. [42] showed that the problems with equal processing times and equal setup times are $\mathcal{NP}$-hard. An overview of these results is given in Table 1.1.

1.3.2 Complexity results for m machines

For the case of an arbitrary number of machines, Brucker et al. [25] and Hall et al. [44] presented complexity results for many objective functions of the PSS problem, namely: C_{max}, L_{max}, $\sum w_j C_j$, $\sum C_j$, $\sum w_j T_j$, $\sum T_j$, $\sum w_j U_j$ and $\sum U_j$. The problems

- $P, S1|p_j, s_j = 1|C_{max}$
- $P, S1|p_j, s_j = 1|\sum C_j$

Table 1.1: Complexity results for two parallel machines scheduling problem with a single server.

Problem	Complexity	Reference
$P2, S1 \mid p_j, s_j = 1 \mid C_{max}$	$\mathcal{NP}$-hard	[25, 44, 67]
$P2, S1 \mid p_j = p, s_j \mid C_{max}$	$\mathcal{NP}$-hard	[25]
$P2, S1 \mid p_j, s_j = s \mid C_{max}$	$\mathcal{NP}$-hard	[44]
$P2, S1 \mid p_j, s_j = 1 \mid C_{max}$	Polynomial Solvable	[67]
$PD2, S1 \mid \alpha_j = \beta_k = \alpha \mid C_{max}$ [1]	$\mathcal{NP}$-hard	[42]
$PD2, S1 \mid a_j = b_k = a \mid C_{max}$ [2]	$\mathcal{NP}$-hard	[42]
$P2, S1 \mid p_j, s_j = s \mid IT$	$\mathcal{NP}$-hard	[67]
$P2, S1 \mid p_j > s, s_j = s \mid IT$	Polynomial Solvable	[67]
$P2, S1 \mid p_j < s, s_j = s \mid IT$	Polynomial Solvable	[67]
$P2, S1 \mid p_j, s_j = 1 \mid IT$	Polynomial Solvable	[67]
$P2, S1 \mid p_j = 1, r_j, s_j \mid L_{max}$	$\mathcal{NP}$-hard	[25]
$P2, S1 \mid p_j, s_j = 1 \mid L_{max}$	$\mathcal{NP}$-hard	[44]
$P2, S1 \mid p_j = p, s_j \mid \sum C_j$	$\mathcal{NP}$-hard	[25]
$P2, S1 \mid p_j, r_j, s_j = 1 \mid \sum C_j$	$\mathcal{NP}$-hard	[25]
$P2, S1 \mid p_j = 1, r_j, s_j \mid \sum C_j$	$\mathcal{NP}$-hard	[25]
$P2, S1 \mid p_j, s_j = s \mid \sum C_j$	$\mathcal{NP}$-hard	[44]
$P2, S1 \mid p_j, s_j = 1 \mid \sum C_j$	Polynomial solvable	[44]
$P2, S1 \mid p_j, s_j = 1 \mid \sum w_j C_j$	$\mathcal{NP}$-hard	[25, 44]
$P2, S1 \mid p_j, s_j = 1 \mid \sum U_j$	$\mathcal{NP}$-hard	[44]
$P2, S1 \mid p_j = 1, s_j \mid \sum wjU_j$	$\mathcal{NP}$-hard	[44]
$P2, S1 \mid p_j = 1, s_j \mid \sum T_j$	$\mathcal{NP}$-hard	[44]
$P2, S1 \mid s_j = 1 \mid \sum T_j$	$\mathcal{NP}$-hard	[44]
$P2, S1 \mid p_j = 1, s_j \mid \sum w_j T_j$	$\mathcal{NP}$-hard	[44]

[1] $\alpha_j = \beta_k = \alpha$ means that all jobs have the same setup time in all machines.

[2] $a_j = b_k = a$ means that all jobs have the same processing time in all machines.

- $P, S1|p_j, s_j = 1|\sum w_j C_j$
- $P, S1|p_j = 1, s_j|\sum w_j T_j$

are $\mathcal{NP}$-hard. On the other hand, the problems

- $P, S1|p_j = p, r_j, s_j = s|C_{max}$
- $P, S1|p_j = 1, s_j|L_{max}$
- $P, S1|p_j \geq m-2, s_j = 1|\sum C_j$
- $P, S1|p_j = p, r_j, s_j = s|\sum C_j$
- $P, S1|p_j = 1, s_j|\sum w_j C_j$
- $P, S1|p_j = 1, r_j, s_j = s|\sum w_j C_j$
- $P, S1|p_j = 1, r_j, s_j = s|\sum w_j U_j$
- $P, S1|p_j = p, s_j = s|\sum w_j U_j$
- $P, S1|p_j = 1, r_j, s_j = s|\sum T_j$
- $P, S1|p_j = 1, s_j = s|\sum T_j$
- $P, S1|p_j = 1, s_j = s|\sum w_j T_j$
- $P, S1|p_j = 1, r_j, s_j = 1|\sum w_j T_j$
- $P, S1|p_j = p, s_j = s|\sum w_j T_j$

are polynomial solvable. While, the complexity of the problems $P, S1|p_j = 1, r_j, s_j|C_{max}$, $Pm, S1|p_j = p, r_j, s_j = s|\sum w_j U_j$ and $Pm, S1|p_j = p, r_j, s_j = s|\sum w_j T_j$ remains open. An overview of these results is given in Table 1.2.

Table 1.2: Complexity results for an arbitrary number of parallel machines scheduling problem with a single server.

Problem	Complexity	Reference
$P, S1 \mid p_j, s_j = 1 \mid C_{max}$	$\mathcal{NP}$-hard	[25, 67]
$P, S1 \mid p_j = p, r_j, s_j = s \mid C_{max}$	Polynomial solvable	[25]
$P, S1 \mid p_j = 1, r_j, s_j \mid C_{max}$	Open problem	[25]
$P, S1 \mid p_j = 1, s_j \mid L_{max}$	Polynomial solvable	[44]
$P, S1 \mid p_j, s_j = 1 \mid \sum C_j$	$\mathcal{NP}$-hard	[25]
$P, S1 \mid p_j \geq m-2, s_j = 1 \mid \sum C_j$	Polynomial solvable	[25]
$P, S1 \mid p_j = p, r_j, s_j = s \mid \sum C_j$	Polynomial solvable	[25]
$P, S1 \mid p_j = 1, s_j \mid \sum C_j$	Polynomial solvable	[44]
$P, S1 \mid p_j, s_j = 1 \mid \sum w_j C_j$	$\mathcal{NP}$-hard	[25]
$P, S1 \mid p_j = 1, s_j \mid \sum w_j C_j$	Polynomial solvable	[44]
$P, S1 \mid p_j = 1, r_j, s_j = s \mid \sum w_j C_j$	Polynomial solvable	[25]
$P, S1 \mid p_j = 1, s_j \mid \sum U_j$	Polynomial solvable	[44]
$P, S1 \mid p_j = 1, r_j, s_j = s \mid \sum w_j U_j$	Polynomial solvable	[25]
$P, S1 \mid p_j = p, s_j = s \mid \sum w_j U_j$	Polynomial solvable	[25]
$Pm, S1 \mid p_j = p, r_j, s_j = s \mid \sum w_j U_j$	Open problem	[25]
$P, S1 \mid p_j = 1, r_j, s_j = s \mid \sum T_j$	Polynomial solvable	[25]
$P, S1 \mid p_j = 1, s_j = s \mid \sum T_j$	Polynomial solvable	[44]
$P, S1 \mid p_j = 1, s_j \mid \sum w_j T_j$	$\mathcal{NP}$-hard	[44]
$P, S1 \mid p_j = 1, s_j = s \mid \sum w_j T_j$	Polynomial solvable	[44]
$P, S1 \mid p_j = 1, r_j, s_j = 1 \mid \sum w_j T_j$	Polynomial solvable	[25]
$P, S1 \mid p_j = p, s_j = s \mid \sum w_j T_j$	Polynomial solvable	[25]
$Pm, S1 \mid p_j = p, r_j, s_j = s \mid \sum w_j T_j$	Open problem	[25]

Table 1.3: An overview of the approaches in the literature on static identical parallel machines scheduling problem with a single server.

Publications	Jobs			Setup type		Number of machine		Objective	Approach (comments)
	r_j	d_j	$pmtn$	Dependent	Independent	$m=2$	$m\geq 2$		
[64]					✓	✓		IT	Complexity results and a beam search heuristic
[67]					✓	✓	✓	C_{max} and IT	Complexity results, polynomially solvable cases and heuristics
[68]					✓	✓	✓	C_{max} and IT	Complexity results for the case of one server and $m-1$ servers
[44]					✓	✓	✓	C_{max}, L_{max}, etc,.	Complexity results, polynomial time algorithms for: $C_{max}, L_{max}, \sum w_jC_j, \sum C_j, \sum w_jT_j, \sum T_j, \sum w_jU_j, \sum U_j$
[69]					✓		✓	$\sum C_j$	Heuristic algorithm
[104]					✓		✓	$\sum w_jC_j$	An approximation algorithm
[25]	✓	✓			✓	✓	✓	C_{max}, L_{max}, etc,.	New complexity results for: $C_{max}, L_{max}, \sum w_jC_j, \sum C_j, \sum w_jT_j, \sum T_j, \sum w_jU_j, \sum U_j$
[1]					✓	✓		C_{max}	Complexity results, MIP model (reglar case)
[2]					✓	✓		C_{max}	Complexity results, lower bound and heuristics (equal processing times)
[3]					✓	✓		C_{max}	Greedy heuristics and GA algorithm (general case)
[62]					✓		✓	C_{max}	MIP models and hybrid SA & TS
[41]					✓	✓		C_{max}	MIP models and variants of a B&P
[58]			✓		✓	✓		C_{max}	An algorithm with a tight worst case ratio
[50]					✓	✓		C_{max}	MIP model based on blocks
[51]					✓	✓		$\sum C_j$	Hybrids HR and SA
[52]					✓	✓		C_{max}	SA and GA algorithms
[**?**]					✓		✓	$\sum w_jC_j$	An approximation algorithm
[54]					✓	✓		IT	MIP model and TS algorihtm
[55]					✓	✓		C_{max}	Two greedy heuristics
[114]					✓		✓	$\sum C_j$	An approximate algorithm
[8]					✓	✓		C_{max}	B&B, ACO algorithm
[47]				✓			✓	C_{max}	MIP model, SA and GA algorithms
[28]			✓		✓	✓	✓	C_{max}	Complexity results, and a pseudo-polynomial time algorithm
[38]					✓		✓	C_{max}	Two MIP models
[36]					✓		✓	C_{max}	Two greedy heuristics
[16]					✓	✓		C_{max}	Equivalence with scheduling with time restriction (equal processing time)
[72]					✓		✓	$\sum w_jC_j$	B&B, lower bound
[4]					✓	✓		C_{max}	TS and PSO algorithms
[88]				✓			✓	C_{max}	MIP model, iterative local search

1.4 Static parallel machine scheduling problem with a single server

This section describes the literature on static parallel machine scheduling problem with a single server, a summary of which is provided in Table 1.3. We mean by static (or off-line), the situation where the information about the problem is available before the scheduling starts. This literature can be classified in two main categories, namely: the case of two machines and the case of an arbitrary number of machines.

1.4.1 Case of two machines

For this case, three objective functions have been considered, namely: makespan minimization (C_{max}), total machine idle time minimization (IT) and total completion time minimization ($\sum C_j$). In Koulamas [64], the authors showed that the problem $P2, S1|p_j, s_j|IT$ with the objective of minimizing machines idle time (IT) is $\mathcal{NP}$-hard in the strong sense and proposed a beam search algorithm to solve it. Jiang et al. [58] considered the problem $P2, S1|pmtn|C_{max}$. They proposed an algorithm with tight bound of 4/3 and showed that it can generate optimal schedules for two special cases: equal setup times and equal processing times. Abdekhodaee and Wirth [1] addressed the problem $P2, S1|p_j, s_j|C_{max}$. They proposed a MIP formulation for the regular case of the problem and two greedy heuristics for the general case of the problem. Abdekhodaee et al. [2] investigated the problem $P2, S1|p_j = p, s_j = s|C_{max}$ with equal processing times and equal setup times. They showed that the problem is $\mathcal{NP}$-hard and proposed two effective heuristics. [3] extended their previous studies in [1, 2]. They proposed two greedy heuristics, a genetic algorithm and the Gilmore-Gomory algorithm for the general case of the problem $P2, S1|p_j, s_j|C_{max}$. Later, [41] addressed the problem $P2, S1|p_j, s_j|C_{max}$. They presented two MIP formulations and two variants of a branch-and-price scheme. [50] proposed a MIP formulation based on the idea for the problem $P2, S1|p_j, s_j|C_{max}$, based on the idea of decomposing a schedule into a set of blocks. The computational results showed that the proposed formulation outperformed all heuristics in [41]. Hasani et al. [52] addressed the problem $P2, S1|p_j, s_j|IT$, which is closely related to the problem $P2, S1|p_j, s_j|C_{max}$. They proposed a MIP formulation and a tabu search (TS) algorithm. Hasani et al. [51] proposed an improved harmony search and a simulated annealing (SA) algorithms for the problem $P2, S1|p_j, s_j|\sum C_i$. In another study, Hasani et al. [54] addressed the problem $P2, S1|p_j, s_j|C_{max}$. They proposed two metaheuristics based on SA and genetic algorithm (GA). The results obtained are much better than all the previous algorithms proposed

in [3, 41]. Hasani et al. [55] investigated the problem $P2, S1|p_j, s_j|C_{max}$. They proposed two greedy heuristics to solve very large instances with up to 10000 jobs. The results obtained are much better than all previous algorithms presented in the literature for only very large-sized instances. Later, Arnaout [8] suggested a branch & bound and an ant colony optimization (ACO) for the problem $P2, S1|p_j, s_j|C_{max}$. The results obtained are much better than the algorithms proposed by [54] for large-sized instances. Benmansour et al. [16] addressed the problem $P2, S1|p_j = p, s_j|C_{max}$ with equal processing times. They showed that the problem is equivalent to the single machine scheduling problem with time restriction (STR). The STR is a new scheduling problem that was firstly studied by [15, 22]. Recently, Alharkan et al. [4] proposed two metaheuristics based on TS and geometric particle swarm optimization algorithms for the problem $P2, S1|p_j, s_j|C_{max}$. The computational results showed that the proposed metaheuristics outperformed the algorithms in [54, 55] for large-sized instances.

1.4.2 Case of an arbitrary number of machines

In this case three machine environments have been investigated, namely: identical parallel machines, dedicated parallel machines and unrelated parallel machines. In addition, only three objective functions have been considered, namely: makespan minimization (C_{max}), the total weighted completion time minimization ($\sum w_j C_j$) and the total weighted tardiness minimization ($\sum w_j T_j$).

Identical parallel machines. Kravchenko and Werner [69] suggested a heuristic algorithm to solve the problem $P, S1|p_j, s_j|\sum C_j$. Wang and Cheng [104] proposed an approximation algorithm for the problem $P, S1|p_j, s_j|\sum w_j C_j$. Kim and Lee [62] addressed the problem $P, S1|p_j, s_j|C_{max}$ and they suggested two MIP models and a hybrid heuristic algorithm. Hasani et al. [53] proposed an approximation algorithm with a worst-case ratio of $3 - \frac{1}{m}$ for the problem $P, S1|p_j, s_j|\sum w_j C_j$. Zhang et al. [114] proved that the SPT algorithm have a worst case ratio of $1 + \frac{\sqrt{m-1}}{\sqrt{m}\sqrt{m-1}}$ for the problem $P, S1|p_j = p, s_j|\sum C_j$ with equal processing times. Hamzadayi and Yildiz [47] considered the problem $Pm, S1|ST_{sd}|C_{max}$ with sequence dependent setup time. They proposed a MIP model and two metaheuristics based on SA and GA. Cheng et al. [28] considered the problems $P2, S1|pmtn|C_{max}$ and $P, S1|pmtn|C_{max}$, by taking into account job's preemption. They showed that the problems are $\mathcal{NP}$-hard and presented pseudo-polynomial time algorithms to solve them. El idrissi [36], two greedy heuristics were presented to solve the $P, S1|p_j, s_j|C_{max}$. The presented heuristics generalize those proposed by [3] and [55]. Recently, Liu et al. [72] presented a branch and bound algorithm, a lower bound, and dom-

inance properties for the problem $P, S1|p_j, s_j|\sum w_j C_j$. Silva et al. [88] proposed a MIP formulation and a iterative local search metaheuristic for the problem $P, S1|ST_{sd}|C_{max}$. The computational results showed that the MIP formulation and the iterative local search outperformed the methods presented by [47]. Elidrissi et al. [37] proposed a metaheuristic based on variable neighborhood search (VNS) for the problem $P, S1|p_j, s_j|C_{max}$. The computational results showed that the proposed VNS outperformed the algorithms suggested by [3, 36, 55] in terms of deviation from the lower bound. Recently, Elidrissi et al. [39] proposed MIP formulations based on different decision variables for solving general and regular job sets of the problem $P, S1|p_j, s_j|C_{max}$. The results showed that two of the proposed formulations outperformed the existing formulations in the literature.

Dedicated parallel machines. Huang et al. [56] proposed a MIP formulation and hybrid genetic algorithm for the problem $PD, S1|ST_{sd}|C_{max}$. Recently, Cheng et al. [29] were the first to investigate the problem $PD, S1|fixed - seq|C_{max}$ in which the job processing sequences are fixed. They showed that the problem is $\mathcal{NP}$-hard even if all the jobs have the same processing length or all the loading operations require a unit time and proposed two heuristic algorithms to deal with the case where all the loading times are unit (ie., $\forall j \quad s_j = 1$).

Unrelated parallel machines. Bektur and saraç [14] were the first to address the problem $R, S1|Mj, ST_{sd}|\sum w_j T_j$. They presented a MIP formulation and two metaheuristics based on TS and SA by taking into consideration machine eligibility restrictions and sequence dependent setup time.

1.5 Dynamic parallel machine scheduling problem with a single server

In this section, we review the related works for the dynamic version of the PSS problem, where the jobs are presented one by one according to an input sequence. This review reveals that only four papers handle this dynamic version. In addition, the majority of studies assumed that the machines are identical and considered only the objective of minimizing the makespan. The first published paper dealing with the dynamic parallel machine scheduling problem with a single server is proposed by Zhang and Wirth [115]. The authors considered the problem of on-line one-by-one scheduling of $P2, S1|p_j, s_j|C_{max}$. They proposed a lower bound for the equal length job case (i.e., $\forall j \quad s_j + p_j = a$), and two heuristics with tight asymptotic competitive ratios for the equal processing times and reg-

ular equal setup times cases. In 2011, Su [92] addressed the problem $P2, S|online, r_j|C_{max}$, where the jobs are completely unknown until their release times. The author showed that the online LPT algorithm has a competitive ratio of 2, and has a lower bound equal to $\frac{\sqrt{5}+1}{2}$. In addition, he showed that for the case with unit setup times, the online LPT algorithm has a competitive ratio of 1.5, and no online algorithm can has a competitive ratio less than $\sqrt{2}$. Later, Hamzadayi and Yildiz [45] addressed the dynamic parallel machine scheduling problem with a single server and sequence dependent setup times, in order to minimize the makespan. The authors considered an arbitrary number of identical parallel machines. To solve the problem, they suggested a simulated annealing and dispatching rule based complete rescheduling approaches, as well as a simulation optimization tools. In an other study, Hamzadayi and Yildiz [46] proposed a simulated annealing and a dispatching rules-based complete rescheduling approach for the dynamic version of the parallel machine scheduling problem with a single server, with job preemption and sequence dependent setup times, in order to minimize the number of tardy jobs as primary goal and the square root of the mean-squared deviation for due dates as secondary goal.

1.6 Variants

In this section, we review some variants of the parallel machine scheduling problem with a single server, namely: scheduling with time restriction, scheduling rentrant jobs and a single server, scheduling with a single server and loading/unloading operations and scheduling with multiple servers.

Scheduling with time restriction

The single-processor scheduling problem with time restrictions (STR) is a new scheduling problem that was first studied in [15, 22, 108]. In this problem a set of independent jobs has to be processed on a single processor, subject only to the following constraint: During any time period of length α, the number of jobs being executed is less than or equal to a given integer value B, (i.e., $\forall x \geq 0$, the unit interval $[x, x + \alpha)$ can intersect at most B jobs). Benmansour et al. [16] showed that the $P2, S1|p_j = p, s_j|C_{max}$ with equal processing times is equivalent to the STR problem. They proposed two mixed integer programming formulations based on assignement and positional date variables (APV) and time-indexed variables (TIV) to solve the problem. Computational experiments showed that APV outperforms TIF and worked well for up to $n = 500$ jobs.

Scheduling reentrant jobs and a single server

Chakhlevitch and Glass [27] are the first to address the static parallel machine scheduling problem with a single server and reentrant jobs constraint. The authors considered that each job consists of three operations: (i) initialization, that can be done by any machine and requires a small amount of time; (ii) setup operation, performed by a single server; and (iii) the main processing operation, completed by the same machine which has initialized the job (implying the reentrance). They showed that the problem is $\mathcal{NP}$-hard in the ordinary sense, and proposed a heuristic algorithm tested on randomly generated benchmark data. It is motivated by a real-life problem from a microbiology laboratory. Later, Wang et al. [105] considered the static parallel machine scheduling problem with a single server and reentrant jobs constraint. The authors assumed that the order of jobs is known in advance. They proposed a GA to solve this problem. The computational results showed that GA outperformed the methods suggested by [27].

Scheduling with a single server and loading/unloading operations

For this variant of the problem, it is assumed that each job has to be loaded by the server before being processed on one of the machines, and unloaded immediately by the same server after its processing. It can be noticed that only two papers are proposed in the literature for this variant of the problem. Xie et al. [112] addressed a static parallel machine scheduling problem with a single server in charge of loading and unloading jobs on machines. The authors presented complexity results for some particular cases of the problem. They derived some optimal properties that helped to prove that LPT heuristic generated a tight worst-case bound of $3/2 - 1/2m$. Later, Jiang et al. [59] addressed a dynamic scheduling problem on two identical parallel machines with a single server. The authors assumed that both the loading and unloading operations of jobs that are performed by the single server take one unit time. To solve the problem, they proposed two online algorithms that have tight competitive ratio.

Scheduling with multiple servers

For the parallel machine scheduling problem involving multiple servers in charge of the setup operation of jobs, only a limited number of works exist in the literature. Kravchenko and Werner [68] addressed the problem $Pm, Sk|p_j, s_j|C_{max}$ with k servers, where $k > 1$. They showed that the problem is unary $\mathcal{NP}$-hard for each $k < m$ and

developed a pseudo-polynomial time algorithm. Later, Werner and Kravchenko [107] generalized and extended some results from Hall et al. [44]. The authors showed that the problem $P, Sk|p_j, s_j = 1|C_{max}$ with k servers is binary $\mathcal{NP}$-hard and that the problem $P, Sk|p_j, s_j = 1|L_{max}$ is unary $\mathcal{NP}$-hard for any fixed number of machines m and any fixed number of servers k with $k < m$. In addition, they conducted a worst case analysis of two list scheduling algorithms for makespan minimization. For the case of loading and unloading operations involving multiple servers, only one paper is proposed in the literature for this variant. Jiang et al. [60] considered a parallel-machine scheduling problem with two dedicated servers: a loading server and an unloading server. Both servers are used to load and unload jobs onto and from the machines before and after their processing, respectively. The objective function involved the minimization of makespan. The authors showed that the classical list scheduling and largest processing time heuristics, have worst-case ratios, 8/5 and 6/5, respectively. For the case of unloading operations involving multiple servers, it is assumed that a job starts its processing immediately in an available machine without prior setup, and an unloading server is needed to remove this job from the machine. It can be noticed that only one paper dealing with this problem is proposed in the literature. In this context, Ou et al. [77] addressed the problem of scheduling m identical parallel machines with multiple unloading servers. The objective function involved the minimization of the total completion time. They showed that the shortest processing time first algorithm has a worst-case bound of 2 and proposed other heuristic algorithms as well as a branch-and-bound algorithm to solve the problem. They stated this variant of the PSS problem is motivated by the milk run operations of a logistics company that faces limited unloading docks at the warehouse.

1.7 Concluding remarks

In this chapter, we have surveyed several papers on parallel machine scheduling problems involving a single server that have appeared since 1996. This survey splits the literature in two main classes: (i) static parallel machine scheduling with a single server and (ii) dynamic parallel machine scheduling with a single server. In the static case, solution methods have been proposed for only four objective functions: makespan minimization, total machine idle time minimization, total weighted completion time minimization and total weighted tardiness minimization. In the dynamic case, only the objective of minimizing the makespan has been considered. The common exact methods suggested for the PSS problem and its variants are: branch and-bound algorithm [8, 72], branch and price algorithm [41] and mathematical programming formulations

[1, 14, 16, 38, 41, 47, 50, 56, 62, 88]. The common metaheuristics designed for the PSS problem and its variants are: simulated annealing [14, 47, 51, 54, 62], genetic algorithm [3, 47, 55, 56], tabu search [4, 14, 52], ant colony optimization [8], harmony search [51], geometric particle swarm optimization [4], iterative local search [88] and hybrid of simulated annealing and tabu search [62]. It can be noticed that some variants of the problem have received less attention of the research community than the others, for example: the case of multiple servers and also the case with loading/unloading operations which can be found in many manufacturing systems. Based on the existing literature on the PSS problem and its variants, in the next chapter, we present different mathematical formulation paradigms to model the problem $P, S1|p_j, s_j|C_{max}$, and we compare our results with those of Kim and Lee [62].

Chapter 2

MIP formulations for the parallel machine scheduling problem with a single server

Contents

2.1 Introduction

Mixed integer programming (MIP) formulations are well studied in the literature for different scheduling problems, such as: single machine, parallel machines, flow shop, job shop, open shop, etc. (see [75]). A MIP formulation for a minimization problem, may be written in the following form:

$$\begin{aligned} \min \quad & z = cx \\ s.t. \quad & Ax = b \\ & 0 \leq x_j \leq U_j, \quad j \in \Psi \\ & x_j \in \{0,1\}, \quad j \in \Psi \end{aligned}$$

where: A is a constant matrix, b is a constant vector, the set Ψ denotes the index set of decision variables. Each decision variable x_j has an upper bound denoted by U_j, which equals 1 if x_j is binary, and otherwise may be infinite. The main MIP formulations for scheduling problems can be classified according to the choice of the decision variables, namely: (i) completion time variables [12], (ii) time-indexed variables [91], (iii) linear ordering variables [35], (iv) assignment and positional date variables [71] and (v) network variables [80]. A comprehensive survey of MIP formulations for different scheduling problems was presented first by Blazewicz et al. [21]. Then, a comparison study of these formulations was conducted by Keha et al. [61] for various single machine scheduling problems with different objective functions such as weighted completion time, maximum lateness, number of tardy jobs, and total weighted tardiness. The computational results using CPLEX 8.1 optimization solver, showed that the performance of the proposed formulations depend on the number of jobs and the objective function. Later, Baker et al. [10] evaluated and compared these formulations for the single-machine total tardiness problem. The authors used CPLEX 11.0 as an optimization solver. The computational results showed that the MIP formulation based on assignment and positional date variables provided the most computationally effective solutions. Unlu et al. [100] performed a computational comparison of these formulations for the non-preemptive parallel machine

scheduling problems to minimize total weighted completion time, makespan, maximum lateness, total weighted tardiness and total number of tardy jobs. The authors used CPLEX 10.1 as optimization solver. The computational results showed that the MIP formulation based on time-indexed variables produced the least computation time for almost all optimally solved cases. Later, Ratli et al. [82] compared these formulations for the single machine scheduling problem with earliness and tardiness penalties. The computational results using CPLEX 11.2 optimization solver, showed that the MIP formulation based on time-indexed variables outperformed the other formulations in term of computational time and the number of instances solved to optimality. Recently, Kramer et al. [66] proposed a comparison study of these formulations, in addition to two other formulations, namely: set covering formulation (see [103]) and arc-flow formulation (see [33, 65]) for the parallel machine scheduling problem with family dependent setup times and total weighted completion time minimization. The authors used Gurobi 7.0 as an optimization solver. The computational results showed that the arc-flow with doubled jobs and set covering formulations, outperformed all other proposed formulations.

The goal of this chapter, is to identify promising MIP formulation paradigms that can be used to solve optimally small and medium-sized instances of the problem $P, S1|p_j, s_j|C_{max}$, and also to establish tight LP relaxation bounds. To do so, first we propose and investigate mathematical formulations to solve the problem $P, S1|p_j, s_j|C_{max}$ with a general job set. All proposed formulations include different decision variables to tackle the difficulties of the problem. The first formulation is based on network variables, the second formulation is based on linear ordering variables, the third formulation is based on completion time variables and the fourth formulation is based on time-indexed variables. We present also sets of valid inequalities that can be used to improve those formulations. Second, we propose a mathematical formulation based on assignment and positional date variables to solve a particular case of the problem $P, S1|p_j, s_j|C_{max}$ with a regular job set (i.e., $\forall i, j \quad p_j + s_j \geq p_i$). The remainder of this chapter is organized as follows: A formal description of the problem $P, S1|p_j, s_j|C_{max}$ is given in section 2.2. Then, existing mathematical formulations in the literature dealing with the problem are presented in section 2.3. In section 2.4 four mathematical formulations are proposed to model the problem, while in section 2.5, sets of valid inequalities are suggested to improved those formulations. A mathematical formulation for a particular case of the problem with a regular job set is presented in section 2.6. Computational results are then discussed in section 2.7. Finally, section 2.8 ends the chapter with conclusions and some perspectives.

2.2 Formal description of the problem

Formally, the addressed problem can be described as follows: we consider that a set of m identical parallel machines are available to process a set of n independent jobs. Each job j is available at the beginning of the scheduling period, and has a known integer processing time $p_j > 0$. Before its processing, job j must be loaded on a machine by the server. The loading operation, which can be also considered as a setup operation has a predefined integer value $s_j > 0$. During the setup operation, both the machine and the server are occupied and after loading a job, the server becomes available for loading the next job. The processing operation starts immediately after the end of the setup operation. The preemption of setup and processing operations of a job is not allowed. The objective is to find a feasible schedule that minimizes the makespan. The notations used to define the problem are shown in Table 2.1.

Table 2.1: Notations

Sets and indices	
n	number of jobs
m	number of machines
$N = \{1, 2, \ldots, n\}$	set of jobs
$N_D = \{n+1, \ldots, 2n\}$	set of dummy jobs
$\Omega = N \cup N_D$	
$M = \{1, 2, \ldots, m\}$	set of machines
i, j	job indices, where $i, j \in N$
k	machine indices, where $k \in M$
Parameters	
p_i	processing time of job i
s_i	setup time of job i
$L \geq \sum_{i=1}^{n}(s_i + p_i)$	very large positive integer
Decision Variables	
C_{max}	makespan
C_i	completion time of job i
$C_{[i]}$	completion time of the job scheduled in position i
$T_{[i]}$	start time of the setup operation of the job scheduled in position i

2.3 Existing formulations in the literature

Two mathematical formulations exist in the literature for the problem $P, S1|p_j, s_j|C_{max}$. Both formulations are proposed by Kim and Lee [62]. The first formulation is developed using assignment and positional dates variables in order to minimize the makespan, and the second formulation is developed to minimize the total server waiting time ($TSWT$). Indeed, the server waiting time denoted by $W_{[i]}$, is the time between the end of the setup operation of the job in position i and the start time of the setup operation of the job in position $i+1$. The equation of the server waiting time is as follows (see [1, 62]):

$$W_{[i]} = (p_{[i-1]} - s_{[i]} - W_{[i-1]})^+ \quad \forall i \in \{2, \ldots, n\}$$

Therefore, $TSWT$ is equal to $\sum_{i=1}^{n} W_i$. Kim and Lee [62] proved according to the proposition 1 that the minimization of the makespan is equivalent to the minimization of $TSWT$.

Proposition 1 *Makespan is equal to the sum of setup time of all jobs and waiting time in the sequence:*

$$C_{max} = \sum_{i=1}^{n} s_i + TSWT$$

Hence, the two formulations suggested by Kim and Lee [62] are equivalent. Moreover, for more details about the assignment and positional dates decision variables, readers can refer to [30, 31]. The first formulation proposed by Kim and Lee [62] which we denote as KL_1 is as follows:

Binary decision variables:

$$x_{[i],j} = \begin{cases} 1 & \text{if the job } j \text{ is assigned to the } i^{th} \text{ position in the schedule} \\ 0 & \text{otherwise} \end{cases}$$

$$y_{[i],k} = \begin{cases} 1 & \text{if the } i^{th} \text{ scheduled job is processed on machine } k \\ 0 & \text{otherwise} \end{cases}$$

MIP_{KL1} :

$$\min \quad C_{max} \tag{2.1}$$

$$s.t. \quad C_{max} \geq C_{[i]} \quad \forall i \in N \tag{2.2}$$

$$\sum_{i=1}^{n} x_{[i],j} = 1 \quad \forall j \in N \tag{2.3}$$

$$\sum_{j=1}^{n} x_{[i],j} = 1 \quad \forall i \in N \tag{2.4}$$

$$\sum_{k=1}^{m} y_{[i],k} = 1 \quad \forall i \in N \tag{2.5}$$

$$T_{[i]} \geq T_{[i-1]} + \sum_{j=1}^{n} s_j x_{[i-1],j} \quad \forall i \in \{2, \ldots, n\} \tag{2.6}$$

$$T_{[i]} \geq C_{[h]} - L(2 - y_{[i],k} - y_{[h],k}) \quad \forall i, h \in \{2, \ldots, n\} \quad h < i, \forall k \in M \tag{2.7}$$

$$C_{[i]} = T_{[i]} + \sum_{j=1}^{n} (s_j + p_j) x_{[i],j} \quad \forall i \in N \tag{2.8}$$

$$x_{[i]j} \in \{0, 1\} \quad \forall i \in N, \forall j \in N \tag{2.9}$$

$$y_{[i]k} \in \{0, 1\} \quad \forall i \in N, \forall k \in M \tag{2.10}$$

$$T_{[i]} \geq 0 \quad \forall i \in N \tag{2.11}$$

$$C_{[i]} \geq 0 \quad \forall i \in N \tag{2.12}$$

The objective function (2.1) indicates that the makespan is to be minimized. Constraints set (2.2) represents the restriction that the makespan of an optimal schedule is greater than or equal to the completion time of all executed jobs. Constraints sets (2.3) and (2.4) imply that only one job can be scheduled at the i^{th} position in the sequence and each job can be assigned at only one position in the sequence. Constraints set (2.5) ensures that the job has to be processed on one machine. Constraints sets (2.6) and (2.7) show that the job can be processed only if both the machines and the server are available simultaneously. Constraints set (2.8) represents that the completion time of the i^{th} scheduled job is equal to the sum of the starting time of the i^{th} scheduled job, setup time, and processing time of the job allocated to the i^{th} position. Constraints sets (2.9) - (2.10) define variables $x_{[i],j}$, $y_{[i],k}$ as binaries. Finally, constraints sets (2.11) - (2.12) are non-negativity restrictions.

The second formulation proposed by Kim and Lee [62] is as follows:

Binary decision variables:

$$x_{[i],j} = \begin{cases} 1 & \text{if the job } j \text{ is assigned to the } i^{th} \text{ position in the schedule} \\ 0 & \text{otherwise} \end{cases}$$

$$y_{[i],k} = \begin{cases} 1 & \text{if the } i^{th} \text{ scheduled job is processed on machine } k \\ 0 & \text{otherwise} \end{cases}$$

MIP_{KL2} :

$$\min \quad \sum_{i=1}^{n} W_{[i]} \tag{2.13}$$

$$s.t \quad \sum_{i=1}^{n} x_{[i],j} = 1 \quad \forall j \in N \tag{2.14}$$

$$\sum_{j=1}^{n} x_{[i],j} = 1 \quad \forall i \in N \tag{2.15}$$

$$\sum_{k=1}^{m} y_{[i],k} = 1 \quad \forall i \in N \tag{2.16}$$

$$W_{[i]} \geq \sum_{j=1}^{n} p_j x_{[g],j} - \sum_{h=g+1}^{i} \sum_{j=1}^{n} s_j x_{[h]j} - \sum_{h=g}^{i-1} W_h - L(2 - y_{[i+1],k} - y_{[g],k}) \quad \forall i, g \in \{1, \ldots, n-1\}, g \leq i, \forall k \in M \tag{2.17}$$

$$W_{[n]} \geq \sum_{j=1}^{n} p_j x_{[i],j} - \sum_{h=i+1}^{n} \sum_{j=1}^{n} s_j x_{[h]j} - \sum_{h=i}^{n-1} W_h \quad \forall i \in N \tag{2.18}$$

$$x_{[i],j} \in \{0, 1\} \quad \forall i \in N, \forall j \in N \tag{2.19}$$

$$y_{[i],k} \in \{0, 1\} \quad \forall i \in N, \forall k \in M \tag{2.20}$$

$$W_{[i]} \geq 0 \quad \forall i \in N \tag{2.21}$$

The objective function (2.13) indicates that $TSWT$ is to be minimized. Constraints sets (2.14) and (2.3) imply that only one job can be scheduled at the i^{th} position in the sequence and each job can be assigned at only one position in the sequence. Constraints set (2.4) ensures that the job has to be processed on one machine. Constraints sets (2.5) and (2.6) are proved in [62]. Constraints sets (2.7) - (2.8) define variables $x_{[i],j}$, $y_{[i],k}$ as binaries. Finally, constraints set (2.11) is non-negativity restriction. It can be noticed that both formulations proposed by Kim and Lee [62], are based on assignment and positional dates variables.

2.4 Formulations for a general job set

In this section, we propose mathematical formulations to model the problem $P, S1|p_j, s_j|C_{max}$. The first formulation is based on network variables, the second formulation is based on linear ordering variables, the third formulation is based on completion time variables and the last formulation is based time-indexed variables. In our formulations, we consider the server as the $(m+1)^{th}$ machine. Each time the server is used to load job $i \in N$ on machine $k \in M$, then a dummy job $(i+n) \in N_D$ is processed on a dummy machine $(m+1)$ at the same time. This dummy job $(i+n)$, has a processing time equal to the setup time of the job i (i.e., $p_{i+n} = s_i \ \forall i \in N$).

2.4.1 Completion time variables formulation

Completion time variables were used for the first time by Balas [12] in his disjunctive formulation for the job shop scheduling problem, and was also studied later by Queyranne and Wang [81] and Queyranne [79]. In order to formally model the problem $P, S1|p_j, s_j|C_{max}$ by completion time variables formulation which is denoted by (MIP_1), we use the following binary decision variables:

$$x_{i,k} = \begin{cases} 1 & \text{if job } i \text{ is processed on machine } k \\ 0 & \text{otherwise} \end{cases}$$

$$z_{i,j} = \begin{cases} 1 & \text{if job } i \text{ is processed before job } j \\ 0 & \text{otherwise} \end{cases}$$

$$z_{i+n,j+n} = \begin{cases} 1 & \text{if dummy job } (i+n) \text{ is processed before} \\ & \text{dummy job } (j+n) \\ 0 & \text{otherwise} \end{cases}$$

MIP_1 :

$$\min \quad C_{max} \tag{2.22}$$

$$s.t. \quad C_{max} \geq C_i \quad \forall i \in N \tag{2.23}$$

$$\sum_{k=1}^{m} x_{i,k} = 1 \quad \forall i \in N \tag{2.24}$$

$$C_i \geq p_{i+n} + p_i \quad \forall i \in N \tag{2.25}$$

$$C_i + p_{j+n} + p_j \leq C_j + L(3 - x_{i,k} - x_{j,k} - z_{i,j}) \quad \forall (i,j) \in N^2, i < j, \forall k \in M \tag{2.26}$$

$$C_j + p_{i+n} + p_i \leq C_i + L(2 - x_{i,k} - x_{j,k} + z_{i,j}) \quad \forall (i,j) \in N^2, i < j, \forall k \in M \tag{2.27}$$

$$C_i + p_{j+n} + p_j \leq C_j + p_i + L(1 - z_{i+n,j+n}) \quad \forall (i,j) \in N^2, i < j \tag{2.28}$$

$$C_j + p_{i+n} + p_i \leq C_i + p_j + L z_{i+n,j+n} \quad \forall (i,j) \in N^2, i < j \tag{2.29}$$

$$x_{i,k} \in \{0,1\} \quad \forall i \in N, \forall k \in M \tag{2.30}$$

$$z_{i,j} \in \{0,1\} \quad \forall (i,j) \in \Omega^2 \tag{2.31}$$

In this model, the objective function (2.22) indicates that the makespan has to be minimized. Constraints (2.23) represent the restriction that the makespan of an optimal schedule is greater or equal than the completion time of all executed jobs. To guarantee that a job i is scheduled on exactly one machine, constraints (2.24) are added to the model. The completion time C_i is calculated according to constraints (2.25). Constraints (2.26) to (2.29) show that a job can be processed only if both the machines and the dummy machine are available simultaneously. Constraints (2.26) and (2.27) indicate that no two jobs i and j, scheduled on the same machine (i.e., $x_{i,k} = x_{j,k} = 1$), can overlap in time. Constraints (2.28) and (2.29) state that the dummy machine $m+1$, can execute at most one dummy job at a time. Finally, constraints (2.30) - (2.31) define variables $x_{i,k}$ and $z_{i,j}$ as binaries.

2.4.2 Time indexed variables formulation

Time indexed variables (TIV) formulation was introduced by Sousa and Wolsey [91] for the non-preemptive single machine scheduling problem. Later, Van den Akker et al. [102], Soric [89] and Pessoa et al. [78] studied this formulation for different machines scheduling environment problems. TIV formulation is based on time-discretization. Time is divided into periods $1, 2, 3, \ldots, T$ where period t starts at time $t-1$ and ends at time t. The planning horizon T is an important part of the model and the size of the TIV formulation depends on. Any upper bound on the cost of an optimal solution can be chosen as T. However, a tighter upper bound is preferable to reduce the problem size as the number of time-points is pseudo-polynomial in the size of the input (see [90]). The following proposition provides an upper bound and a lower bound for the problem $P, S1|p_j, s_j|C_{max}$.

Proposition 2 *Let C^*_{max} denotes the objective function value of an optimal solution of the $P, S1|p_j, s_j|C_{max}$ and $\overline{P} = \sum_{i \in N}(s_i + p_i)$. We consider also a permutation σ of the jobs such that all jobs are ordered by their setup times $s_{\sigma(1)} \leq s_{\sigma(2)} \leq \ldots \leq s_{\sigma(n)}$.*

Then, we have:

$$\max\Big(\frac{\overline{P} + \overline{J_p}}{m}, \sum_{1 \leq i \leq n} s_i + \min_{1 \leq i \leq n} p_i\Big) \leq C^*_{max} \leq \overline{P}$$

Where: $\overline{J_p} = (m-1)s_{\sigma(1)} + (m-2)s_{\sigma(2)} + (m-3)s_{\sigma(3)} + \ldots + s_{\sigma(m-1)}$

Proof

- Case (i): If there is no server waiting time in an optimal schedule of the $P, S1|p_j, s_j|C_{max}$, then C^*_{max} will be equal to the sum of setup times plus the shortest processing time, and then inequality:

$$C^*_{max} \geq \sum_{1 \leq i \leq n} s_i + \min_{1 \leq i \leq n} p_i$$

will be always satisfied.

- Case (ii): If there is no machine idle time (i.e., the gap between the end of processing time and the start time of the setup operation of two jobs scheduled on the same machine is equal to zero) in an optimal schedule of the $P, S1|p_j, s_j|C_{max}$, then C^*_{max} will be equal to the sum of setup times plus the sum of processing times plus $\overline{J_p}$ which corresponds to:

$$(m-1)s_{\sigma(1)} + (m-2)s_{\sigma(2)} + (m-3)s_{\sigma(3)} + \ldots + s_{\sigma(m-1)}$$

divided by the number of machines m. Indeed, it is clear that each job from 1 to m must use a different machine. So, the first job in the schedule $\sigma(1)$ starts at time zero, the second job $\sigma(2)$ at $t = s_{\sigma(1)}$, the third job at $t = s_{\sigma(1)} + s_{\sigma(2)}$ and so on. Thus, each machine (except the first one) will be available only from a deadline $s_{\sigma(i)} > 0$. The fact of adding these deadlines with all the setup times and the processing times will constitute the total load to be executed by the m machines. It is then sufficient to divide this charge by m to obtain the aforementioned lower bound.

From Case (i) and Case (ii), we have:

$$\max\Big(\frac{\overline{P} + \overline{J_p}}{m}, \sum_{1 \leq i \leq n} s_i + \min_{1 \leq i \leq n} p_i\Big) \leq C^*_{max}$$

- Case (iii): For the case of an arbitrary number of machines, it is easy to see that all jobs can't be scheduled on one machine. Thus:

$$C^*_{max} \leq \overline{P}$$

From Proposition 2, it follows that the upper bound T can be chosen as $T = \overline{P}$. Indeed, to reduce the length of the time horizon, we propose to fix it to the value of the approximate makespan solution given by the two greedy heuristics proposed by El idrissi el al. [36] for the problem $P, S1|p_j, s_j|C_{max}$. A comparative study is made between these two values of T in the computational results (Section 2.7) as initial and improved formulation. TIV formulation for the problem $P, S1|p_j, s_j|C_{max}$ is denoted by MIP_2. In addition, we refer to MIP_2 with the improved value of the time horizon T as time-indexed variables improvement formulation ($MIP_{2'}$). Binary decision variables for MIP_2 are defined as follows:

$$x_{i,t} = \begin{cases} 1 & \text{if job } i \text{ starts processing at time } t \\ 0 & \text{otherwise} \end{cases}$$

MIP_2 :

$$\min \quad C_{max} \tag{2.32}$$

$$s.t. \quad \sum_{t=0}^{T-p_{i+n}-p_i} (t + p_{i+n} + p_i) x_{i,t} \leq C_{max} \quad \forall i \in N \tag{2.33}$$

$$\sum_{t=0}^{T-p_i} x_{i,t} = 1 \quad \forall i \in N_D \tag{2.34}$$

$$\sum_{i=n+1}^{2n} \sum_{s=max(0,t-p_i+1)}^{t} x_{i,s} \leq 1 \quad \forall t \in [0, T] \tag{2.35}$$

$$\sum_{t=0}^{T-p_{i+n}-p_i} x_{i,t} = 1 \quad \forall i \in N \tag{2.36}$$

$$\sum_{i=1}^{n} \sum_{s=max(0,t-p_{i+n}-p_i+1)}^{t} x_{i,s} \leq m \quad \forall t \in [0, T] \tag{2.37}$$

$$x_{i+n,t} = x_{i,t} \quad \forall i \in N, \forall t \in [0, T - p_{i+n} - p_i] \tag{2.38}$$

$$x_{i,t} \in \{0, 1\} \quad \forall i \in \Omega, \forall t \in [0, T - p_{i+n} - p_i] \tag{2.39}$$

In this model, the objective function (2.32) indicates that the makespan, is to be minimized. Constraints (2.33) represent the restriction that the makespan of an optimal schedule is greater than or equal to the completion time of all executed jobs, where job i that starts its processing at time point t (i.e., the job for which $x_{i,t} = 1$) and will finish at time $C_i = t + p_{i+n} + p_i$. The completion time of job i is calculated as

$C_i = \sum_{t=0}^{T-p_{i+n}-p_i}(t + p_{i+n} + p_i)x_{i,t}$. Constraints (2.34) state that each dummy job must start at some time point t on the dummy machine in the scheduling horizon. The set of constraints (2.35) ensures that, at any given time, at most one dummy job can be processed on the dummy machine. Constraints (2.36) state that each job i must start at some time-point t in the scheduling horizon. Constraints (2.37) ensure that at any given time at most m jobs can be processed on all machines. Constraints (2.38) state that the start time of the dummy job $(i+n)$ on the dummy machine and the start time of the job i is the same (i.e., $x_{i,t} = x_{i+n,t}$). The last set of constraints (2.39) defines variables $x_{i,t}$ as binaries.

2.4.3 Linear ordering variables formulation

Linear ordering variables were first used by Dyer and Wolsey [35] in their single machine scheduling problem with release dates. Binary decision variables for this formulation which is denoted by (MIP_3), are defined as follows:

$$\delta_{i,j} = \begin{cases} 1 & \text{if job } i \text{ precedes job } j \\ 0 & \text{otherwise} \end{cases}$$

$$\delta_{i+n,j+n} = \begin{cases} 1 & \text{if job dummy job } (i+n) \text{ precedes dummy job } (j+n) \\ 0 & \text{otherwise} \end{cases}$$

Note that job i is not necessarily positioned immediately before job j when $\delta_{i,j} = 1$.

$$z_{i,k} = \begin{cases} 1 & \text{if job } i \text{ is scheduled on machine } k \\ 0 & \text{otherwise} \end{cases}$$

$$y_{i,j} = \begin{cases} 1 & \text{if jobs } i \text{ and } j \text{ are not scheduled on the same machine} \\ 0 & \text{otherwise} \end{cases}$$

MIP_3 :

$$\min \quad C_{max} \tag{2.40}$$

$$s.t. \quad C_{max} \geq C_i \quad \forall i \in N \tag{2.41}$$

$$\delta_{i,j} + \delta_{j,i} + y_{i,j} = 1 \quad \forall (i,j) \in N^2, i < j \tag{2.42}$$

$$\delta_{i,j} + \delta_{j,k} + \delta_{k,i} \leq 2 \quad \forall (i,j,k) \in N^3, i < j < k \tag{2.43}$$

$$z_{i,k} + z_{j,k} + y_{i,j} \leq 2 \quad \forall (i,j) \in N^2, \forall k \in M, i < j \tag{2.44}$$

$$\sum_{k=1}^{m} z_{i,k} = 1 \quad \forall i \in N \tag{2.45}$$

$$C_i \geq p_{i+n} + p_i \quad \forall i \in N \tag{2.46}$$

$$C_i + (p_j + p_{j+n})(\delta_{i,j} + z_{i,k} + z_{j,k} - 2)$$
$$-L(1 - \delta_{i,j}) \leq C_j \quad \forall (i,j) \in N^2, \forall k \in M, i \neq j \tag{2.47}$$

$$C_i + p_{j+n} + p_j \leq C_j + p_i + L(1 - \delta_{i+n,j+n}) \quad \forall (i,j) \in N^2, i \neq j \tag{2.48}$$

$$\delta_{i+n,j+n} + \delta_{j+n,i+n} \geq 1 \quad \forall (i,j) \in N^2, i \neq j \tag{2.49}$$

$$\delta_{i,j} \in \{0,1\} \quad \forall (i,j) \in \Omega^2 \tag{2.50}$$

$$z_{i,k} \in \{0,1\} \quad \forall i \in N, \forall k \in M \tag{2.51}$$

$$y_{i,j} \in \{0,1\} \quad \forall (i,j) \in N^2 \tag{2.52}$$

The objective function (2.40) indicates that the makespan is to be minimized. Constraints set (2.41) confirms that the makespan is greater than or equal to the completion time of each job. Constraints set (2.42) ensures that either job i is positioned before job j or after, provided the two jobs are scheduled on the same machine. Constraints set (2.43) represents the transitivity constraints that ensure a linear order between three jobs. Constraints set (2.44) properly calculates variable $y_{i,j}$. Constraints set (2.45) ensures that each job is positioned on only one machine. The completion time C_i of the job i is calculated according to constraints (2.46). Constraints (2.47) indicate that no two jobs i and j, scheduled on the same machine, can overlap in time. Constraints (2.48) and (2.49) state that the dummy machine, can execute at most one dummy job at a time. Constraints sets (2.50) - (2.51) - (2.52) define variables $\delta_{i,j}$, $z_{i,k}$ and $y_{i,j}$ as binaries.

2.4.4 Networks variables formulation

Network variables were initially used by Queyranne and Schulz [80] to model the single machine scheduling problem with sequence dependent processing times. This formulation can be regarded also as a vehicle routing problem (VRP) variables formulation where the jobs to be scheduled are modeled as customers and the machines represent the vehicles being routed (see Unlu and Mason [100]). In this formulation, job 0 is required to be both the first and the last job processed on each machine, its processing time is set to 0. In this way, it indicates both the starting and finishing of job processing on each machine. Binary decision variables for this formulation which is denoted by (MIP_4), are defined as follows:

$$x_{i,j} = \begin{cases} 1 & \text{if job } i \text{ immediately precedes job } j \text{ on the same machine} \\ 0 & \text{otherwise} \end{cases}$$

Due to the server availability constraints, it must be ensured that jobs requiring this server do not overlap over time. Therefore, another binary decision variable is introduced:

$$e_{i+n,j+n} = \begin{cases} 1 & \text{if dummy job } (i+n) \text{ finishes its processing} \\ & \text{before dummy job } (j+n) \\ 0 & \text{otherwise} \end{cases}$$

MIP_4 :

$$\min \quad C_{max} \tag{2.53}$$

$$s.t. \quad C_{max} \geq C_i \quad \forall i \in N \tag{2.54}$$

$$\sum_{j=1}^{n} x_{0,j} \leq m \tag{2.55}$$

$$\sum_{i=1}^{n} x_{i,0} \leq m \tag{2.56}$$

$$\sum_{j=0:j\neq i}^{n} x_{i,j} = 1 \quad \forall i \in N \tag{2.57}$$

$$\sum_{i=0:i\neq j}^{n} x_{i,j} = 1 \quad \forall j \in N \tag{2.58}$$

$$C_i \geq p_{i+n} + p_i \quad \forall i \in N \tag{2.59}$$

$$C_i + p_{j+n} + p_j \leq C_j + L(1 - x_{i,j}) \qquad \forall (i,j) \in N^2, i \neq j \tag{2.60}$$

$$C_i + p_{j+n} + p_j \leq C_j + p_i + L(1 - e_{i+n,j+n}) \qquad \forall (i,j) \in N^2, j \neq i \tag{2.61}$$

$$e_{i+n,j+n} + e_{j+n,i+n} \geq 1 \quad \forall (i,j) \in N^2, i \neq j \tag{2.62}$$

$$x_{i,j} \in \{0,1\} \quad \forall i \in N \cup \{0\}, \forall j \in N \cup \{0\} \tag{2.63}$$

$$e_{i+n,j+n} \in \{0,1\} \quad \forall (i,j) \in N^2 \tag{2.64}$$

The objective function (2.53) indicates that the makespan is to be minimized. Constraints set (2.54) represents the restriction that the makespan of an optimal schedule is greater than or equal to the completion time of all executed jobs. Constraints sets (2.55) and (2.56) are presented in a manner typically associated with the VRP, where the jobs are assigned to each of the m available machines, subject to each machine starting and

ending its schedule with job 0. Constraints set (2.57) and (2.58) guarantee that all jobs are scheduled on a particular machine. The completion time C_i of the job i is calculated according to constraints (2.59). Constraints (2.60) indicate that no two jobs i and j, scheduled on the same machine, can overlap in time. Constraints sets (2.61) and (2.62) state that the dummy machine can execute at most one dummy job at a time. Constraints sets (2.63) - (2.64) define variables $x_{i,j}$, $e_{i+n,j+n}$ as binaries.

2.4.5 Formulations size

In order to analyze the effect of variables selection on formulation size, Table 2.2 shows the number of constraints, the number of binary variables and formulation size associated with each proposed formulation. All formulations have a polynomial number of variables and constraints in the number of jobs. However, this is not the case for MIP_2, as it is strongly dependent on the time horizon T. The number of constraints and variables for MIP_2 can be very large, even for small instances if the sum of setup and processing times is relatively large.

Table 2.2: The size for each proposed formulation

Formulation	Number of constraints	Number of binary variables	Formulation size
MIP_1	$2n^2(m+1)+3n$	$4n^2+nm$	$O(n^2)$
MIP_2	$(2+n)T+3n-\overline{P}$	$nT-\overline{P}$	$O(T)$
MIP_3	$3n^2(m+1)+2n$	$5n^2+nm$	$O(n^2)$
MIP_4	$3n^2+4n+2$	$2n(n+1)+1$	$O(n^2)$
KL_1	n^2m+6n	n^2+nm	$O(n^2)$
KL_2	n^2m+4n	n^2+nm	$O(n^2)$

2.5 Valid inequalities

In order to improve models tractability and reduce the time required to solve the problem by one of the proposed MIP_1, MIP_3 and MIP_4 formulations, the following constraints sets can be added:

Proposition 3 *The following constraints*

$$C_{max} \geq \sum_{i=1}^{n}(s_i+p_i)x_{i,k} \qquad \forall k \in M \qquad (2.65)$$

are valid for MIP_1.

Proof The completion time of the last job scheduled on machine k satisfies:

$$C_k \ \geq \ \sum_{i=1}^{n} (s_i + p_i) x_{i,k} \quad \forall k \in M$$

where C_k represents the makespan of machine k. Thus, since the makespan of a machine is greater than or equal to the completion times of all jobs scheduled in this machine. Hence, the proposition is valid.

We refer to MIP_1 considering the set of constraints (2.65) as $(MIP_{1'})$.

Proposition 4 *The following constraints*

$$C_{max} \geq \sum_{i=1}^{n} (s_i + p_i) z_{i,k} \qquad \forall k \in M \qquad (2.66)$$

are valid for MIP_3.

Proof Analogous to the proof of Proposition 3

We refer to MIP_3 considering the set of constraints (2.66) as $MIP_{3'}$.

Proposition 5 *The two sets of constraints*

$$C_i \geq \sum_{j=1:j\neq i}^{n} (s_j + p_j) x_{j,i} + s_i + p_i \qquad \forall i \in N \qquad (2.67)$$

$$C_i \leq L - \sum_{j=1:j\neq i}^{n} (s_j + p_j) x_{i,j} \qquad \forall i \in N \qquad (2.68)$$

are valid for MIP_4.

Proof The valid equalities (2.67) and (2.68) are directly deduced from constraints (2.60).

We refer to MIP_4 considering the set of constraints (2.67) and (2.68) as $(MIP_{4'})$. In addition, we refer to MIP_4 considering the sets of constraints (2.67) as $(MIP_{4''})$.

2.6 Formulation for a regular job set

In this section, we propose a mathematical formulation dedicated to the $P, S1|p_j, s_j|C_{max}$ with a regular job set. Instead of using one of the proposed MIP formulations designed for a general job set, it is better to use the above formulation that benefits from the structure of a regular job set. Before presenting the MIP formulation proposed for this variant of the problem, we define the concept of a regular job set.

2.6.1 Definition and some properties of a regular job set

Definition 1 *According to Abdekhodaee et al. [1]. A set of jobs is regular, if:*

$$p_i \leq s_j + p_j \quad \forall i, j \in \{1, 2, \ldots, n\} \tag{2.69}$$

Indeed, regularity helps to simplify and makes it possible to model the problem as a mixed integer program. Abdekhodaee et al. [3], showed that the region which satisfies the inequality 2.69 is all of the shaded part of Figure 2.1. The regular region can be divided in various parts as shown in Figure 2.1 (this figure is taken from [3]).

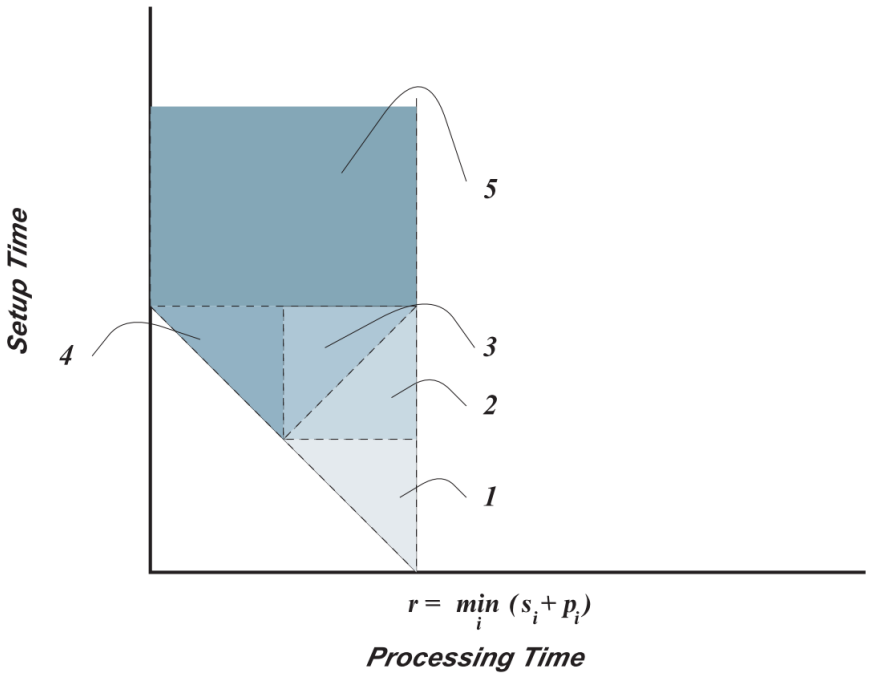

Figure 2.1: Regular region [3]

- In part 1, $s_j \leq r/2 \leq p_i \quad \forall i, j \in N$
- In part 2, $r/2 \leq s_i \leq p_i \quad \forall i, j \in N$
- In part 3, $r/2 \leq p_i \leq s_i \quad \forall i, j \in N$
- In part 4, $p_i \leq r/2 \leq s_j \quad \forall i, j \in N$
- In part 5, $p_i \leq r \leq s_j \quad \forall i, j \in N$

For each part a special MIP formulation can be designed, which may not necessarily be effective for other parts. In the following, a MIP formulation is proposed for the overall regular region. Additional notation and some new properties are needed to define the MIP formulation:

Decision Variables:

- $x_{[i]j} = \begin{cases} 1 & \text{if job } j \text{ is assigned to the } i^{th} \text{ position in the sequence} \\ 0 & \text{otherwise} \end{cases}$

Proposition 6 *For the case of an arbitrary number of machines, if the jobs are regular then they are processed alternately between the m machines.*

Proof We assume that jobs are regular. For the first m jobs, it's clear that each job in position $i \leq m$, will be scheduled on one of the m available machines. Hence:

$$C_{[1]} \leq C_{[2]} \leq \ldots \leq C_{[m]}$$

For the jobs in positions $i > m$, we have:

$$T_{[i]} + s_{[i]} \leq T_{[i+1]}$$

In addition, since we have regular jobs:

$$p_{[i]} \leq s_{[i+1]} + p_{[i+1]}$$

Thus: $C_{[i]} \leq C_{[i+1]} \leq \ldots \leq C_{[i+m]}$ and no two jobs in position i and $i+1$ can be scheduled on the same machine. Hence, jobs will be carried out alternately on the m machines.

Proposition 7 *Under the assumption of a regular job set, for all $i \in N \cup \{0\}$, $T_{[i]}$ is computed as follows:*

$$T_{[i]} = \begin{cases} T_{[0]} = s_{[0]} = p_{[0]} = 0 & \\ T_{[i-1]} + s_{[i-1]} & if \quad 1 \leq i \leq m \\ \max(T_{[i-1]} + s_{[i-1]}, \min\limits_{1 \leq k \leq m} \left(T_{[i-k]} + s_{[i-k]} + p_{[i-k]}\right)) & if \quad m+1 \leq i \leq n \end{cases}$$

Proof First of all, one can observe that a job starts its setup operation immediately whenever the server and a machine are available simultaneously. Otherwise, the job will be unnecessarily delayed. For the first m jobs, each job in position $i \leq m$ will start immediately its setup operation after the completion of the setup operation of the job in position $i-1$ on one of the $\{1, \ldots, m\}$ available machines (where $T_{[0]} = 0$, since all jobs are available at beginning of the schedule). This is trivial because each time, at least, one machine will be available for processing this job.

So:

$$T_{[i]} = T_{[i-1]} + s_{[i-1]} \quad \forall i \in \{1, \ldots, m\}$$

For the second part of the property. Let suppose that we want to schedule a job in position $i > m$, then its start time $T_{[i]}$ will depend on the availability of the server and a machine. The server will be available to perform the setup operation of the job at position i if:

$$T_{[i]} \geq T_{[i-1]} + s_{[i-1]}$$

Second, according to (Proposition 6), if the jobs are regular then they are processed alternately between machines. Since jobs are regular no two consecutive jobs in position i and $i+1$ can be scheduled on the same machine. Hence, the first available machine k' corresponds to the one with smallest completion times of the processing operations of jobs in positions $i-1$, $i-2$, ..., $i-m$. i.e.,

$$k' = \arg\min_{1 \leq k \leq m} \left(T_{[i-k]} + s_{[i-k]} + p_{[i-k]}\right)$$

Finally, in order to assure that both the server and a machine are available at same time, we choose the maximum of the two values: $T_{[i-1]} + s_{[i-1]}$ and $\min\limits_{1 \leq k \leq m} \left(T_{[i-k]} + s_{[i-k]} + p_{[i-k]}\right)$.

Thus:

$$T_{[i]} = \max\left(T_{[i-1]} + s_{[i-1]}, \min_{1\leq k\leq m}\left(T_{[i-k]} + s_{[i-k]} + p_{[i-k]}\right)\right) \quad \forall i \in \{m+1,\ldots,n\}$$

2.6.2 Mathematical formulation

In this subsection, we deduce from Proposition 7 a mathematical formulation for the problem $P, S1|p_j, s_j|C_{max}$ with a regular job set, which is denoted by (MIP_5).

MIP_5 :

$$\min \quad C_{max} \tag{2.70}$$

$$s.t. \quad C_{max} \geq C_{[i]} \quad \forall i \in N \tag{2.71}$$

$$\sum_{i=1}^{n} x_{[i]j} = 1 \quad \forall j \in N \tag{2.72}$$

$$\sum_{j=1}^{n} x_{[i]j} = 1 \quad \forall i \in N \tag{2.73}$$

$$C_{[i]} - T_{[i]} = \sum_{j=1}^{n}(s_j + p_j)x_{[i]j} \quad \forall i \in N \tag{2.74}$$

$$T_{[i]} - T_{[i-1]} \geq \sum_{j=1}^{n} s_j x_{[i-1]j} \quad \forall i \in \{2,\ldots,n\} \tag{2.75}$$

$$T_{[i]} \geq \min_{1\leq k\leq m}(T_{[i-k]} + s_{[i-k]} + p_{[i-k]}) \quad \forall i \in \{m+1,\ldots,n\} \tag{2.76}$$

$$x_{[i]j} \in \{0,1\} \quad \forall i \in N, \forall j \in N \tag{2.77}$$

The objective function (2.70) indicates that the makespan, i.e. the completion time of the last job that finishes its processing on the machines, is to be minimized. Constraints set (2.71) represents the restriction that the makespan of an optimal schedule is greater than or equal to the completion time of the last executed job. Constraints set (2.72) guarantees that each position i contains exactly one job. Constraints set (2.73) ensures that all jobs are assigned to exactly one position. Constraints set (2.74) indicates that the completion time of the i^{th} scheduled job is equal to the sum of the start time of the i^{th} scheduled job, setup time, and processing time of the job allocated to the i^{th} position. Constraints (2.75) - (2.76) derive from Proposition 7. Constraints set (2.77) defines variables $x_{[i]j}$

as binaries. Obviously, constraints (2.76) are nonlinear, but all other constraints and the objective function are linear. Thus, MIP_5 (2.70)-(2.77) is a mixed integer nonlinear programming model. In this subsection, we propose a linearisation of this model.

Proposition 8 *Let us consider MIP_5 (2.70)-(2.77). Constraints (2.76) can be linearised as follows:*

$$T_{[i]} \geq T_{[i-k]} + s_{[i-k]} + p_{[i-k]} - L(1 - y_k) \quad \forall i \in \{m+1, \ldots, n\}, \forall k \in M \tag{2.78}$$

$$\sum_{k=1}^{m} y_k \geq 1 \tag{2.79}$$

$$y_k \in \{0, 1\} \quad \forall k \in M \tag{2.80}$$

Based on Proposition 8, the resulting mixed integer linear programming model for the $P, S1|p_j, s_j|C_{max}$ with a regular job set (MIP_5), is defined by (2.70)-(2.75),(2.77)-(2.80), where (2.78)-(2.80) refer to the linearisation.

2.7 Computational results

All tests presented in this section were conducted on a personal computer Intel(R) Core(TM) with i7-4600M 2.90 GHz CPU and 16GB of RAM, running Windows 7. To solve the MIP formulations we have used the Concert Technology library of CPLEX 12.6 version in C++. The MIP formulations are compared in terms of (1) the number of test instances solved to optimality within 1 hour, $\#opt$, (2) the average solution time for these optimally solved instances, $t(s)$, (3) the number of test instances unsolved within 1 hour (instances with feasible solutions), $\#is$, (4) the average optimality gap for the test instances which could not be solved within 1 hour, $gap_1(\%)$, (5) the average gap between the formulation's lower bound and the best upper bound, $gap_2(\%)$, (6) the average gap between the linear programming (LP) relaxation lower bound and the best upper bound, $gap_{lp}(\%)$, and (7) the number of branch and bound nodes analysed within 1 hour, $\#BB$. The symbol $\star$ is used to specify that no feasible solution was found within a time limit of 1 hour.

2.7.1 Benchmark instances

In our experiments, we considered two sets of benchmark instances.

(i) The first one was originally proposed by [62], to generate a general job set. Instances were generated randomly for the case where: $E(p_j) \geq E(s_j)$, where $E(x)$ denotes the mean of x, using the combination of four parameters: (1) n is the number of jobs, (2) m is the number of machines, (3) α is the diversity factor for the range of setup times, and for processing times and (4) ρ is the setup time severity factor, defined as $\rho = mE(s_i)/E(p_i)$. Where: $E(p_i) = 25$ and $\rho \in [0.5, 1.0]$. Setup times and processing times are randomly generated as integer values within the ranges: $s_j \in [E(s_j) - \alpha E(s_j), E(s_j) + \alpha E(s_j)]$, $p_j \in [E(p_j) - \alpha E(p_j), E(p_j) + \alpha E(p_j)]$ for each job j. The parameter α is chosen in the interval [0.1, 0.5]. This set is composed of instances with $n \in \{10, 20, 50, 100\}$, $m \in \{3, 4, 5, 7\}$, $(\alpha = 0.1, \rho = 0.5)$, $(\alpha = 0.1, \rho = 0.7)$, $(\alpha = 0.1, \rho = 1.0)$, $(\alpha = 0.3, \rho = 0.5)$, $(\alpha = 0.3, \rho = 0.7)$, $(\alpha = 0.3, \rho = 1.0)$, $(\alpha = 0.5, \rho = 0.5)$, $(\alpha = 0.5, \rho = 0.7)$ and $(\alpha = 0.5, \rho = 1.0)$.

(ii) The second one, is a regular job set. To generate a regular job set, first we generate a general job set, where the processing time values p_j are generated from a discrete uniform distribution $U(1, 100)$ and the setup time values s_j are generated from a discrete uniform distribution $U(1, 100)$. Then, we reduce this general job set into a regular one by using the Koulama's reduction algorithm (see [64]). This set is composed of instances with $n \in \{20, 40, 100, 250, 350, 500\}$ and $m \in \{3, 4, 5, 6\}$. All instances are publicly available at https://sites.google.com/site/dataforpssproblem/data.

The computational results section consists of two parts. In the first part we test the formulations with a general job set, while in the second part with a regular job set.

2.7.2 Comparison of formulations – general job set

In this section, we present the computational results for the benchmark instances with a general job set. We evaluate the performance of the proposed mathematical formulations MIP_1, $MIP_{1'}$, MIP_2, $MIP_{2'}$, MIP_3, $MIP_{3'}$, MIP_4, $MIP_{4'}$ and $MIP_{4''}$ and compare their performances with the formulations MIP_{KL1} (to minimise the makespan) and MIP_{KL2} (to minimise the total server waiting time) both proposed in [62].

The general job set is composed of instances with: $(n, m) \in \{(10, 3); (10, 4); (20, 3)\}$ and $(\alpha, \rho) \in \{(0.1, 0.5); (0.1, 1.0); (0.3, 0.5); (0.3, 0.7); (0.5, 0.7); (0.5, 1.0)\}$ and the instances with: $(n = 50, m = 3, (\alpha = 0.1, \rho = 0.5))$, $(n = 50, m = 3, (\alpha = 0.5, \rho = 0.5))$, $(n = 50, m = 7, (\alpha = 0.1, \rho = 0.7))$, $(n = 50, m = 7, (\alpha = 0.3, \rho = 1.0))$ and $(n = 100, m = 5, (\alpha = 0.1, \rho = 0.5))$. For each value of the combination $(n, m, (\alpha, \rho))$, 10 instances are generated randomly, totalising 230 instances.

These results are detailed in Table 2.3, 2.4, 2.5, 2.6 and 2.7:

For $n = 10$ and $m = 3$ (Table 2.3): MIP_1 and MIP_3 are the only formulations for which CPLEX is not able to find an optimal solution for any instance. The best overall performance is demonstrated by: $MIP_{2'}$ for ($[\alpha = 0.1, \rho = 0.5], [\alpha = 0.3, \rho = 0.5], [\alpha = 0.5, \rho = 0.7]$) and by MIP_{KL2} for ($[\alpha = 0.1, \rho = 1.0], [\alpha = 0.3, \rho = 0.7], [\alpha = 0.5, \rho = 1.0]$). It is observed that $MIP_{2'}$ explored less number of branch and bound nodes in comparison with the other formulations. It can be noted that for the improved formulations $MIP_{1'}$, $MIP_{2'}$ and $MIP_{3'}$, CPLEX is able to produce optimal solutions in less computational time in comparison with the original formulations. Based on formulations $MIP_{4'}$ and $MIP_{4''}$, CPLEX is able to produce optimal solutions in less computational time in comparison with MIP_4 for only $[\alpha = 0.1, \rho = 0.5]$ and $[\alpha = 0.5, \rho = 0.7]$. In addition, the improved formulations $MIP_{1'}$, $MIP_{3'}$, $MIP_{4'}$ and $MIP_{4''}$ produced better LP relaxation lower bound estimates. For $n = 10$ and $m = 4$ (Table 2.4): for all formulations, CPLEX is able to produce an optimal solution for any instance. The best overall performance is demonstrated by: MIP_{KL1} for $[\alpha = 0.1, \rho = 0.5]$ with: $t(s) = 0.77$, MIP_{KL2} for $[\alpha = 0.1, \rho = 1.0]$ with: $t(s) = 0.49$, $MIP_{1'}$ for $[\alpha = 0.3, \rho = 0.5]$ with: $t(s) = 0.71$, MIP_2 for $[\alpha = 0.3, \rho = 0.7]$ with: $t(s) = 0.85$, and $MIP_{2'}$ for $[\alpha = 0.5, \rho = 0.7]$ and $[\alpha = 0.5, \rho = 1.0]$. Based on formulations $MIP_{1'}$, $MIP_{2'}$, $MIP_{3'}$, $MIP_{4'}$ and $MIP_{4''}$, CPLEX is able to produce optimal solutions in less computational time in comparison with the original formulations. In addition, the improved formulations $MIP_{1'}$, $MIP_{3'}$, $MIP_{4'}$ and $MIP_{4''}$ reduced significantly the value of $gap_{lp}(\%)$. For $n = 20$ and $m = 3$ (Table 2.5): MIP_2 and $MIP_{2'}$ are the only formulations for which CPLEX is able to produce an optimal solution for any instance, except for one instance for $[\alpha = 0.5, \rho = 1.0]$. MIP_1, $MIP_{1'}$, MIP_3, $MIP_{3'}$, MIP_4, $MIP_{4'}$ and $MIP_{4''}$ are the formulations for which CPLEX is able to produce a feasible solution at best. CPLEX is able to find an optimal solution for any instance for MIP_{KL1} and MIP_{KL2} for only $[\alpha = 0.1\ ,\ \rho = 0.5\ ]$. Based on formulation $MIP_{2'}$, CPLEX is able to produce optimal solutions in less computational time in comparison with MIP_2. It can be observed that the improved formulations $MIP_{1'}$, $MIP_{3'}$, $MIP_{4'}$ and $MIP_{4''}$ reduced significantly the value of $gap_1(\%)$, $gap_2(\%)$ and $gap_{lp}(\%)$. For $n = 50$ and $m = 3$ (Table 2.6): MIP_2 and $MIP_{2'}$ are the only formulations for which CPLEX is able to produce an optimal solution for any instance of $[\alpha = 0.1\ ,\ \rho = 0.5\ ]$. For instances with $[\alpha = 0.5\ ,\ \rho = 0.5\ ]$, $MIP_{2'}$ is the only formulation for which CPLEX is able to produce an optimal solution for 5 instances. In addition, based on formulation MIP_2, CPLEX is able to produce an

optimal solution for 4 instances for [$\alpha = 0.5$, $\rho = 0.5$]. The improved formulations $MIP_{1'}$, $MIP_{3'}$, $MIP_{4'}$ and $MIP_{4''}$ reduced the value of $gap_1(\%)$ and $gap_2(\%)$. Moreover, the improved formulations $MIP_{1'}$, $MIP_{3'}$, $MIP_{4'}$ and $MIP_{4''}$ produced better LP relaxation lower bound estimates. For $n = 50$ and $m = 7$ (Table 2.6): MIP_2 and $MIP_{2'}$ are the only formulations for which CPLEX is able to produce an optimal solution for any instance. Based on formulation $MIP_{2'}$, CPLEX is able to find an optimal solution for any instance in less than 1627.57 seconds, with: $t(s) = 363.98$ and $\#BB = 1389$. Moreover, the improved formulations $MIP_{1'}$ and $MIP_{3'}$ reduced significantly the value of $gap_1(\%)$, $gap_2(\%)$ and $gap_{lp}(\%)$. For $n = 100$ and $m = 5$ (Table 2.7): $MIP_{2'}$ is the only formulation for which CPLEX is able to produce an optimal solution for 5 instances with: $t(s) = 3065.78$, $gap_1(\%) = 35.02$, $gap_2(\%) = 35.02$, $gap_{lp}(\%) = 47.31$ and $\#BB = 782$. Based on formulation MIP_4, CPLEX is not able to find a feasible solution for any instance. Based on $MIP_{4'}$, CPLEX is able to find a feasible solution for all instances at best. In addition, based on formulation $MIP_{4''}$, CPLEX is not able to find a feasible solution for only one instance. It can be observed that the improved formulations $MIP_{1'}$ and $MIP_{3'}$ produced much smaller $gap_1(\%)$, $gap_2(\%)$ and $gap_{lp}(\%)$ in comparison with $MIP_{2'}$.

The overall results show that the performance of all formulations is related to the number of jobs and the number of machines. It can also be noticed the positive impact of the proposed strengthening constraints (2.65) and (2.66), as indeed $MIP_{1'}$ and $MIP_{3'}$ produced LP relaxation bounds better than all other formulations for all instances. In addition, $MIP_{1'}$ and $MIP_{3'}$ produced similar LP relaxation bounds for all instances. While, the formulations MIP_1, MIP_3 and MIP_4 produced similar LP relaxation bounds for the majority of instances. Moreover, the best performance in terms of computational time and the number of instances solved to optimality is observed by $MIP_{2'}$. $MIP_{2'}$ is the only formulation for which CPLEX is able to produce an optimal solution for 5 instances for $n = 100$ and $m = 5$. From a formulation size point of view (memory requirements), the main disadvantage of MIP_2 is its size (the number of variables and the number of constraints are $O(nT)$). Unlike the other formulations, the number of variables and constraints of MIP_2 depend on the length of the time horizon. The number of constraints and variables can be very large, even for small instances if jobs length $(s_i + p_i)$ is relatively large. In addition, the LP relaxation bounds of MIP_2 took more computational time to be solved as compared to the other formulations. This is the challenge in dealing with time-indexed variables formulation for solving scheduling problems. Van den Akker et al. [101] discuss how Dantzig-Wolfe decomposition techniques can be applied to overcome the difficulties associated with the size of time-indexed formulation

for solving a single machine scheduling problem.

Table 2.3: Comparison of all formulations for $n = 10$ and $m = 3$ – general job set

		$[n = 10, m = 3]$					
		$[\alpha, \rho]$					
		[0.1,0.5]	[0.1,1.0]	[0.3,0.5]	[0.3,0.7]	[0.5,0.7]	[0.5,1.0]
MIP_1	$\#opt$ $[t(s)]$	2 [2784.37]	8 [1285.30]	10 [449.39]	10 [499.00]	10 [66.26]	10 [36.49]
	$\#is$ $[gap_1(\%)]$	8 [13.84]	2 [10.01]	0	0	0	0
	$gap_2(\%)$ $[gap_{lp}(\%)]$	13.84 [71.14]	10.01 [71.24]	0 [62.74]	0 [64.77]	0 [59.75]	0 [60.15]
	$\#BB$	4541085	2807085	867630	797836	163482	91892
$MIP_{1'}$	$\#opt$ $[t(s)]$	10 [0.82]	10 [1.21]	10 [1.96]	10 [2.90]	10 [2.31]	10 [7.81]
	$\#is$ $[gap_1(\%)]$	0	0	0	0	0	0
	$gap_2(\%)$ $[gap_{lp}(\%)]$	0 [11.73]	0 [11.68]	0 [3.46]	0 [5.26]	0 [3.07]	0 [5.87]
	$\#BB$	661	2384	5016	11189	8492	28192
MIP_2	$\#opt$ $[t(s)]$	10 [0.94]	10 [1.20]	10 [1.15]	10 [1.31]	10 [1.51]	10 [3.95]
	$\#is$ $[gap_1(\%)]$	0	0	0	0	0	0
	$gap_2(\%)$ $[gap_{lp}(\%)]$	0 [39.40]	0 [36.65]	0 [34.18]	0 [33.78]	0 [33.02]	0 [32.88]
	$\#BB$	0	0	0	0	0	313
$MIP_{2'}$	$\#opt$ $[t(s)]$	**10 [0.62]**	10 [0.76]	**10 [0.91]**	10 [0.92]	**10 [1.37]**	10 [2.44]
	$\#is$ $[gap_1(\%)]$	**0**	0	**0**	0	**0**	0
	$gap_2(\%)$ $[gap_{lp}(\%)]$	**0 [39.42]**	0 [36.71]	**0 [34.37]**	0 [33.99]	**0 [33.17]**	0 [33.07]
	$\#BB$	**0**	0	**0**	0	**0**	181
MIP_3	$\#opt$ $[t(s)]$	0	7 [1963.63]	9 [1163.77]	10 [649.13]	10 [120.09]	10 [66.54]
	$\#is$ $[gap_1(\%)]$	10 [14.68]	3 [5.74]	1 [8.57]	0	0	0
	$gap_2(\%)$ $[gap_{lp}(\%)]$	14.68 [71.14]	5.74 [71.24]	8.57 [62.74]	0 [64.77]	0 [59.75]	0 [60.15]
	$\#BB$	7231413	3480641	2190504	1029542	273291	177009
$MIP_{3'}$	$\#opt$ $[t(s)]$	10 [0.76]	10 [3.13]	10 [18.37]	10 [19.18]	10 [21.51]	10 [42.34]
	$\#is$ $[gap_1(\%)]$	0	0	0	0	0	0
	$gap_2(\%)$ $[gap_{lp}(\%)]$	0 [11.73]	0 [11.68]	0 [3.46]	0 [5.26]	0 [3.07]	0 [5.87]
	$\#BB$	1021	10041	44477	62130	60630	112117
MIP_4	$\#opt$ $[t(s)]$	10 [258.67]	10 [193.44]	10 [124.92]	10 [134.19]	10 [101.50]	10 [156.15]
	$\#is$ $[gap_1(\%)]$	0	0	0	0	0	0
	$gap_2(\%)$ $[gap_{lp}(\%)]$	0 [71.14]	0 [71.24]	0 [62.74]	0 [64.77]	0 [59.75]	0 [60.15]
	$\#BB$	353892	353503	256858	269729	206617	298517
$MIP_{4'}$	$\#opt$ $[t(s)]$	10 [153.98]	10 [526.54]	10 [134.04]	10 [220.79]	10 [93.38]	10 [239.29]
	$\#is$ $[gap_1(\%)]$	0	0	0	0	0	0
	$gap_2(\%)$ $[gap_{lp}(\%)]$	0 [55.50]	0 [55.49]	0 [52.47]	0 [53.24]	0 [53.24]	0 [54.38]
	$\#BB$	305333	407993	222020	255127	181906	265658
$MIP_{4''}$	$\#opt$ $[t(s)]$	10 [228.05]	10 [277.68]	10 [126.05]	10 [182.89]	10 [125.50]	10 [171.84]
	$\#is$ $[gap_1(\%)]$	0	0	0	0	0	0
	$gap_2(\%)$ $[gap_{lp}(\%)]$	0 [55.50]	0 [55.49]	0 [52.47]	0 [53.24]	0 [53.24]	0 [54.38]
	$\#BB$	283770	368730	214579	250516	180507	263574
MIP_{KL1}	$\#opt$ $[t(s)]$	10 [0.70]	10 [0.56]	10 [1.92]	10 [1.70]	10 [3.56]	10 [2.45]
	$\#is$ $[gap_1(\%)]$	0	0	0	0	0	0
	$gap_2(\%)$ $[gap_{lp}(\%)]$	0 [45.52]	0 [16.66]	0 [43.30]	0 [29.85]	0 [35.95]	0 [16.95]
	$\#BB$	2637	1715	5638	5175	13307	8322
MIP_{KL2}	$\#opt$ $[t(s)]$	10 [0.64]	**10 [0.47]**	10 [1.00]	**10 [0.77]**	10 [2.59]	**10 [1.79]**
	$\#is$ $[gap_1(\%)]$	0	**0**	0	**0**	0	**0**
	$gap_2(\%)$ $[gap_{lp}(\%)]$	0 [68.44]	**0 [47.90]**	0 [69.50]	**0 [63.40]**	0 [71.91]	**0 [56.04]**
	$\#BB$	673	**277**	2372	**1341**	11521	**6213**

Table 2.4: Comparison of all formulations for $n = 10$ and $m = 4$ – general job set

		$[n = 10, m = 4]$					
		$[\alpha, \rho]$					
		[0.1,0.5]	[0.1,1.0]	[0.3,0.5]	[0.3,0.7]	[0.5,0.7]	[0.5,1.0]
MIP_1	$\#opt$ $[t(s)]$	10 [210.59]	10 [440.49]	10 [73.61]	10 [130.81]	10 [15.73]	10 [18.62]
	$\#is$ $[gap_1(\%)]$	0	0	0	0	0	0
	$gap_2(\%)$ $[gap_{lp}(\%)]$	0 [61.73]	0 [64.29]	0 [54.54]	0 [56.41]	0 [46.17]	0 [49.64]
	$\#BB$	496473	1092389	218916	325602	50814	60440
$MIP_{1'}$	$\#opt$ $[t(s)]$	10 [1.96]	10 [29.00]	**10 [0.71]**	10 [1.17]	10 [1.41]	10 [9.27]
	$\#is$ $[gap_1(\%)]$	0	0	**0**	0	0	0
	$gap_2(\%)$ $[gap_{lp}(\%)]$	0 [14.04]	0 [17.44]	**0 [8.36]**	0 [10.35]	0 [6.32]	0 [10.63]
	$\#BB$	6327	130189	**1321**	3913	3288	26324
MIP_2	$\#opt$ $[t(s)]$	10 [0.79]	10 [1.04]	10 [0.87]	**10 [0.85]**	10 [0.98]	10 [1.86]
	$\#is$ $[gap_1(\%)]$	0	0	0	**0**	0	0
	$gap_2(\%)$ $[gap_{lp}(\%)]$	0 [35.08]	0 [33.52]	0 [30.57]	**0 [30.57]**	0 [27.51]	0 [27.29]
	$\#BB$	0	0	0	**0**	0	0
$MIP_{2'}$	$\#opt$ $[t(s)]$	10 [1.16]	10 [0.71]	10 [0.83]	10 [0.90]	**10 [0.94]**	**10 [1.53]**
	$\#is$ $[gap_1(\%)]$	0	0	0	0	**0**	**0**
	$gap_2(\%)$ $[gap_{lp}(\%)]$	0 [35.11]	0 [33.55]	0 [30.74]	0 [30.73]	**0 [27.67]**	**0 [27.32]**
	$\#BB$	0	0	**0**	0	**0**	**2**
MIP_3	$\#opt$ $[t(s)]$	10 [905.87]	10 [963.37]	10 [464.06]	10 [399.94]	10 [70.03]	10 [67.57]
	$\#is$ $[gap_1(\%)]$	0	0	0	0	0	0
	$gap_2(\%)$ $[gap_{lp}(\%)]$	0 [61.73]	0 [64.29]	0 [54.54]	0 [56.41]	0 [46.17]	0 [49.64]
	$\#BB$	1758320	1638534	972845	830339	190830	200397
$MIP_{3'}$	$\#opt$ $[t(s)]$	10 [7.17]	10 [231.05]	10 [1.65]	10 [5.67]	10 [4.31]	10 [26.30]
	$\#is$ $[gap_1(\%)]$	0	0	0	0	0	0
	$gap_2(\%)$ $[gap_{lp}(\%)]$	0 [14.04]	0 [17.44]	0 [8.36]	0 [10.35]	0 [6.32]	0 [10.63]
	$\#BB$	15395	731237	3074	13142	13772	62240
MIP_4	$\#opt$ $[t(s)]$	10 [25.53]	10 [65.80]	10 [14.75]	10 [20.57]	10 [13.84]	10 [26.29]
	$\#is$ $[gap_1(\%)]$	0	0	0	0	0	0
	$gap_2(\%)$ $[gap_{lp}(\%)]$	0 [61.73]	0 [64.29]	0 [54.54]	0 [56.41]	0 [46.17]	0 [49.64]
	$\#BB$	54877	141172	34716	47935	35276	60339
$MIP_{4'}$	$\#opt$ $[t(s)]$	10 [18.23]	10 [73.06]	10 [12.39]	10 [16.86]	10 [10.01]	10 [23.67]
	$\#is$ $[gap_1(\%)]$	0	0	0	0	0	0
	$gap_2(\%)$ $[gap_{lp}(\%)]$	0 [45.99]	0 [47.88]	0 [43.81]	0 [44.74]	0 [43.41]	0 [46.11]
	$\#BB$	34963	147353	26554	34747	22721	51835
$MIP_{4''}$	$\#opt$ $[t(s)]$	10 [18.81]	10 [64.13]	10 [10.87]	10 [15.28]	10 [10.04]	10 [24.50]
	$\#is$ $[gap_1(\%)]$	0	0	0	0	0	0
	$gap_2(\%)$ $[gap_{lp}(\%)]$	0 [45.99]	0 [47.88]	0 [43.81]	0 [44.74]	0 [43.41]	0 [46.11]
	$\#BB$	34366	111961	22176	31688	22892	50898
MIP_{KL1}	$\#opt$ $[t(s)]$	**10 [0.77]**	10 [0.57]	10 [1.98]	10 [1.48]	10 [5.35]	10 [5.51]
	$\#is$ $[gap_1(\%)]$	**0**	0	0	0	0	0
	$gap_2(\%)$ $[gap_{lp}(\%)]$	**0 [39.00]**	0 [14.78]	0 [36.54]	0 [28.11]	0 [29.63]	0 [10.31]
	$\#BB$	**2780**	1538	6496	5851	22459	23912
MIP_{KL2}	$\#opt$ $[t(s)]$	10 [0.86]	**10 [0.49]**	10 [1.40]	10 [1.06]	10 [4.54]	10 [2.42]
	$\#is$ $[gap_1(\%)]$	0	**0**	0	0	0	0
	$gap_2(\%)$ $[gap_{lp}(\%)]$	0 [57.97]	**0 [37.29]**	0 [60.24]	0 [54.72]	0 [61.66]	0 [34.34]
	$\#BB$	1618	**423**	3554	2573	22649	8258

Table 2.5: Comparison of all formulations for $n = 20$ and $m = 3$ – general job set

		$[n = 20, m = 3]$					
		$[\alpha, \rho]$					
		[0.1,0.5]	[0.1,1.0]	[0.3,0.5]	[0.3,0.7]	[0.5,0.7]	[0.5,1.0]
MIP_1	$\#opt$ $[t(s)]$	0	0	0	0	0	0
	$\#is$ $[gap_1(\%)]$	10 [70.60]	10 [69.98]	10 [68.12]	10 [67.52]	10 [63.45]	10 [63.31]
	$gap_2(\%)$ $[gap_{lp}(\%)]$	70.53 [84.15]	69.87 [84.33]	68.09 [81.34]	67.35 [81.46]	63.33 [79.04]	62.92 [79.50]
	$\#BB$	2918445	3165554	3014313	3202445	3275162	3404706
$MIP_{1'}$	$\#opt$ $[t(s)]$	0	0	0	0	0	0
	$\#is$ $[gap_1(\%)]$	10 [2.20]	10 [5.29]	10 [1.30]	10 [2.41]	10 [1.43]	10 [3.04]
	$gap_2(\%)$ $[gap_{lp}(\%)]$	2.10 [2.81]	4.99 [5.01]	1.25 [1.25]	2.17 [2.17]	1.23 [1.23]	2.15 [2.15]
	$\#BB$	999548	4889943	1214444	2367975	1394831	3716483
MIP_2	$\#opt$ $[t(s)]$	10 [11.69]	10 [30.66]	10 [23.84]	10 [34.47]	10 [42.67]	9 [188.03]
	$\#is$ $[gap_1(\%)]$	0	0	0	0	0	1 [0.96]
	$gap_2(\%)$ $[gap_{lp}(\%)]$	0 [42.53]	0 [42.32]	0 [41.97]	0 [41.57]	0 [41.57]	0 [41.23]
	$\#BB$	0	42	134	113	416	13151
$MIP_{2'}$	$\#opt$ $[t(s)]$	**10 [7.82]**	**10 [28.19]**	**10 [14.10]**	**10 [27.08]**	**10 [38.85]**	**9 [146.81]**
	$\#is$ $[gap_1(\%)]$	**0**	**0**	**0**	**0**	**0**	**1 [0.96]**
	$gap_2(\%)$ $[gap_{lp}(\%)]$	**0 [42.55]**	**0 [42.35]**	**0 [42.01]**	**0 [41.61]**	**0 [41.65]**	**0 [41.31]**
	$\#BB$	**2**	**399**	**98**	**469**	**487**	**18592**
MIP_3	$\#opt$ $[t(s)]$	0	0	0	0	0	0
	$\#is$ $[gap_1(\%)]$	10 [79.39]	10 [74.72]	10 [76.68]	10 [73.00]	10 [69.30]	10 [68.36]
	$gap_2(\%)$ $[gap_{lp}(\%)]$	79.28 [84.15]	74.62 [84.33]	76.58 [81.34]	72.78 [81.46]	69.14 [79.04]	67.79 [79.50]
	$\#BB$	1441832	1428259	1586847	1415031	1820008	1574601
$MIP_{3'}$	$\#opt$ $[t(s)]$	0	0	0	0	0	0
	$\#is$ $[gap_1(\%)]$	10 [3.06]	10 [5.29]	10 [1.25]	10 [2.36]	10 [1.33]	10 [3.15]
	$gap_2(\%)$ $[gap_{lp}(\%)]$	2.72 [2.81]	4.96 [5.01]	1.25 [1.25]	2.17 [2.17]	1.23 [1.23]	2.15 [2.15]
	$\#BB$	596910	4511115	1299083	2385939	1578972	2477000
MIP_4	$\#opt$ $[t(s)]$	0	0	0	0	0	0
	$\#is$ $[gap_1(\%)]$	10 [72.50]	10 [73.46]	10 [73.84]	10 [74.30]	10 [73.30]	10 [71.44]
	$gap_2(\%)$ $[gap_{lp}(\%)]$	72.36 [84.15]	73.35 [84.33]	73.75 [81.34]	74.07 [81.46]	73.14 [79.04]	70.95 [79.50]
	$\#BB$	3053455	4991840	3123188	3428997	3325607	4402895
$MIP_{4'}$	$\#opt$ $[t(s)]$	0	0	0	0	0	0
	$\#is$ $[gap_1(\%)]$	10 [70.55]	10 [69.71]	10 [70.32]	10 [70.28]	10 [70.51]	10 [70.27]
	$gap_2(\%)$ $[gap_{lp}(\%)]$	70.45 [73.21]	69.57 [73.83]	70.19 [73.09]	70.12 [73.38]	70.35 [73.36]	69.67 [73.57]
	$\#BB$	4279709	3288824	4276858	3723524	3811707	3487254
$MIP_{4''}$	$\#opt$ $[t(s)]$	0	0	0	0	0	0
	$\#is$ $[gap_1(\%)]$	10 [70.69]	10 [69.75]	10 [70.42]	10 [70.27]	10 [70.49]	10 [70.45]
	$gap_2(\%)$ $[gap_{lp}(\%)]$	70.58 [73.21]	69.62 [73.83]	70.37 [73.09]	70.13 [73.37]	70.27 [73.37]	69.69 [73.58]
	$\#BB$	3634665	3456765	4159405	3839102	3750006	2727108
MIP_{KL1}	$\#opt$ $[t(s)]$	10 [932.64]	9 [159.92]	0	0	0	0
	$\#is$ $[gap_1(\%)]$	0	1 [0.22]	10 [19.12]	10 [9.70]	10 [28.11]	10 [12.03]
	$gap_2(\%)$ $[gap_{lp}(\%)]$	0 [52.30]	0.22 [20.85]	19.10 [53.76]	9.70 [36.96]	28.11 [44.98]	11.95 [22.21]
	$\#BB$	1156072	1911254	3284582	1200824	2764436	2229967
MIP_{KL2}	$\#opt$ $[t(s)]$	10 [676.50]	8 [42.02]	0	0	0	1 [901.42]
	$\#is$ $[gap_1(\%)]$	0	2 [0.56]	10 [21.60]	10 [9.65]	10 [56.07]	9 [31.32]
	$gap_2(\%)$ $[gap_{lp}(\%)]$	0 [82.04]	0.56 [68.33]	21.60 [85.45]	9.65 [81.06]	56.07 [88.00]	31.25 [77.94]
	$\#BB$	451600	5316506	978771	1402425	572855	911313

2.7.3 Comparison of formulations – regular job set

This section presents the computational results for the benchmark instances with a regular job set. We evaluate the performance of the proposed mathe-

matical formulation MIP_5 (2.70)-(2.75),(2.77)-(2.80) and we compare his performance with the formulations $MIP_{1'}$, $MIP_{2'}$, $MIP_{3'}$, $MIP_{4'}$, $MIP_{4''}$, MIP_{KL1} and MIP_{KL2}. The regular job set is composed of instances with: $(n, m) \in \{(20, 3); (40, 3); (40, 6); (100, 3); (250, 4); (350, 3); (350, 5); (500, 5)\}$. For each value of the combination (n, m), 10 instances are generated randomly, resulting in a total of 80 instances.

These results are detailed in Table 2.8:

For $n = 20$ and $m = 3$: MIP_5, MIP_{KL1} and MIP_{KL2} are the only formulations for which CPLEX is able to produce an optimal solution for any instance. Based on formulation MIP_5, CPLEX is able to find an optimal solution for any instance in less than 0.67 seconds, based on formulation MIP_{KL2}, CPLEX is able to find an optimal solution for any instance in less than 3.64 seconds and based on formulation MIP_{KL1}, CPLEX is able to find an optimal solution for any instance in less than 13.96 seconds. CPLEX is able to find an optimal solution for only 5 instances for $MIP_{2'}$ in less than 2464.29 seconds. It can be noticed that CPLEX is not able to produce a feasible solution for 14 instances for $MIP_{4'}$ and $MIP_{4''}$. Based on formulation $MIP_{3'}$, CPLEX is not able to find a feasible solution for 8 instances. Moreover, the LP relaxation of MIP_5, MIP_{KL1} and MIP_{KL2} reached an optimal solution for all instances. The best performance is demonstrated by MIP_5, with: $t(s) = 0.41$ and $\#BB = 25$. For $n = 40$ and $m = 3$: MIP_5, MIP_{KL1} and MIP_{KL2} are the only formulations for which CPLEX is able to produce an optimal solution for any instance. CPLEX is not able to find a feasible solution for any instance for $MIP_{1'}$, $MIP_{4'}$ and $MIP_{4''}$. CPLEX is able to find a feasible solution for any instance for $MIP_{2'}$ and $MIP_{3'}$ at best. It can be noticed that the LP relaxation of MIP_5, MIP_{KL1} and MIP_{KL2} reached an optimal solution for all instances. The best performance is demonstrated by MIP_5, with: $t(s) = 0.84$ and $\#BB = 28$. For $n = 40$ and $m = 6$: MIP_5, MIP_{KL1} and MIP_{KL2} are the only formulations for which CPLEX is able to produce an optimal solution for any instance. CPLEX is not able to find a feasible solution for $MIP_{1'}$, $MIP_{4'}$ and $MIP_{4''}$ for any instance. Based on formulations $MIP_{2'}$ and $MIP_{3'}$, CPLEX is able to produce a feasible solution for any instance at best. In addition, it can be noticed that the LP relaxation of MIP_5, MIP_{KL1} and MIP_{KL2} reached the optimal solution for all instances. The best performance is demonstrated by MIP_5, with: $t(s) = 0.34$ and $\#BB = 0$. For $n = 100$ and $m = 3$: based on formulation MIP_5, CPLEX is able to find an optimal solution for any instance in less than 6.85 seconds, based on formulation MIP_{KL2}, CPLEX is able to find an optimal solution for any instance in less than 583.44 seconds and based on formulation MIP_{KL1}, CPLEX is able to find an optimal solution for any instance in less than 278.33 seconds. CPLEX is not able to find a feasible

solution for any instance for $MIP_{1'}$, $MIP_{2'}$, $MIP_{4'}$ and $MIP_{4''}$. In addition, based on formulation $MIP_{2'}$, CPLEX is able to produce a feasible solution for any instance at best. Moreover, it can be seen that the LP relaxation of MIP_5, MIP_{KL1} and MIP_{KL2} reached an optimal solution for all instances. The best performance is demonstrated by MIP_5, with: $t(s) = 3.28$. For $n = 250$ and $m = 4$: MIP_5 and MIP_{KL1} are the only formulations for which CPLEX is able to produce an optimal solution for any instance. It can be noticed that CPLEX is able to find an optimal solution for any instance for MIP_5 in less than 23.85 seconds. CPLEX produced an optimal solution for any instance for MIP_{KL1} in less than 1594.34 seconds. Based on formulations $MIP_{2'}$ and $MIP_{3'}$, CPLEX is not able to solve the LP relaxation within 1 hour. In addition, the LP relaxation of MIP_5 and MIP_{KL1} reached the optimal solution for all instances. The best performance is demonstrated by MIP_5, with: $t(s) = 16.68$ and $\#BB = 0$. For $n = 350$ and $m = 3$: MIP_5 is the only formulation for which CPLEX is able to produce an optimal solution for any instance. Based on formulation MIP_{KL1}, CPLEX is able to produce a feasible solution for all instances at best, with: $gap_{lp}(\%) = 0$. In addition, for all the remaining formulations, CPLEX is not able to produce a feasible solution for any instance. It can be observed also that based on formulations $MIP_{2'}$, $MIP_{3'}$ and MIP_{KL2}, CPLEX is not able to solve the LP relaxation within 1 hour. The best performance is demonstrated by MIP_5, with: $t(s) = 189.08$ and $gap_{lp}(\%) = \#BB = 0$. For $n = 350$ and $m = 5$: MIP_5 is the only formulation for which CPLEX is able to produce an optimal solution for any instance, with: $t(s) = 57.40$ and $gap_{lp}(\%) = \#BB = 0$. Based on formulation MIP_{KL1}, CPLEX is able to produce a feasible solution for all instances at best, with: $\#BB = 0$. The best performance is demonstrated by MIP_5. For $n = 500$ and $m = 5$: MIP_5 is the only formulation for which CPLEX is able to produce an optimal solution for any instance in less than 316.21 seconds, with: $t(s) = 191.58$ and $gap_{lp}(\%) = \#BB = 0$. In addition, for all other formulations, CPLEX is not able to produce a feasible solution for any instance. In addition, based on formulations $MIP_{2'}$, $MIP_{3'}$ and MIP_{KL2}, CPLEX is not able to solve the LP relaxation for all instances.

The overall results show that MIP_5 has a better performance, being able to deal with instances containing up to 500 jobs and 5 machines in less than 5.27 minutes. It can be noticed that the structure of MIP_5 allowed a significant reduction of the number of constraints. Indeed, the number of constraints of $RAPV$ is $O(n)$, while the number of constraints for all other formulations except TIV is $O(n^2)$. In addition, the LP relaxation of MIP_5 reached an optimal solution for all instances. In sum, MIP_5 is the best formulation to solve a regular job set of the $P, S1|p_j, s_j|C_{max}$.

Table 2.6: Comparison of all formulations for $n = 50$ with $m = 3$ and $m = 7$ – general job set

		$[n = 50, m = 3]$		$[n = 50, m = 7]$	
		$[\alpha, \rho]$		$[\alpha, \rho]$	
		$[0.1, 0.5]$	$[0.5, 0.5]$	$[0.1, 0.7]$	$[0.3, 1.0]$
MIP_1	$\#opt$ $[t(s)]$	0	0	0	0
	$\#is$ $[gap_1(\%)]$	10 [91.99]	10 [89.38]	10 [84.67]	10 [81.76]
	$gap_2(\%)$ $[gap_{lp}(\%)]$	91.89 [93.42]	89.22 [91.17]	83.97 [85.35]	80.30 [82.54]
	$\#BB$	230029	263728	161045	169539
$MIP_{1'}$	$\#opt$ $[t(s)]$	0	0	0	0
	$\#is$ $[gap_1(\%)]$	10 [1.43]	10 [1.26]	10 [7.37]	10 [8.28]
	$gap_2(\%)$ $[gap_{lp}(\%)]$	0.78 [0.78]	0.51 [0.51]	4.42 [4.42]	3.16 [3.16]
	$\#BB$	287773	383377	155583	165606
MIP_2	$\#opt$ $[t(s)]$	10 [1497.79]	4 [3078.89]	10 [116.89]	10 [712.22]
	$\#is$ $[gap_1(\%)]$	0	6 [29.38]	0	0
	$gap_2(\%)$ $[gap_{lp}(\%)]$	0 [46.77]	29.38 [46.63]	0 [42.49]	0 [41.69]
	$\#BB$	1583	3153	66	2259
$MIP_{2'}$	$\#opt$ $[t(s)]$	**10 [1055.80]**	**5 [2045.87]**	**10 [92.16]**	**10 [635.81]**
	$\#is$ $[gap_1(\%)]$	**0**	**5 [29.79]**	**0**	**0**
	$gap_2(\%)$ $[gap_{lp}(\%)]$	**0 [46.77]**	**29.70 [46.64]**	**0 [42.49]**	**0 [41.69]**
	$\#BB$	**890**	**1983**	**436**	**2342**
MIP_3	$\#opt$ $[t(s)]$	0	0	0	0
	$\#is$ $[gap_1(\%)]$	10 [92.62]	10 [93.02]	10 [88.67]	10 [82.21]
	$gap_2(\%)$ $[gap_{lp}(\%)]$	92.49 [93.42]	90.18 [91.17]	84.34 [85.35]	80.62 [82.54]
	$\#BB$	634123	1467	54145	121224
$MIP_{3'}$	$\#opt$ $[t(s)]$	0	0	0	0
	$\#is$ $[gap_1(\%)]$	10 [2.66]	10 [20.74]	10 [20.53]	10 [8.93]
	$gap_2(\%)$ $[gap_{lp}(\%)]$	0.78 [0.78]	0.51 [0.51]	4.43 [4.42]	3.16 [3.16]
	$\#BB$	193273	2188	8622	100755
MIP_4	$\#opt$ $[t(s)]$	0	0	0	0
	$\#is$ $[gap_1(\%)]$	10 [91.70]	10 [89.57]	10 [83.84]	10 [80.81]
	$gap_2(\%)$ $[gap_{lp}(\%)]$	91.61 [93.42]	89.37 [91.17]	83.35 [85.35]	79.73 [82.54]
	$\#BB$	564714	864214	816914	911233
$MIP_{4'}$	$\#opt$ $[t(s)]$	0	0	0	0
	$\#is$ $[gap_1(\%)]$	10 [88.14]	10 [88.12]	10 [74.44]	10 [75.21]
	$gap_2(\%)$ $[gap_{lp}(\%)]$	88.05 [88.49]	87.95 [88.58]	73.56 [75.29]	73.85 [75.25]
	$\#BB$	429535	525535	302447	249648
$MIP_{4''}$	$\#opt$ $[t(s)]$	0	0	0	0
	$\#is$ $[gap_1(\%)]$	10 [88.18]	10 [88.03]	10 [74.43]	10 [75.09]
	$gap_2(\%)$ $[gap_{lp}(\%)]$	88.07 [88.49]	87.88 [88.57]	73.68 [75.26]	73.90 [75.25]
	$\#BB$	332696	492611	185806	282841
MIP_{KL1}	$\#opt$ $[t(s)]$	0	0	0	0
	$\#is$ $[gap_1(\%)]$	10 [56.95]	10 [54.98]	10 [42.11]	10 [25.79]
	$gap_2(\%)$ $[gap_{lp}(\%)]$	56.00 [58.01]	54.24 [46.64]	38.38 [42.49]	19.64 [19.64]
	$\#BB$	141264	140708	623099	134116
MIP_{KL2}	$\#opt$ $[t(s)]$	0	0	0	0
	$\#is$ $[gap_1(\%)]$	10 [79.44]	10 [95.53]	10 [79.44]	10 [74.33]
	$gap_2(\%)$ $[gap_{lp}(\%)]$	79.28 [92.55]	95.50 [95.53]	77.55 [77.55]	70.05 [70.05]
	$\#BB$	549349	61354	553750	240874

Table 2.7: Comparison of all formulations for $n = 100$ with $m = 5$ – general job set

		$[n = 100, m = 5]$
		$[\alpha, \rho]$
		$[0.1, 0.5]$
MIP_1	$\#opt$ $[t(s)]$	0
	$\#is$ $[gap_1(\%)]$	10 [94.47]
	$gap_2(\%)$ $[gap_{lp}(\%)]$	94.08 [94.58]
	$\#BB$	62631
$MIP_{1'}$	$\#opt$ $[t(s)]$	0
	$\#is$ $[gap_1(\%)]$	10 [3.32]
	$gap_2(\%)$ $[gap_{lp}(\%)]$	1.06 [1.06]
	$\#BB$	62865
$MIP_{2'}$	$\#opt$ $[t(s)]$	0
	$\#is$ $[gap_1(\%)]$	10 [39.06]
	$gap_2(\%)$ $[gap_{lp}(\%)]$	38.05 [47.30]
	$\#BB$	221
$MIP_{2'}$	$\#opt$ $[t(s)]$	**5 [3065.78]**
	$\#is$ $[gap_1(\%)]$	**5 [35.02]**
	$gap_2(\%)$ $[gap_{lp}(\%)]$	**35.02 [47.31]**
	$\#BB$	**782**
MIP_3	$\#opt$ $[t(s)]$	0
	$\#is$ $[gap_1(\%)]$	10 [96.81]
	$gap_2(\%)$ $[gap_{lp}(\%)]$	94.23 [94.58]
	$\#BB$	6179
$MIP_{3'}$	$\#opt$ $[t(s)]$	0
	$\#is$ $[gap_1(\%)]$	10 [33.38]
	$gap_2(\%)$ $[gap_{lp}(\%)]$	1.06 [1.06]
	$\#BB$	3453
MIP_4	$\#opt$ $[t(s)]$	$\star$
	$\#is$ $[gap_1(\%)]$	$\star$
	$gap_2(\%)$ $[gap_{lp}(\%)]$	$\star$ [95.88]
	$\#BB$	$\star$
$MIP_{4'}$	$\#opt$ $[t(s)]$	0
	$\#is$ $[gap_1(\%)]$	10 [92.50]
	$gap_2(\%)$ $[gap_{lp}(\%)]$	90.16 [90.38]
	$\#BB$	15317
$MIP_{4''}$	$\#opt$ $[t(s)]$	0
	$\#is$ $[gap_1(\%)]$	9 [91.22]
	$gap_2(\%)$ $[gap_{lp}(\%)]$	90.09 [90.38]
	$\#BB$	23032
MIP_{KL1}	$\#opt$ $[t(s)]$	0
	$\#is$ $[gap_1(\%)]$	10 [61.33]
	$gap_2(\%)$ $[gap_{lp}(\%)]$	58.51 [58.51]
	$\#BB$	153170
MIP_{KL2}	$\#opt$ $[t(s)]$	0
	$\#is$ $[gap_1(\%)]$	10 [93.83]
	$gap_2(\%)$ $[gap_{lp}(\%)]$	96.32 [93.43]
	$\#BB$	136270

Table 2.8: Comparison of all formulations – regular job set

n	m		MIP_5	$MIP_{1'}$	$MIP_{2'}$	$MIP_{3'}$	$MIP_{4'}$	$MIP_{4''}$	MIP_{KL1}	MIP_{KL2}
20	3	$\#opt$ $[t(s)]$	**10 [0.41]**	0	5 [2256.92]	0	0	0	10 [2.59]	10 [1.95]
		$\#is$ $[gap_1(\%)]$	**0**	5 [22.97]	5 [27.34]	2 [29.74]	3 [48.22]	3 [65.49]	0	0
		$gap_2(\%)$ $[gap_{lp}(\%)]$	**0 [0]**	21.66 [28.67]	26.62 [43.00]	32.61 [28.67]	48.22 [88.13]	64.98 [80.36]	0 [0]	0 [0]
		$\#BB$	**25**	3611596	1021	4338683	9065694	3300417	3139	793
40	3	$\#opt$ $[t(s)]$	**10 [0.84]**	$\star$	0	0	$\star$	$\star$	10 [11.09]	10 [7.84]
		$\#is$ $[gap_1(\%)]$	**0**	$\star$	10 [73.14]	10 [33.51]	$\star$	$\star$	0	0
		$gap_2(\%)$ $[gap_{lp}(\%)]$	**0 [0]**	$\star$ [32.65]	46.69 [46.69]	32.65 [32.65]	$\star$ [94.47]	$\star$ [90.31]	0 [0]	0 [0]
		$\#BB$	**28**	$\star$	0	46656	$\star$	$\star$	3812	446
	6	$\#opt$ $[t(s)]$	**10 [0.34]**	$\star$	0	0	$\star$	$\star$	10 [2.63]	10 [13.15]
		$\#is$ $[gap_1(\%)]$	**0**	$\star$	10 [72.37]	10 [67.60]	$\star$	$\star$	0	0
		$gap_2(\%)$ $[gap_{lp}(\%)]$	**0 [0]**	$\star$ [67.50]	46.86 [46.86]	67.50 [67.50]	$\star$ [94.62]	$\star$ [91.05]	0 [0]	0 [0]
		$\#BB$	**0**	$\star$	0	52464	$\star$	$\star$	25	0
100	3	$\#opt$ $[t(s)]$	**10 [3.28]**	$\star$	$\star$	0	$\star$	$\star$	10 [102.96]	10 [264.79]
		$\#is$ $[gap_1(\%)]$	**0**	$\star$	$\star$	10 [56.03]	$\star$	$\star$	0	0
		$gap_2(\%)$ $[gap_{lp}(\%)]$	**0 [0]**	$\star$ [33.27]	$\star$ [$\star$]	33.27 [33.27]	$\star$ [97.47]	$\star$ [96.07]	0 [0]	0 [0]
		$\#BB$	**28**	$\star$	$\star$	1	$\star$	$\star$	2444	2716
250	4	$\#opt$ $[t(s)]$	**10 [16.68]**	$\star$	$\star$	$\star$	$\star$	$\star$	10 [1131.31]	$\star$
		$\#is$ $[gap_1(\%)]$	**0**	$\star$	$\star$	$\star$	$\star$	$\star$	0	$\star$
		$gap_2(\%)$ $[gap_{lp}(\%)]$	**0 [0]**	$\star$ [50.32]	$\star$ [$\star$]	$\star$ [$\star$]	$\star$ [98.85]	$\star$ [98.41]	0 [0]	$\star$ [$\star$]
		$\#BB$	**0**	$\star$	$\star$	$\star$	$\star$	$\star$	677	$\star$
350	3	$\#opt$ $[t(s)]$	**10 [189.08]**	$\star$	$\star$	$\star$	$\star$	$\star$	0	$\star$
		$\#is$ $[gap_1(\%)]$	**0**	$\star$	$\star$	$\star$	$\star$	$\star$	10 [36.09]	$\star$
		$gap_2(\%)$ $[gap_{lp}(\%)]$	**0 [0]**	$\star$ [33.38]	$\star$ [$\star$]	$\star$ [$\star$]	$\star$ [99.18]	$\star$ [98.83]	27.87 [0]	$\star$ [$\star$]
		$\#BB$	**94**	$\star$	$\star$	$\star$	$\star$	$\star$	511	$\star$
	5	$\#opt$ $[t(s)]$	**10 [57.40]**	$\star$	$\star$	$\star$	$\star$	$\star$	0	$\star$
		$\#is$ $[gap_1(\%)]$	**0**	$\star$	$\star$	$\star$	$\star$	$\star$	10 [95.71]	$\star$
		$gap_2(\%)$ $[gap_{lp}(\%)]$	**0 [0]**	$\star$ [59.96]	$\star$ [$\star$]	$\star$ [$\star$]	$\star$ [99.18]	$\star$ [98.85]	91.42 [10.00]	$\star$ [$\star$]
		$\#BB$	**0**	$\star$	$\star$	$\star$	$\star$	$\star$	0	$\star$
500	5	$\#opt$ $[t(s)]$	**10 [191.58]**	$\star$	$\star$	$\star$	$\star$	$\star$	$\star$	$\star$
		$\#is$ $[gap_1(\%)]$	**0**	$\star$	$\star$	$\star$	$\star$	$\star$	$\star$	$\star$
		$gap_2(\%)$ $[gap_{lp}(\%)]$	**0 [0]**	$\star$ [59.74]	$\star$ [$\star$]	$\star$ [$\star$]	$\star$ [99.41]	$\star$ [99.18]	$\star$ [20.01]	$\star$ [$\star$]
		$\#BB$	**0**	$\star$	$\star$	$\star$	$\star$	$\star$	$\star$	$\star$

2.8 Concluding remarks

In this chapter, the problem of scheduling jobs on identical parallel machines with a single server to minimise the makespan, $P, S1|p_j, s_j|C_{max}$, has been studied. We present and solve the problem in two particular cases: a general job set and a regular job set. In the first case, we present mixed integer programming formulations to solve the problem, namely: completion time variables formulation (MIP_1), time-indexed variables formulation (MIP_2), linear ordering variables formulation (MIP_3), network variables formulation (MIP_4), and, we propose sets of inequalities that can be used to improve MIP_1, MIP_3 and MIP_4 formulations. In the second case, we present a new mixed integer programming formulation to solve the problem based on some properties. Extensive computational experiments on benchmark instances from the literature have been conducted and our results have been compared with the ones in Kim and Lee [62]. For the general case: by reducing the value of the time horizon T thanks to the approximate value of the makespan solution, $MIP_{2'}$ outperforms all formulations, being able to solve 5 instances containing up to 100 jobs and 5 machines. For the regular case: MIP_5 formulation proved to have a better performance, being able to deal with instances containing up to 500 jobs and 5 machines in a very short computational time. According to these results, we indicate the use of $MIP_{2'}$ formulation for a general job set and MIP_5 for a regular job set. In future research, it would be interesting to adapt the presented formulations to solve other machine scheduling problems with a single server and also take into consideration other types of objective functions such as: total tardiness and total weighted completion times.

Chapter 3

Approximate methods for the parallel machine scheduling problem with a single server to minimize the makespan

Contents

3.1 Introduction

The methods used to solve optimization problems (e.g., scheduling, routing, assignment, graph coloring, set covering, knapsack, etc.) can be divided on two main categories, namely : exact and approximate. The exact methods (e.g., branch-and-bound, branch-and-cut, branch-and-price, constraint programming and dynamic programming etc.) are not suitable for solving large scale problems to the proven optimality. Indeed, despite that they can find in some cases near optimal solutions in a short computational time, exact methods need a lot of time to prove their optimality. Therefore, there is a need for meta/heuristics able to find an approximate solution if possible of high quality in a short computational time, or sometimes an optimal solution but without proof of its optimality. This chapter considers the same problem than already discussed in Chapter 2 $(P, S1|p_j, s_j|C_{max})$, and presents its approximate resolution. Firstly, a lower bound is suggested and solution representation is discussed. Secondly, two complementary greedy heuristics are proposed to minimize respectively machine idle time and server waiting time which generalize already existing works. Indeed, the server waiting time is the total gap between the loading of all jobs, and, the machine idle time is the time resulting in simultaneous requests by machines for the server. Thirdly, a VNS metaheuristic is designed to solve the problem. The proposed algorithms were tested and compared with other algorithms proposed in the literature for small, medium and large-sized instances on a set of randomly generated instances from the literature. Over this set of instances, the computational results showed that our methods outperformed the methods in [3, 55]. The remainder of this chapter is organized as follows. A lower bound of the problem $P, S1|p_j, s_j|C_{max}$ is suggested in Section 3.2. Then, Section 3.3 discusses solution representation of the problem. Section 3.4 presents two greedy heuristics dedicated to solve small, medium and large-sized instances of the problem while Section 3.5 exhibits a metaheuristic based on VNS. Computational experiments are then presented in Section 3.6. Finally, Section 3.7 ends the chapter with conclusions and some perspectives.

3.2 Lower bound

The investigation of lower bound is useful for benchmarking meta/heuristic solutions. Thus, to evaluate the quality of the solutions, the following lower bound which we denote by LB is proposed for the problem $P, S1|p_j, s_j|C_{max}$.

$$LB = \max(LB1, LB2)$$

The description of $LB1$ and $LB2$ is presented in Proposition 9 and Proposition 10.

Proposition 9 *We consider a permutation σ of the jobs such that all jobs are ordered by their setup times $s_{\sigma(1)} \leq s_{\sigma(2)} \leq \ldots \leq s_{\sigma(n)}$, and $\overline{P} = \sum_{i \in N}(s_i + p_i)$.*

Then, we have:

$$LB1 = \frac{\overline{P} + \overline{J_p}}{m}$$

Where: $\overline{J_p} = (m-1)s_{\sigma(1)} + (m-2)s_{\sigma(2)} + (m-3)s_{\sigma(3)} + \ldots + s_{\sigma(m-1)}$

Proof Let C^*_{max} denotes the objective function value of an optimal solution of the problem $P, S1|p_j, s_j|C_{max}$.

If there is no machine idle time (i.e., the gap between the end of processing time and the start time of the setup operation of two jobs scheduled on the same machine is equal to zero) in an optimal schedule of the $P, S1|p_j, s_j|C_{max}$, then C^*_{max} will be equal to the sum of setup times plus the sum of processing times plus $\overline{J_p}$ which corresponds to:

$$(m-1)s_{\sigma(1)} + (m-2)s_{\sigma(2)} + (m-3)s_{\sigma(3)} + \ldots + s_{\sigma(m-1)}$$

divided by the number of machines m. Indeed, it is clear that each job from 1 to m must use a different machine. So, the first job in the schedule $\sigma(1)$ starts at time zero, the second job $\sigma(2)$ at $t = s_{\sigma(1)}$, the third job at $t = s_{\sigma(1)} + s_{\sigma(2)}$ and so on. Thus, each machine (except the first one) will be available only from a deadline $s_{\sigma(i)} > 0$. The fact of adding these deadlines with all the setup times and the processing times will constitute the total load to be executed by the m machines. It is then sufficient to divide

this charge by m to obtain the aforementioned lower bound.

Proposition 10

$$LB2 = \sum_{1 \leq i \leq n} s_i + \min_{1 \leq i \leq n} p_i$$

Proof If there is no server waiting time in an optimal schedule of the $P, S1|p_j, s_j|C_{max}$, then C^*_{max} will be equal to the sum of setup times plus the shortest processing time, and then inequality:

$$C^*_{max} \geq \sum_{1 \leq i \leq n} s_i + \min_{1 \leq i \leq n} p_i$$

will be always satisfied.

From Proposition 9 and Proposition 10, we have:

$$LB = \max(LB1, LB2) = \max\Big(\frac{\overline{P} + \overline{J_p}}{m}, \sum_{1 \leq i \leq n} s_i + \min_{1 \leq i \leq n} p_i\Big)$$

3.3 Solution representation and notation

A schedule of the problem $P, S1|p_j, s_j|C_{max}$ can be presented as a permutation $\pi = \{\pi_1, \ldots, \pi_k, \ldots, \pi_n\}$ of the set N, where π_k indicates the job which is processed in the k^{th} position. Any permutation of all jobs defines a feasible schedule, where each job will be scheduled if both a machine and the server are available simultaneously. Additional notation are defined as follows:

- $\pi = \{\pi_1, \pi_2, \ldots, \pi_n\}$: the list of jobs to be scheduled.
- $\pi' = \{\pi'_1, \pi'_2, \ldots, \pi'_n\}$: the scheduled list sequence obtained by one of the proposed heuristics.
- T_{π_k}: the start time of the setup operation of the job scheduled in position k.
- C_{π_k}: the completion time of the processing operation of the job scheduled in position k.
- E_j: the completion time of the last job scheduled on machine j.

- R_1: the rule that sort the jobs in non-increasing order of their processing times (LPT).
- R_2: the rule that sort the jobs in non-increasing order of their setup times (LST).
- R_3: the rule that sort the jobs in increasing order of their setup times (SST).

3.4 Greedy heuristics

In this section, we present two complementary greedy heuristics called HS1 and HS2 for the problem $P, S1|p_j, s_j|C_{max}$ under consideration in order to minimize respectively the server waiting time and the machine idle time. Indeed, it has been shown that the minimization of the total server waiting time (i.e., the gaps between the loading of two jobs) is equivalent to the minimization of the makespan (see Kim and Lee [62]). The basic aim at each step of the two heuristics is, if possible, not to generate any machine idle time or any server waiting time. Following the work proposed in [55] for the case of two machines, one can see that for the case of an arbitrary number of machines two types of optimal schedules can be found. The first one can be generated based on minimizing the machine idle time, and in this case $LB = LB1$. While, the second one tries to minimize the server waiting time, and in this case $LB = LB2$. In the following, we suggest the greedy heuristic HS1 for instances with $LB = LB1$ and heuristic HS2 for instances with $LB = LB2$.

3.4.1 Greedy heuristic HS1

The greedy heuristic HS1 aims to minimize machine idle time (the time machines are idle due to unavailability of the server). The pseudo-code of the proposed heuristic is summarized in Algorithm 1. In the first step, the initial list of jobs are arranged according to R_3 rule. In the second step, the first $m-1$ jobs of the list π are scheduled on the first available machines. After that, the remaining unscheduled jobs are sorted according to the rule R_2. In the third step, an eligible job is selected π_k (i.e., one that would not create any machine idle time). Hence, the selected job must verify the following Inequality 3.1:

$$E_j \geq \max\left(T_{\pi'_{i-1}} + s_{\pi'_{i-1}}, E_t\right) + s_{\pi_k} \tag{3.1}$$

Where E_t is the current available machine, E_j is the next available machine, and π'_{i-1} is

the job already scheduled in position $i-1$. If no eligible jobs exist, the first job in the remaining list is selected.

Illustrative example

Given an instance of $n = 10$ jobs and $m = 3$ machines. The processing times p_i and setup times of the jobs s_i are given in Table 3.1. The makespan value can be obtained by applying HS1 as given above. It takes 0.002 seconds to solve this instance. The feasible sequence is shown in Figure 3.1. The objective function (makespan) is equal to 53.

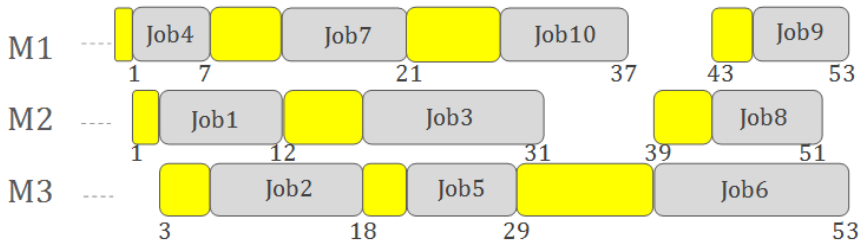

Figure 3.1: Feasible schedule for the considered instance with 10 jobs and 3 machines obtained by the heuristic HS1.

In this illustrative example, the initial list of jobs is sorted according to R_3 rule, then the first $m-1$ jobs are scheduled on the first available machines. The remaining jobs of the list π are sorted according to R_2 rule, and the job selection procedure is started. Thus, Job 4 and Job 1 are considered as the first jobs. The third job must be scheduled on the first available machine (machine 3). In order to choose the job to be scheduled in position m, a decision must be taken taking into account the end of the setup operation of the last scheduled job (Job 1), the completion time of last job scheduled on the current available machine, and the completion time of the last job scheduled on the next available machine (Job 4). Indeed, to minimize the machine idle time as much as possible, it will be interesting to choose a job whose loading time is less than or equal to the completion time of the next available machine minus the maximum value of the end of setup time of the last scheduled job in the final list π' and the completion time of the last job scheduled on the current available machine. Thus, Job 2 is selected to be the third job. The fourth job to be scheduled, which has to be determined, must be scheduled on the first available machine (machine 1) and must have a setup time less than or equal to the completion time of the next available machine machine 2 minus the maximum value of the end of setup time of the last scheduled job (Job 2) and the completion time of the last job scheduled on the current available machine (Job 4). Among the remaining unscheduled jobs, Job 7 is chosen. These steps are repeated until scheduling all of the remaining jobs

Algorithm 1: HS1 Heuristic

Sort the initial list $\pi = \{\pi_1, \pi_2, \ldots, \pi_n\}$ according to the R_3 rule
$\pi' \leftarrow \emptyset$
for $k = 1 \quad to \quad m-1$ **do**
    $\pi'_k \leftarrow \pi_k$
    $\pi_k \leftarrow \emptyset$
    Execute π'_k on machine k
    $C_{\pi'_k} = T_{\pi'_k} + s_{\pi'_k} + p_{\pi'_k}$
    $T_{\pi'_{k+1}} = T_{\pi'_k} + s_{\pi'_k}$
    $E_k = C_{\pi'_k}$
    $\pi' \leftarrow \pi' \cup \{\pi'_k\}$
Sort the remaining jobs of the list π according to R_2 rule
$i \leftarrow m$
$t \leftarrow m$
$j \leftarrow \underset{h \in M \setminus \{m\}}{\text{argmin}} (E_h)$
while $i \leq n-1$ **do**
    test=0
    for $k = m \quad to \quad n-1$ **do**
        if $E_j \geq \max\left(T_{\pi'_{i-1}} + s_{\pi'_{i-1}}, E_t\right) + s_{\pi_k}$ **then**
            $test \leftarrow 1$
            $\pi'_i \leftarrow \pi_k$
            Execute π'_i on machine t
            $C_{\pi'_i} = T_{\pi'_i} + s_{\pi'_i} + p_{\pi'_i}$
            $\pi \leftarrow \pi \setminus \{\pi_k\}$
            $\pi' \leftarrow \pi' \cup \{\pi'_i\}$
            $E_t = C_{\pi'_i}$
            $t \leftarrow \underset{h \in M}{\text{argmin}}(E_h)$
            $j \leftarrow \underset{v \in M \setminus \{t\}}{\text{argmin}} (E_v)$
            $T_{\pi'_{i+1}} = \max\left(T_{\pi'_i} + s_{\pi'_i}, E_t\right)$
            $i \leftarrow i + 1$
            Break
    if *test=0* **then**
        for $k = m \quad to \quad n-1$ **do**
            if $E_j \leq \max\left(T_{\pi'_{i-1}} + s_{\pi'_{i-1}}, E_t\right) + s_{\pi_k}$ **then**
                $test \leftarrow 1$
                $\pi'_i \leftarrow \pi_k$
                Execute π'_i on machine t
                $C_{\pi'_i} = T_{\pi'_i} + s_{\pi'_i} + p_{\pi'_i}$
                $\pi \leftarrow \pi \setminus \{\pi_k\}$
                $\pi' \leftarrow \pi' \cup \{\pi'_i\}$
                $E_t = C_{\pi'_i}$
                $t \leftarrow \underset{h \in M}{\text{argmin}}(E_h)$
                $j \leftarrow \underset{v \in M \setminus \{t\}}{\text{argmin}} (E_v)$
                $T_{\pi'_{i+1}} = \max\left(T_{\pi'_i} + s_{\pi'_i}, E_t\right)$
                $i \leftarrow i + 1$
                Break
Execute π'_n on machine t
$C_{max} \leftarrow \max(E_1, \ldots, E_m)$

Table 3.1: Instance with $n = 10$ and $m = 3$.

	Job 1	Job 2	Job 3	Job 4	Job 5	Job 6	Job 7	Job 8	Job 9	Job 10
p_j	9	11	13	6	8	14	9	8	7	9
s_j	2	4	6	1	3	10	5	4	3	7

of the list π.

3.4.2 Greedy heuristic HS2

In this section, we present the second greedy heuristic HS2 that aims to minimize the server waiting time. Contrarily to the heuristic Min-loagap proposed for the case of two machines by Hasani et al. [55] where jobs were scheduled in staggered order on machines (i.e., jobs alternate between machines), in our heuristic the jobs are scheduled according to the availability of machines and the server. In addition, the job with the minimal processing time is considered as the last job to be scheduled in the final sequence.

Illustrative example

Given an instance of $n = 10$ jobs and $m = 4$ machines. The processing times p_i and setup times of the jobs s_i are given in Table 3.2. The makespan value can be obtained by applying HS2 as given above. It takes 0.002 seconds to solve this instance. An optimal sequence is shown in Figure 3.2. The objective function (makespan) is equal to 90. In this instance, equality $C_{max} = LB2$ holds.

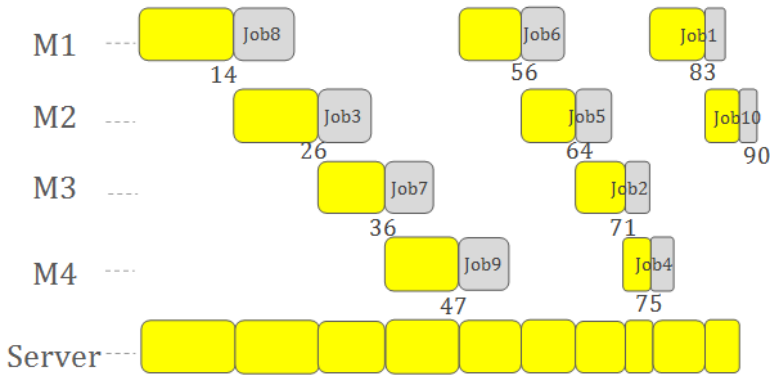

Figure 3.2: Optimal schedule for the considered instance with 10 jobs and 4 machines obtained by the heuristic HS2.

In this illustrative example, the initial list of jobs π is sorted according to R_2 rule, then

Algorithm 2: HS2 Heuristic

Data: An isntance $\pi = \{\pi_1, \ldots, \pi_k, \ldots, \pi_n\}$ of the problem $P, S1|p_j, s_j|C_{max}$
Result: π' and $C_{max}(\pi')$

Sort the initial list π according to R_1 rule
$\pi' \leftarrow \emptyset$
Find a job $\pi_r \in \pi$, $r \in \{1, \ldots, n\}$ with the smallest processing time
$\pi'_n \leftarrow \pi_r$
$\pi_t \leftarrow \pi_{t+1} \quad \forall t \in \{r, \ldots, n-1\}$
for $k = 1 \quad to \quad m-1$ **do**
    $\pi'_k \leftarrow \pi_k$
    $\pi_k \leftarrow \emptyset$
    Execute π'_k on machine k
    $C_{\pi'_k} = T_{\pi'_k} + s_{\pi'_k} + p_{\pi'_k}$
    $T_{\pi'_{k+1}} = T_{\pi'_k} + s_{\pi'_k}$
    $E_k = C_{\pi'_k}$
    $\pi' \leftarrow \pi' \cup \{\pi'_k\}$
$i \leftarrow m$
$t \leftarrow m$
$j \leftarrow \underset{h \in M \setminus \{m\}}{\text{argmin}} \ (E_h)$
while $i \leq n-1$ **do**
    test=0
    for $k = m \quad to \quad n-1$ **do**
        if $E_j \leq \max\left(T_{\pi'_{i-1}} + s_{\pi'_{i-1}}, E_t\right) + s_{\pi_k}$ **then**
            $test \leftarrow 1$
            $\pi'_i \leftarrow \pi_k$
            Execute π'_i on machine t
            $C_{\pi'_i} = T_{\pi'_i} + s_{\pi'_i} + p_{\pi'_i}$
            $\pi \leftarrow \pi \setminus \{\pi_k\}$
            $\pi' \leftarrow \pi' \cup \{\pi'_i\}$
            $E_t = C_{\pi'_i}$
            $t \leftarrow \underset{h \in M}{\text{argmin}}(E_h)$
            $j \leftarrow \underset{v \in M \setminus \{t\}}{\text{argmin}} \ (E_v)$
            $T_{\pi'_{i+1}} = \max\left(T_{\pi'_i} + s_{\pi'_i}, E_t\right)$
            $i \leftarrow i + 1$
            Break
    if *test=0* **then**
        for $k = m \quad to \quad n-1$ **do**
            if $E_j \geq \max\left(T_{\pi'_{i-1}} + s_{\pi'_{i-1}}, E_t\right) + s_{\pi_k}$ **then**
                $test \leftarrow 1$
                $\pi'_i \leftarrow \pi_k$
                Execute π'_i on machine t
                $C_{\pi'_i} = T_{\pi'_i} + s_{\pi'_i} + p_{\pi'_i}$
                $\pi \leftarrow \pi \setminus \{\pi_k\}$
                $\pi' \leftarrow \pi' \cup \{\pi'_i\}$
                $E_t = C_{\pi'_i}$
                $t \leftarrow \underset{h \in M}{\text{argmin}}(E_h)$
                $j \leftarrow \underset{v \in M \setminus \{t\}}{\text{argmin}} \ (E_v)$
                $T_{\pi'_{i+1}} = \max\left(T_{\pi'_i} + s_{\pi'_i}, E_t\right)$
                $i \leftarrow i + 1$
                Break
Execute π'_n on machine t
$C_{max}(\pi') \leftarrow \max\{E_1, \ldots, E_m\}$

Table 3.2: Instance with $n = 10$ and $m = 4$.

	Job 1	Job 2	Job 3	Job 4	Job 5	Job 6	Job 7	Job 8	Job 9	Job 10
p_j	2	4	8	3	5	6	7	9	7	2
s_j	8	7	12	4	8	9	10	14	11	5

the first $m - 1$ jobs are scheduled on the first available machines. The job with the smallest processing time is scheduled in the last position of the schedule (i.e., π_n). So, Job 8, Job 3 and Job 7 are considered as the first scheduled jobs. After this step, we start the job selection procedure. The fourth job must be scheduled on the first available machine (machine 4) and a decision must be made taking into account:

- The end of the setup time of the last scheduled job (Job 7).
- The completion time of last job scheduled on the current available machine (machine 4).
- The completion time of the last job scheduled on the next available machine (machine 1).

Indeed, to minimize the server waiting time as much as possible, it will be interesting to choose a job whose loading time is greater than or equal to the completion time of the next available machine minus the maximum value of the end of setup time of the last scheduled job of π' and the completion time of the last job scheduled on the current available machine. Thus, Job 9 is selected as the fourth job. The fifth job to be scheduled, which has to be determined, must be scheduled on the first available machine (machine 1). It setup time must be greater or equal than the completion time of the next available machine (machine 2) minus the maximum value of the end of setup time of the last scheduled job (Job 9) and the completion time of the last scheduled job on the current machine (Job 8). Among all the remaining unscheduled jobs, Job 6 is chosen. These steps are repeated until scheduling all of the remaining jobs of the list π.

3.5 Variable neighborhood search

Variable Neighborhood Search (VNS) is a metaheuristic initially proposed by Mladenović and Hansen [76]. It employs various neighborhood structures $\mathcal{N}_1, \mathcal{N}_2, \ldots$ (usually

ranked increasingly with respect to the modification they can bring to the solution structure) for exploring the search space (diversification ability) and a local search procedure for intensifying the search around promising solutions (exploitation ability). Since 1997, VNS and its variants have been widely applied to different fields [19, 84, 94, 97, 106]. At the beginning of the search, let π be the initial solution (usually generated with a constructive heuristic) and let $i = 1$ (index of the employed neighborhood structure). VNS repeats the following three main steps until a stopping criterion is not met (e.g., a time limit).

1. *Shaking*: generate a neighbor solution $\pi' \in \mathcal{N}_i(\pi)$.
2. *Local search*: try to improve π' with a local search, and let π'' be the resulting solution.
3. *Move or not*: if π'' is not better than π, set $i = i + 1$; otherwise, set $i = 1$.

More generally, anytime step (2) does not lead to an improvement of the current solution π, we use the next neighborhood structure in the next iteration, which will perform more modifications on π (as more diversification is required to move the search away from the local optimum π). In contrast, anytime π'' improves π, we intensify the search in this promising zone of the search space by coming back to smaller modifications in the shaking phase (i.e., employing again $\mathcal{N}_1(\pi)$).

3.5.1 Objective function calculation

In order to compute the cost of a given sequence of the jobs π, denoted as $C_{max}(\pi)$, the following Proposition 11 is used.

Proposition 11 *Given a sequence π of jobs, for all $k \in N$, T_{π_k} is computed as follows:*

$$T_{\pi_k} = \begin{cases} T_{\pi_0} = s_{\pi_0} = p_{\pi_0} = 0 & \\ T_{\pi_{k-1}} + s_{\pi_{k-1}} & if \quad 1 \leq k \leq m \\ \max\left(T_{\pi_{k-1}} + s_{\pi_{k-1}}, \min\limits_{1 \leq t \leq m} E_t\right) & if \quad m+1 \leq k \leq n \end{cases}$$

Proof For the first m jobs, each job in position $k \leq m$ will start immediately its setup operation after the completion of the setup operation of the job in position $k-1$ on one

of the m available machines (where $T_{\pi_0} = 0$, since all jobs are available at beginning of the schedule).

So:

$$T_{\pi_k} = T_{\pi_{k-1}} + s_{\pi_{k-1}} \quad \forall k \in \{1, \ldots, m\}$$

For the second part of the property. Let suppose that we want to schedule a job in position $k > m$, then its start time T_{π_k} will depend on the availability of the server and a machine. The server will be available to perform the setup operation of the job at position k, if :

$$T_{\pi_k} \geq T_{\pi_{k-1}} + s_{\pi_{k-1}}$$

Second, the first available machine corresponds to the one with smallest completion times of the processing operations of jobs scheduled in it (ie., $\min\limits_{1 \leq t \leq m} E_t$). Finally, in order to assure that both the server and a machine are available at same time, we choose the maximum of the two values: $T_{\pi_{k-1}} + s_{\pi_{k-1}}$ and $\min\limits_{1 \leq t \leq m} E_t$.

Thus:

$$T_{\pi_k} = \max\left(T_{\pi_{k-1}} + s_{\pi_{k-1}}, \min_{1 \leq t \leq m} E_t\right) \quad \forall k \in \{m+1, \ldots, n\}$$

3.5.2 Initial solution

Since VNS is a trajectory-based algorithm, we need to start from an initial solution. Any permutation of π defines a feasible solution for the considered problem $P, S1|p_j, s_j|C_{max}$. The jobs should be scheduled on the machine which becomes free first at the earliest time. The first m jobs are scheduled on the first m machines, and the remaining jobs of π are scheduled if both the server and a particular machine are available. The initial solution is generated by using the Longest Processing Time with Earliest Start (LPT+ES) scheduling rule [32]. This means that jobs are first sorted according the non-increasing order with respect to their processing times and scheduled in such a way that each job starts as early as possible. To calculate the earliest starting time, one has to take into account the availability of both server and the corresponding machine.

3.5.3 Neighborhood structures

To obtain an efficient VNS algorithm we have to decide about three things [76]: (i) the neighborhood structures to use, (ii) the order of these neighborhoods in the search process,

(iii) the search strategy to use in changing neighborhoods. We propose the following four neighborhood structures to explore the solution space for the problem $P, S1|p_j, s_j|C_{max}$.

- $\mathcal{N}_1(\pi) = 2\text{-}opt(\pi)$: It consists of all solutions obtained from solution π by reversing a subsequence of π. More precisely, given two jobs π_i and π_j, we construct a new sequence by first deleting the connection between π_i and its successor π_{i+1} and the connection between π_j and its successor π_{j+1}. Next, we connect π_i with π_j and π_{i+1} with π_{j+1}.
- $\mathcal{N}_2(\pi) = Insertion(\pi)$: It consists of all solutions obtained from solution π by reinserting one of its job somewhere else in the sequence.
- $\mathcal{N}_3(\pi) = Swap(\pi)$: It consists of all solutions obtained from solution π by swapping two jobs of π.
- $\mathcal{N}_4(\pi) = Transpose(\pi)$: It consists of all permutations that can be obtained by swapping two adjacent jobs of π.

These neighborhood structures have been used in literature to solve different single and parallel machines scheduling problems [17, 34, 63, 70]. After performing preliminary tests, the following neighborhood order was chosen in our proposed VNS: $\mathcal{N}_1(\pi)$, $\mathcal{N}_2(\pi)$, $\mathcal{N}_3(\pi)$ and $\mathcal{N}_4(\pi)$ ($k_{max} = 4$).

3.5.4 Shaking and Local search

The aim of a shaking procedure used within a VNS algorithm is to hopefully escape from local minima traps. The simple shaking procedure consists of selecting a random solution from the current neighborhood of the current solution $\mathcal{N}_k(\pi)$. Algorithm 3, summarizes the steps of the shaking phase.

Algorithm 3: Shaking(π,k)

Data: Solution π, neighborhood structure $\mathcal{N}_k$
Result: Solution π
1 Select randomly $\pi' \in \mathcal{N}_k(\pi)$
2 $\pi \leftarrow \pi'$
3 return π

The Local Search procedure receives an initial solution π^0 from the shaking procedure and tries continually to construct a new improved solution (improved neighbor) from the

current solution π by exploring its neighborhood $\mathcal{N}_k(\pi)$. This procedure returns the local optimum within the neighborhood of the solution. In our VNS, we propose to use the first improvement search strategy for each neighborhood structure (i.e. as soon as an improving solution π' in a neighborhood structure $\mathcal{N}_k(\pi)$ is detected it is set to be the new incumbent solution ($\pi \leftarrow \pi'$)). In Algorithm 4, we present the pseudocode of the local search procedure.

Algorithm 4: Local_Search(π^0,k)

Data: Solution π^0, neighborhood structure $\mathcal{N}_k$
Result: Solution π

```
1  π ← π^0
2  Stop ← False
3  while Stop = False do
4      Select π' ∈ N_k(π) such that C_max(π') < C_max(x)
5      if π' exists then
6          π ← π'
7      else
8          Stop ← True
9  return π
```

Algorithm 5: Variable Neighborhood Search

Data: An instance π of the problem $P, S1|p_j, s_j|C_{max}$, neighborhood structures $\mathcal{N}_k$ for $k = 1, 2, \ldots, k_{max}$, CPU_{max}: the execution time limit
Result: Solution π

```
1  Generate an initial solution π using (LPT+ES) rule;
2  k ← 1 ;
3  while CPU < CPU_max do
4      while k ≤ k_max do
5          π' ← Shaking(π, k);
6          π'' ← Local_Search(π',k);
7          if C_max(π'') < C_max(π) then
8              π ← π'';
9              k ← 1;
10         else
11             k ← k + 1;
12     if C_max(π) = LB then
13         CPU ← CPU_max;
14     else
15         k ← 1 ;
16 return π
```

3.5.5 VNS algorithm

The different steps of our proposed VNS algorithm for the problem $P, S1|p_j, s_j|C_{max}$ are summarized in Algorithm 5. It starts with an initial solution generated by the LPT+ES rule. We generate a random neighbor of this solution π with respect to the correspondent neighborhood structure $\mathcal{N}_k$. Then, we apply a first improvement local search (Algorithm 4). If no improvement exists, the neighborhood structure will be changed and the VNS algorithm stops when the execution time limit is reached or if the makespan solution is equal to the lower bound that we propose in the next section.

3.6 Computational results

In this section, the performance evaluation of the proposed algorithms HS1, HS2 and VNS is conducted by computational experiments. The proposed algorithms were coded using the C++ language and run on a PC with 2.90GHz Intel(R) Core(TM) i7-4600M CPU and 16GB of RAM memory, on Windows 7 operating system. VNS was executed 10 times in all experiments reported in this section. In addition, the stopping criteria for VNS was set to 5 seconds.

3.6.1 Benchmark instances

To the best of our knowledge, there are no publicly available benchmark instances from the literature for the problem $P, S1|p_j, s_j|C_{max}$, so we decided to generate a new set of test instances. This set was generated in the same way as described by [3, 55, 64]. The data are generated in the uncorrelated case, where the processing time values p_j are generated from a discrete uniform distribution $U(0, 100)$ and setup time values s_j are generated from a discrete uniform distribution $U(0, 100L)$, where $L = E(s_j)/E(p_j)$ is the server load and $E(x)$ denotes the mean of x. The set is composed of instances with: $n \in \{8, 20, 50, 100, 150, 200, 250, 300, 350\}$, $m = 2$, and $L \in \{0.1, 0.5, 0.8\}$ and the instances with: $n \in \{8, 20, 30, 40, 50, 60, 100, 200, 250\}$, $m \in \{3, 4, 5\}$ and $L \in \{1.5, 1.8, 2.0\}$. For each value of the server load 5 instances are generated randomly for $n \in \{8, 20\}$ and 10 instances were generated randomly for each of the other combinations of (n, m, L).

3.6.2 Comparative study

In this section, we evaluate the performance of the proposed algorithms HS1, HS2, and VNS and compare their performances with the algorithms in [3, 55]. The computational results are detailed in Tables 3.3, 3.4, and 3.5. Column 1 gives the number of jobs, column 2 gives the number of machines, columns 4 until the last column give the values R_{avg}, denoting the average value of the ratio C_{max}/LB, and R_{max}, denoting the maximum value of the ratio C_{max}/LB among all instances for a particular value of L and for a particular algorithm. From our experiments, it turns out that for all instances with $L \in \{0.1, 0.5, 0.8\}$, we always have $LB = LB1$ and for all instances with $L \in \{1.5, 1.8, 2\}$, we always have $LB = LB2$. Thus:

- For $L \in \{0.1, 0.5, 0.8\}$: we compare the obtained results of HS1 and VNS with the results in Hasani et al. [55] for the Min-idle algorithm which we denote as (MIT) and the results in Abdekhodaee and Wirth [3] for the forward heuristic which we denote as (FH).
- For $L \in \{1.5, 1.8, 2.0\}$: we compare the obtained results of HS2 and VNS with the results in Hasani et al. [55] for the Min-loadgap algorithm which we denote as (MLG) and the results in Abdekhodaee and Wirth [3] for the backward heuristic which we denote as (BH) using the same instances.

For properly evaluating the performance of our methods with the literature, we reimplemented the algorithms FH, BH, MIT and MLG. It must be noted that the maximum value of the solution time for all instances is less than 0,006 seconds for the algorithms: HS1, HS2, BH, FH, MIT, MLG. The following summary can be given :

- In Table 3.3, VNS, HS1, FH and MIT are compared for $L \in \{0.1, 0.5, 0.8\}$ and $m = 2$. It can be noticed that FH and MIT can be applied only for the case of two machines. It must be noted that FH and MIT provide the same result for all proposed instances. As shown in Figure 3.3, VNS outperformed all algorithms in terms of deviations from the lower bound for $n \in \{8, 20, 50, 100\}$. In addition, for $n \in \{150, 200, 250, 300, 350\}$, FH and MIT are better than VNS and HS1 in terms of deviations from the lower bound.
- In Table 3.4, VNS, HS2, BH and MLG are compared for $L \in \{1.5, 1.8, 2.0\}$ for small-sized instances. As a point of clarification MLG has been proposed only for the case of two machines and the symbol $(\star)$ is used to specify that no solution

Table 3.3: Comparison of VNS and HS1 with FH and MIT by [3, 55] for small, medium and large-sized instances with $L \in \{0.1, 0.5, 0.8\}$.

n	m		$L = 0.1$			$L = 0.5$			$L = 0.8$		
			VNS	HS1	FH/MIT	VNS	HS1	FH/MIT	VNS	HS1	FH/MIT
8	2	R_{avg}	1.00146	1.03591	1.03591	1.00890	1.17261	1.08607	1.04293	1.20681	1.13497
		R_{max}	1.00424	1.11385	1.11385	1.01645	1.30882	1.13388	1.07066	1.35988	1.17917
20	2	R_{avg}	1.00018	1.02269	1.02464	1.00374	1.06107	1.04708	1.02854	1.08417	1.1065
		R_{max}	1.00089	1.03768	1.04288	1.00725	1.13828	1.09338	1.04620	1.17402	1.27185
50	2	R_{avg}	1.00018	1.01033	1.01119	1.00413	1.0205	1.01431	1.02841	1.06299	1.03064
		R_{max}	1.00042	1.03451	1.0308	1.00568	1.03686	1.02459	1.05196	1.12111	1.0761
100	2	R_{avg}	1.00011	1.00611	1.00611	1.00973	1.01405	1.0091	1.03794	1.04793	1.02527
		R_{max}	1.00019	1.01296	1.01296	1.01262	1.04478	1.03046	1.04488	1.08919	1.04048
150	2	R_{avg}	1.00022	1.00459	1.00459	1.01217	1.00572	1.00429	1.03575	1.01687	1.00722
		R_{max}	1.00029	1.00745	1.00745	1.01656	1.01021	1.00822	1.04586	1.03687	1.01504
200	2	R_{avg}	1.00053	1.00484	1.00484	1.02321	1.00443	1.00259	1.04823	1.01611	1.00844
		R_{max}	1.00086	1.00825	1.00825	1.03482	1.00985	1.00518	1.06696	1.02805	1.01666
250	2	R_{avg}	1.00104	1.00251	1.00251	1.03505	1.0037	1.0023	1.07177	1.01129	1.00498
		R_{max}	1.00145	1.00614	1.00614	1.04174	1.00883	1.0061	1.08966	1.0228	1.01124
300	2	R_{avg}	1.00156	1.00182	1.00182	1.04886	1.00144	1.00098	1.10423	1.02038	1.00678
		R_{max}	1.00191	1.0047	1.0047	1.05565	1.00509	1.00262	1.11526	1.03069	1.01391
350	2	R_{avg}	1.00170	1.0009	1.0009	1.05687	1.00122	1.00098	1.11822	1.01518	1.00646
		R_{max}	1.00206	1.00248	1.00248	1.06100	1.00395	1.00167	1.13631	1.03719	1.02185

Table 3.4: Comparison of VNS and HS2 with BH by [3] and MLG by [55] for small-sized instances with $L \in \{1.5, 1.8, 2.0\}$.

n	m		$L = 1.5$				$L = 1.8$				$L = 2.0$			
			VNS	HS2	BH	MLG	VNS	HS2	BH	MLG	VNS	HS2	BH	MLG
8	2	R_{avg}	1.01157	1.04044	1.02051	1.03388	1.01273	1.03848	1.0274	1.03959	1.01126	1.03309	1.02827	1.0348
		R_{max}	1.05785	1.16529	1.08058	1.16942	1.04874	1.11372	1.11372	1.11045	1.05628	1.09957	1.09668	1.13131
	3	R_{avg}	1	1	1	*	1.00053	1.00103	1.01303	*	1.00101	1.00101	1.0206	*
		R_{max}	1	1	1	*	1.00266	1.00266	1.0625	*	1.00504	1.00504	1.05804	*
20	2	R_{avg}	1.00461	1.02533	1.02085	1.01949	1.01522	1.044	1.03147	1.05137	1.00182	1.01054	1.00295	1.00536
		R_{max}	1.01092	1.039	1.03874	1.04212	1.04343	1.09224	1.07453	1.09835	1.0091	1.03893	1.01416	1.02679
	3	R_{avg}	1.00045	1.00045	1.00076	*	1.00011	1.00011	1.00011	*	1.0001	1.0001	1.0001	*
		R_{max}	1.00226	1.00226	1.00226	*	1.00057	1.00057	1.00057	*	1.0005	1.0005	1.0005	*
	4	R_{avg}	1.00083	1.00083	1.00547	*	1	1	1	*	1	1	1	*
		R_{max}	1.00413	1.00413	1.02324	*	1	1	1	*	1	1	1	*
	5	R_{avg}	1	1	1	*	1	1	1	*	1.00011	1.00011	1.00011	*
		R_{max}	1	1	1	*	1	1	1	*	1.00054	1.00054	1.00054	*

Table 3.5: Comparison of VNS and HS2 with BH by [3] for medium and large-sized instances with $L \in \{1.5, 1.8, 2.0\}$.

n	m		$L = 1.5$			$L = 1.8$			$L = 2.0$		
			VNS	HS2	BH	VNS	HS2	BH	VNS	HS2	BH
30	3	R_{avg}	1	1	1.00264	1	1	1.00026	1	1	1
		R_{max}	1	1	1.01634	1	1	1.0026	1	1	1
	4	R_{avg}	1.00014	1.00014	1.00014	1.00018	1.00018	1.00018	1	1	1
		R_{max}	1.00141	1.00141	1.00141	1.00181	1.00181	1.00181	1	1	1
40	3	R_{avg}	1	1	1.0025	1	1	1	1.00034	1.00034	1.0007
		R_{max}	1	1	1.0184	1	1	1	1.00336	1.00336	1.00336
	4	R_{avg}	1	1	1	1	1	1	1	1	1
		R_{max}	1	1	1	1	1	1	1	1	1
	5	R_{avg}	1	1	1	1	1	1	1	1	1
		R_{max}	1	1	1	1	1	1	1	1	1
50	3	R_{avg}	1	1	1.00145	1	1	1.00007	1	1	1
		R_{max}	1	1	1.00848	1	1	1.00065	1	1	1
	4	R_{avg}	1	1	1	1	1	1	1	1	1
		R_{max}	1	1	1	1	1	1	1	1	1
60	3	R_{avg}	1.00002	1.00002	1.00027	1	1	1.00029	1	1	1
		R_{max}	1.00021	1.00021	1.00142	1	1	1.00235	1	1	1
	5	R_{avg}	1.00002	1.00002	1.00002	1	1	1	1	1	1
		R_{max}	1.00021	1.00021	1.00021	1	1	1	1	1	1
100	3	R_{avg}	1	1	1	1	1	1.0001	1	1	1
		R_{max}	1	1	1	1	1	1.00102	1	1	1
	5	R_{avg}	1	1	1	1	1	1	1	1	1
		R_{max}	1	1	1	1	1	1	1	1	1
200	3	R_{avg}	1.00002	1	1.00002	1.00001	1	1	1	1	1.000005
		R_{max}	1.00008	1	1.00014	1.00006	1	1	1.00001	1	1.00005
	4	R_{avg}	1	1	1	1	1	1	1	1	1
		R_{max}	1	1	1	1	1	1	1	1	1
250	3	R_{avg}	1.0007	1	1.00006	1.00019	1	1.00001	1.00002	1	1
		R_{max}	1.00209	1	1.00051	1.00069	1	1.00009	1.00012	1	1
	5	R_{avg}	1	1	1	1	1	1	1	1	1
		R_{max}	1	1	1	1	1	1	1	1	1

can be found with this algorithm for $m > 2$. VNS is better than all examined algorithms in term of the deviation from the lower bound for $(n = 8, m = 2, L = 1.5)$, $(n = 8, m = 2, L = 1.8)$, $(n = 8, m = 2, L = 2.0)$, $(n = 8, m = 3, L = 1.8)$, $(n = 20, m = 2, L = 1.5)$ and $(n = 20, m = 2, L = 1.8)$. It provides also the same results as HS2 for the remaining cases. The best performance in terms of deviation from the lower bound is demonstrated by VNS.

- In Table 3.5, VNS, HS2 and BH are compared for $m > 2$ and $L \in \{1.5, 1.8, 2.0\}$. It can be noticed that VNS and HS2 are able to produce an optimal solution for all instances of $(n = 30, m = 3)$, $(n = 40, m = 4)$, $(n = 40, m = 5)$, $(n = 50, m = 3)$, $(n = 50, m = 4)$, $(n = 100, m = 3)$, $(n = 100, m = 5)$, $(n = 200, m = 3)$, $(n = 200, m = 4)$, $(n = 250, m = 3)$ and $(n = 250, m = 5)$. In addition, VNS and HS2 outperformed the heuristic BH in terms of deviation from the lower bound for $(n = 30, m = 3, L = 1.5)$, $(n = 30, m = 3, L = 1.8)$, $(n = 40, m = 3, L = 2.0)$, $(n = 50, m = 3, L = 1.5)$, $(n = 50, m = 3, L = 1.8)$, $(n = 60, m = 3, L = 1.5)$, $(n = 60, m = 3, L = 1.8)$, $(n = 100, m = 3, L = 1.8)$, $(n = 200, m = 3, L = 2.0)$ and $(n = 200, m = 3, L = 1.5)$. The best performance is demonstrated by HS2 as shown in

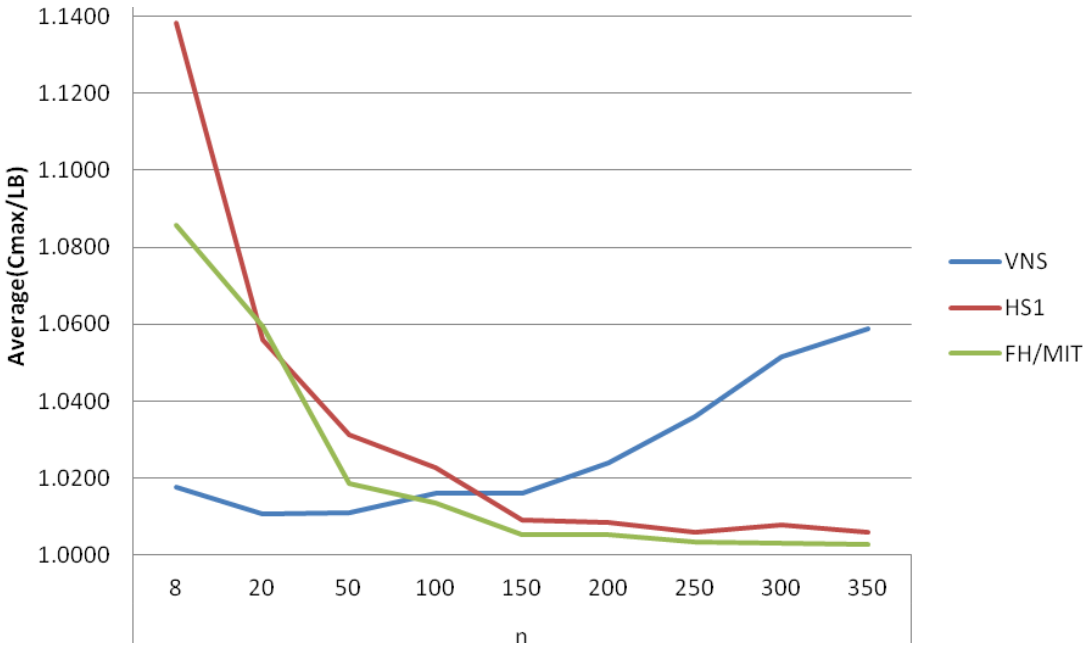

Figure 3.3: Comparison of the average value of the relation C_{max}/LB of VNS and HS1-LST with FH and MIT proposed by [3, 55] for $L \in \{0.1, 0.5, 0.8\}$.

The overall results show that the performance of HS1, HS2 and VNS is related to the number of jobs, the number of machines and the value of the server load L. For

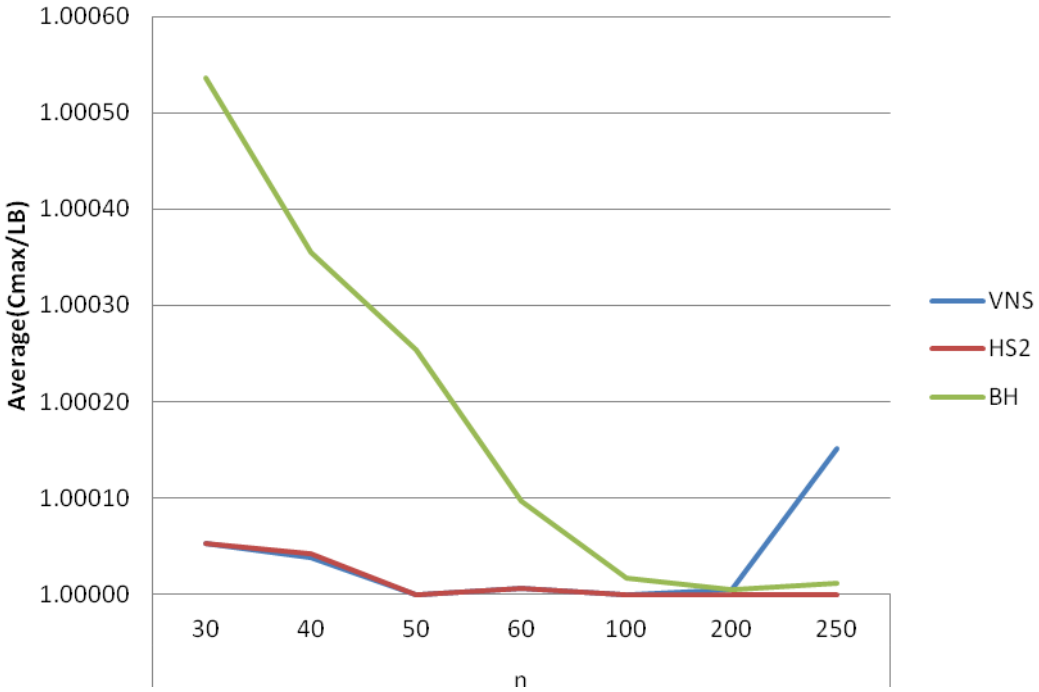

Figure 3.4: Comparison of the average value of the relation C_{max}/LB of VNS and HS2 with BH proposed by [3] for $L \in \{1.5, 1.8, 2.0\}$.

$L \in \{0.1, 0.5, 0.8\}$, VNS showed better performance in terms of deviation from the lower bound in comparison with all proposed algorithms for $n \in \{8, 20, 50, 100\}$, while, FH and MIT showed better performance in comparison with all proposed algorithms for $n \in \{150, 200, 250, 300, 350\}$. In addition, for $L \in \{1.5, 1.8, 2.0\}$, the best performance in terms of deviation from the lower bound is observed by VNS for small-sized instances and by HS2 for medium and large-sized instances. Therefore, in terms of computational time vs quality of solution, HS1 and HS2 are both more competitive than VNS, FH and MIT.

In simple words and in the context of this thesis, we recall the No Free Lunch Theorem [109, 110], which states that provided that the results are based on a sufficient number of representative instances of the target problem – that no metaheuristic is better than another one, and that the only way to improve the results is to specialize the "components" of the considered metaheuristic (i.e. encoding, operators...) to the target problem. However, on a given set of instances of a target problem, it is possible that one metaheuristic is statistically better than another one. This is not mentioned when presenting results about metaheuristics even if this applies.

3.7 Conclusions and perspectives

In this chapter we presented solution methods for solving small, medium and large-size instances of the arbitrary number of identical parallel machines scheduling problem with a single server to minimize the makespan, $P, S1|p_j, s_j|C_{max}$. Two greedy heuristics called HS1 and HS2 are proposed to minimize machine idle time and serve waiting time respectively. In addition, a VNS metaheuristic is designed for the addressed problem. The instances were generated in the same way as in previous works so that the results can be compared with existing algorithms in the literature. Based on the generated instances, it turns out that the proposed algorithms HS1, HS2 and VNS are complementary, especially VNS reached the optimal solution for the majority of the instances and showed better results in terms of deviation from de the lower bound than all proposed algorithms from the literature for $L > 1$ and $n \in \{30, 40, 50, 60, 100, 200\}$. While, For $L < 1$, VNS generated better results only for $n \in \{8, 20, 50, 100\}$. In future work, it will be interesting to explore new neighborhoods for the problem $P, S1|p_j, s_j|C_{max}$, and also to adapt the presented VNS for solving the server waiting time objective function. Another avenue of research would be to compare our proposed VNS with other metaheuristics proposed in the literature for the problem $P2, S1|p_j, s_j|C_{max}$ (see [4]), which is a particular case of the problem $P, S1|p_j, s_j|C_{max}$.

Chapter 4

Maximum lateness minimization for the parallel machine scheduling problem with a single server and job release dates

Contents

4.1 Introduction

In this chapter, we investigate a static identical parallel machine scheduling problem with a single server (SIPSS problem) by taking into account job release dates, involving the minimization of the maximum lateness, $P, S1|r_j|L_{max}$. In the scheduling literature, only a limited number of works addressed this problem. Among them, Hall et al. [44] showed that the problem $P2, S1|p_j, s_j = 1|L_{max}$ is unary $\mathcal{NP}$-hard and that the early due date rule can solve optimally the more general problem with an arbitrary number of machines with unit processing times ($P, S1|p_j = 1, s_j|L_{max}$) in $O(n \log n)$. In addition, Brucker et al. [25] showed that the problem $P2, S1|p_j, r_j = 1|L_{max}$ is unary $\mathcal{NP}$-hard. We are not aware of any recent work suggesting solution methods for the problem $P, S1|r_j|L_{max}$. A goal of this chapter aims at bridging this gap. The problem $P, S1|r_j|L_{max}$ is $\mathcal{NP}$-hard since it is a generalization of the problem $P2, S1|p_j = 1, r_j|L_{max}$. However, only small-sized instances can be solved optimally, and meta/heuristic are generally required. We therefore suggest a constructive heuristic, as well as two metaheuristics based on general variable neighborhood search (GVNS), whereas the second relies on greedy randomized adaptive search procedures (GRASP) with variable neighborhood descent (VND). Indeed, many metaheuristics are proposed for different variants of the SIPSS problem, namely: simulated annealing [14, 47, 51, 54, 62], genetic algorithm [3, 47, 55, 56], tabu search [4, 14, 52, 62], ant colony optimization [8], harmony search [51], geometric particle swarm optimization [4], iterative local search [88]. To the best of our knowledge, we could not identify any GVNS and GRASP/VND metaheuristics for scheduling problems involving a single server. The main contributions of this chapter are the following. First, we provide a MIP formulation of the considered problem, based on network variables and a lower bound for the problem. Second, we propose for the first time solution methods for the problem $P, S1|r_j|L_{max}$, namely a constructive heuristic and two metaheuristics (one based on GVNS, the other one relying on GRASP with VND). Finally, numerical results are provided for reasonable computing times, including a comparison with an exact method using CPLEX optimization solver. The remainder of this chapter is organized as follows. Problem formulation and a lower bound is presented in Section 4.2. In Section 4.3, a constructive heuristic and two metaheuristics are presented. Numerical experiments are performed in Section 4.4. Finally, concluding remarks are made in Section 4.5.

4.2 Problem formulation and lower bounds

To define the problem $P, S1|r_j|L_{max}$, let $M = \{1, \ldots, m\}$ be the set of m identical parallel machines that are available to process a set $N = \{1, \ldots, n\}$ of n independent jobs. Each job $j \in N$ is to be processed by exactly one of the machines during a given positive time p_j. Before its processing, job j must be set up on a machine by the server. The setup operation, which can be also considered as a loading or preparation operation, has a predefined integer value s_j. Each job j becomes available at its release date r_j and should be completed by its due date d_j. In addition, during the setup operation, both the machine and the server are occupied, and after setting up a job, the server becomes available for setting up the next job. The processing operation starts immediately after the end of the setup operation. Moreover, there is no precedence among jobs, and preemption is not allowed. The objective is to find a feasible schedule that minimizes the maximum lateness, $L_{max} = \max_{j \in N} L_j$, where: $L_j = C_j - d_j$ is the lateness of job j. If $L_j \leq 0$, job j is early, otherwise it is tardy. Such an objective function L_{max} is highly relevant from a practical standpoint, as its minimization contributes to the satisfaction of the clients. In contrast, minimizing C_{max} focuses on the satisfaction of the production plant. This shift from the manufacturer satisfaction to the client satisfaction can be observed in recent works [83, 95].

4.2.1 Network variables formulation

We now present a MIP formulation based on network variables for the problem $P, S1|r_j|L_{max}$. Network variables formulation (or traveling salesman problem variables formulation, or tour constraint formulation) was initially used by [80] to model the non-preemptive single machine scheduling problem with sequence-dependent processing times to minimize the makespan. This technique has been successfully used to model different $\mathcal{NP}$-hard scheduling problems [6, 10]. In this formulation, a dummy job 0 is required to be the first and the last job processed on each machine, and its release date, setup time and processing time are set to 0. Indeed, it indicates the start and the completion of the job setup and processing operations on each machine (similarly to the vehicle routing problem, where the jobs represent the customers and the machines represent the vehicles being routed [100]). The decision variables are defined as follows:

$$x_{i,j} = \begin{cases} 1 & \text{if job } i \text{ immediately precedes job } j \text{ on the same machine} \\ 0 & \text{otherwise} \end{cases}$$

$$z_{i,j} = \begin{cases} 1 & \text{if job } i \text{ finishes its processing before job } j \\ 0 & \text{otherwise} \end{cases}$$

Let B be a large positive integer, computed as $B \geq \sum_{j \in N}(r_j + s_j + p_j)$, and C_j be the completion time of job j. The problem $P, S1|r_j|L_{max}$ can be formulated as the following MIP.

$$\min \quad L_{max} \tag{4.1}$$

$$s.t. \quad L_{max} \geq C_j - d_j \qquad \forall j \in N \tag{4.2}$$

$$\sum_{j=1}^{n} x_{0,j} \leq m \tag{4.3}$$

$$\sum_{i=1}^{n} x_{i,0} \leq m \tag{4.4}$$

$$\sum_{j=0:j\neq i}^{n} x_{i,j} = 1 \qquad \forall i \in N \tag{4.5}$$

$$\sum_{i=0:i\neq j}^{n} x_{i,j} = 1 \qquad \forall j \in N \tag{4.6}$$

$$C_j \geq r_j + s_j + p_j \qquad \forall j \in N \tag{4.7}$$

$$C_i + s_j + p_j \leq C_j + B(1 - x_{i,j}) \qquad \forall (i,j) \in N^2, i \neq j \tag{4.8}$$

$$C_i + s_j + p_j \leq C_j + p_i + B(1 - z_{i,j}) \qquad \forall (i,j) \in N^2, j \neq i \tag{4.9}$$

$$z_{i,j} + z_{j,i} \geq 1 \qquad \forall (i,j) \in N^2, i \neq j \tag{4.10}$$

$$x_{i,j} \in \{0,1\} \qquad \forall i \in N \cup \{0\}, \forall j \in N \cup \{0\} \tag{4.11}$$

$$z_{i,j} \in \{0,1\} \qquad \forall (i,j) \in N^2 \tag{4.12}$$

The objective function (4.1) indicates that the maximum lateness has to be minimized. Constraints (4.2) stipulate that the maximum lateness is greater than or equal to the difference between C_i and d_i. Constraints (4.3) and (4.4) are presented in line with some vehicle-routing formulations, where the jobs are assigned to the m available machines, such that each machine starts and finishes its schedule with job 0. Constraints (4.5) and (4.6) guarantee that each job is scheduled on a particular machine. The completion time C_j of the job j is calculated according to constraints (4.7). Constraints (4.8) indicate that no two jobs i and j, scheduled on the same machine, can overlap in time. Constraints (4.9) and (4.10) state that the server can set-up at most one job at a time. Constraints (4.11) and (4.12) define variables $x_{i,j}$, $z_{i,j}$ as binaries.

4.2.2 Illustrative example

We now illustrate the previous formulation for an instance of $n = 6$ jobs and $m = 3$ machines. The processing time p_j, the setup time s_j, the release date r_j and the due date d_j of the jobs are given in Table 4.1. It takes 0.44 seconds to solve the instance using the above MIP formulation on IBM ILOG CPLEX 12.6. The optimal objective-function value is 6, and the obtained schedule of the problem is given in Figure 4.1.

Table 4.1: Instance with $n = 6$ and $m = 3$.

	Job 1	Job 2	Job 3	Job 4	Job 5	Job 6
p_j	2	2	4	3	2	2
s_j	2	3	5	6	2	2
r_j	2	12	4	4	7	3
d_j	7	18	20	19	12	9

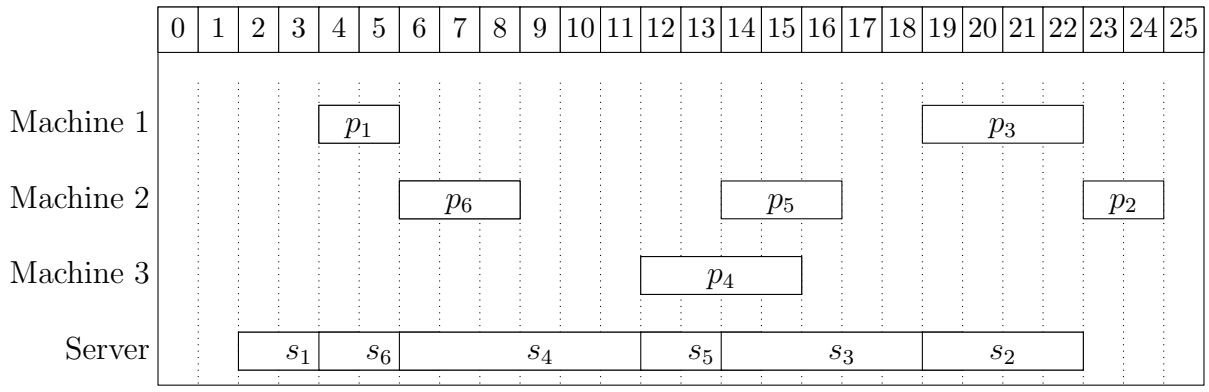

Figure 4.1: Optimal schedule for the considered instance with 6 jobs and 3 machines.

4.2.3 Lower Bound

In this subsection, we introduce a lower bound (LB) on the optimal objective-function value for the problem $P, S1|r_j|L_{max}$. This can be useful in order to evaluate the quality of the meta/heuristics presented in Section 4.4. $LB = \max(LB1, LB2)$, where $LB1$ and $LB2$ are given in Propositions 12 and 13, respectively.

Our proposed lower bound is based on the lower bounds suggested by Carlier [26] and Schutten and Leussink [87]. Indeed, Carlier [26] introduced a lower bound on the optimal makespan for the parallel machine scheduling problem in which the jobs have heads and tails with the objective of minimizing the maximum lateness. Later, Schutten and Leussink [87] extended the Carlier's lower bounds for the parallel machine schedul-

ing problem with release dates, due dates and family setup times, to minimize the maximum lateness.

Proposition 12 *$LB1$ is a valid lower bound for the problem $P, S1|r_j|L_{max}$.*

$$LB1 = \max_{j \in N} (r_j + s_j + p_j - d_j)$$

Proposition 13 *Let i_1, i_2, ..., i_m be the m jobs with the smallest release dates, and j_1, j_2, ..., j_m the m jobs with the largest due dates. $LB2$ is a valid lower bound for the problem $P, S1|r_j|L_{max}$.*

$$LB2 = \frac{(r_{i_1} + r_{i_2} + \ldots + r_{i_m}) + \sum_{i \in N} (s_i + p_i) - (d_{j_1} + d_{j_2} + \ldots + d_{j_m})}{m}$$

4.3 Solution methods

In this section, we first discuss the solution representation and the objective-function calculation of the studied problem. Next, we present a constructive heuristic $H1$. Finally, we present two metaheuristics: one based on GVNS, and the other one being a combination of GRASP and VND. Note that $H1$ will be used to generate initial solutions for GVNS.

4.3.1 Solution representation and objective-function calculation

A schedule of the problem $P, S1|r_j|L_{max}$ can be presented as a permutation $\pi = \{\pi_1, \ldots, \pi_k, \ldots, \pi_n\}$ of the job set N, where π_k indicates the job which is processed in the k^{th} position. Any permutation of all jobs defines a feasible schedule, where each job will be scheduled if it is released and if both a machine and the server are available simultaneously. Additional notation is defined in Table 4.2.

In order to compute the objective function of a given sequence π of jobs, denoted as $L_{max}(\pi)$, the following Proposition 14 is used.

Table 4.2: Employed notation.

Terms	Definition
$\pi = \{\pi_1, \ldots, \pi_k, \ldots, \pi_n\}$	Job sequence to be scheduled
E_t	Completion time of the machine $t \in M$
T_{π_k}	Start time of the setup operation of the job scheduled at position k
C_{π_k}	Completion time of the job scheduled at position k
s_{π_k}	Setup time of the job scheduled at position k
p_{π_k}	Processing time of the job scheduled at position k
r_{π_k}	Release date of the job scheduled at position k
d_{π_k}	Due date of the job scheduled at position k
L_{π_k}	Lateness of the job scheduled at position k

Proposition 14 *Given a sequence π of jobs, for all $i \in N$, T_{π_k} is computed as follows:*

$$T_{\pi_k} = \begin{cases} r_{\pi_k} & \text{if} \quad k = 1 \\ \max\left(r_{\pi_k}, T_{\pi_{k-1}} + s_{\pi_{k-1}}\right) & \text{if} \quad 2 \leq k \leq m \\ \max\left(r_{\pi_k}, T_{\pi_{k-1}} + s_{\pi_{k-1}}, \min\limits_{1 \leq t \leq m} E_t\right) & \text{if} \quad m+1 \leq k \leq n \end{cases}$$

Proof First of all, it is easy to see that the first job will start its setup operation at its release date. Thus, $T_{\pi_k} = r_{\pi_k}$ if $k = 1$.

Second, each job in position $2 \leq k \leq m$ will start immediately its setup operation if it is released and after the completion of the setup operation of the job in position $k-1$, on one of the $\{1, \ldots, m\}$ available machines. This is trivial because each time, at least one machine will be available for processing this job. Therefore: $T_{\pi_k} = \max\left(r_{\pi_k}, T_{\pi_{k-1}} + s_{\pi_{k-1}}\right)\ \forall k \in \{2, \ldots, m\}$.

Third, suppose that we want to schedule a job in position $k \geq m+1$. Its start time T_{π_k} will depend on its release date and also on the availability of the server and a machine. The job at position k can only be scheduled if it is released, thus $T_{\pi_k} \geq r_{\pi_k}$.

Moreover, the server will be available to perform the setup operation of the job at position k if $T_{\pi_k} \geq T_{\pi_{k-1}} + s_{\pi_{k-1}}$. In addition, the job at position k will be scheduled on the first available machine t, which corresponds to the one with the smallest completion time of all jobs scheduled on it. Hence, $t = \arg\min\limits_{1 \leq t' \leq m} (E_{t'})$.

Finally, in order to ensure that the job, the server and a machine are available at same time, we choose the maximum of the three values: r_{π_k}, $T_{\pi_{k-1}} + s_{\pi_{k-1}}$ and $\min\limits_{1 \leq t \leq m} E_t$.

Therefore, the objective-function value associated with the job sequence π is computed

as follows:

$$L_{max}(\pi) = \max_{1 \leq k \leq n} (C_{\pi_k} - d_{\pi_k}) = \max_{1 \leq k \leq n} (T_{\pi_k} + p_{\pi_k} + s_{\pi_k} - d_{\pi_k})$$

4.3.2 Constructive heuristic ($H1$)

The idea of the heuristic $H1$ relies on the work of McMahon and Florian [74] for the problem $1|r_j|L_{max}$, considering a single machine with job release dates and maximum lateness minimization. In each step of $H1$, we choose a job to be scheduled taking into account its release date, the availability of the machines and the availability of the server.

For the first job π_1 to be scheduled, we choose the job j of N with the smallest release date r_j. Ties are broken with the largest lateness (computed here as $s_j + p_j - d_j$). The potential remaining ties are broken randomly. The job π_1 will be scheduled in the first machine of M (where M contains m machines). For the second job π_2 to be scheduled, among all the non-scheduled jobs of N that are released before the end of the setup operation of the job π_1, we choose again the job with the largest lateness. The job π_2 is scheduled in the second machine of M. This process continues the same way for the m first jobs to be scheduled, as there is always an available machine in such a case. Next, a job in position $k > m + 1$ can start its setup operation if it is released and if both a machine and the server are simultaneously available. Thus, the chosen job to be scheduled in position $k > m + 1$ must be released before a given time $\max(T_{\pi_{k-1}} + s_{\pi_{k-1}}, E_t)$, where $t = \arg\min_{h \in M}(E_h)$. Among all jobs that are released before that time, we choose the job with the largest lateness and we schedule it in the available machine.

4.3.3 General Variable Neighborhood Search (GVNS)

The simplest VNS variant is Reduced VNS (RVNS), which is VNS without the local search step. In most of the cases, it is applied to provide good initial solutions for other VNS variants. If the shaking step is omitted, the corresponding method is known as Variable Neighborhood Descent (VND), where a local search is performed relying on multiple neighborhood structures. Furthermore, different variants of VND are proposed in the literature, such as: sequential VND, pipe VND, cyclic VND [49]. In General VNS (GVNS), VND is used as a local search [48]. Following this research line, in this section, we propose a GVNS to solve the problem $P, S1|r_j|L_{max}$. It starts with an initial solution generated by the heuristic $H1$. Next, the shaking procedure and VND are applied to try to improve the current solution. This procedure continues until all predefined neighborhoods

have been explored.

Neighborhood structures $\mathcal{N}_1, \mathcal{N}_2, \mathcal{N}_3$

Below, we propose three different neighborhood structures $\mathcal{N}_1, \mathcal{N}_2, \mathcal{N}_3$ to tackle the problem $P, S1|p_j, s_j|L_{max}$. These neighborhood structures have been widely used in the literature (e.g., for the two parallel machines scheduling problem with a single server [4, 54], for other scheduling problems on parallel machines [83, 94, 96]).

- $\mathcal{N}_1(\pi) = \text{Swap}(\pi)$. It consists of all solutions obtained from solution π by swapping two jobs of π.
- $\mathcal{N}_2(\pi) = \text{Insert}(\pi)$. It consists of all solutions obtained from solution π by reinserting one of its job somewhere else in the sequence.
- $\mathcal{N}_3(\pi) = 2\text{-}opt(\pi)$. It consists of all solutions obtained from solution π by reversing a subsequence of π. More precisely, given two jobs π_i and π_j, we construct a new sequence by first deleting the connection between π_i and its successor π_{i+1} and the connection between π_j and its successor π_{j+1}. Next, we connect π_i with π_j and π_{i+1} with π_{j+1}.

Variable Neighborhood Descent (VND)

We propose to use $\mathcal{N}_1, \mathcal{N}_2, \mathcal{N}_3$ in a VND framework. The output solution will be a local optimum with respect to all the proposed neighborhood structures. Preliminary experiments that are not reported here showed that the following sequence of the neighborhood structures is the most efficient: $\mathcal{N}_3, \mathcal{N}_1, \mathcal{N}_2$. The employed VND pseudocode is presented in Algorithm 6. The *first-improvement process* means that a neighbor solution π' of the current solution π is generated randomly in the considered neighborhood structure $\mathcal{N}_i$ as long as one of the following conditions is true: (a) π' is better than π; (b) all the neighbor

solutions have been generated.

Algorithm 6: VND

1 **Initialization:** Construct a solution π with $H1$ (or read the input solution), and set $i = 3$.

2 **While** π can be improved, **do**:

1. Generate a solution $\pi' \in \mathcal{N}_i$ with a *first-improvement process*.
2. If π' is better than π, set $\pi = \pi'$.
3. Change neighborhood structure: set $i = (i + 1) \mod 3$.

Return the local optimum π with respect to $\mathcal{N}_1, \mathcal{N}_2, \mathcal{N}_3$.

Shaking procedure and overall pseudocode

Starting from the input solution π, the employed shaking procedure consists of sequentially generating h (diversification parameter) neighbor solutions in $\mathcal{N}_3$. In other words, h iterations are performed in $\mathcal{N}_3$. In each iteration, a single neighbor solution is generated at random. After preliminary experiments that are not reported here, we have decided to fix the value for the diversification parameter h, which is $n \cdot m$. The overall pseudocode of GVNS is given in Algorithm 7. The stopping criterion is a CPU time limit T. The diversification (resp. intensification) ability of GVNS relies on the shaking phase (resp. VND).

Algorithm 7: GVNS

1 **Initialization:** construct an initial solution π with $H1$.

2 **While** the computing time limit T is not reached, **do**:

1. Apply the shaking procedure to generate a solution π' from π.
2. Apply VND on π', and let π'' be the resulting solution.
3. If π'' is better than π, set $\pi = \pi''$.

Return the local optimum π with respect to $\mathcal{N}_1, \mathcal{N}_2, \mathcal{N}_3$.

4.3.4 Greedy Randomized Adaptive Search Procedures (GRASP) with VND

GRASP [40] is a multi-start metaheuristic used to solve hard optimization problems. It has been applied for different scheduling problems [7, 9, 113]. As presented in Algorithm 8, it consists of two phases which are repeated in turn as long as a stopping condition is not met: (1) a greedy randomized construction phrase CP (presented below); (2) an improvement procedure (VND in our case). The diversification (resp. intensification) ability of GRASP relies on CP (resp. VND).

Algorithm 8: GRASP

1 **While** the computing time limit T is not reached, **do**:

1. Apply the construction phase CP to generate a solution π.
2. Apply VND on π.

Return the best encountered solution during the search.

CP relies on the following ingredients: a *Candidate List* CL; a subset RCL of RL called *Restricted Candidate List*; the incremental cost $\Delta(j)$ associated with the integration of a job $j \in CL$ into π. CP generates a solution $\pi = \{\pi_1, \ldots, \pi_n\}$ iteratively from scratch. All jobs of N are initially inserted in CL and the algorithm ends when all jobs of CL are scheduled in π. In each iteration, a new job j is randomly selected in $RCL \subseteq CL$ and then scheduled in π. Now, the key issue is to determine RCL. At the beginning of the iteration r, the job sequence π contains r jobs (i.e., $\pi = \{\pi_1, \ldots, \pi_r\}$). The incremental cost $\Delta(j)$ associated with the integration of a job $j \in CL$ into π is computed as in Equation (4.13), where α is the randomness parameter. The incremental cost is used to decide if a job j is selected or not to be part of RCL. More precisely, let Δ^{min} and Δ^{max} be the smallest and largest values of $\Delta(j)$ (among the jobs of CL), respectively. RCL contains the jobs j of CL satisfying $\Delta(j) \leq \Delta^{min} + \alpha(\Delta^{max} - \Delta^{min})$.

$$\Delta(j) = \max(T_{\pi_r} + s_{\pi_r}, \min_{1 \leq t \leq m} E_t, r_j) + s_j + p_j - d_j \tag{4.13}$$

The amount of randomness in CP is controlled by the parameter α, which is selected uniformly at random in interval [0, 1]. $\alpha = 1$ (resp. $\alpha = 0$) corresponds to the purely

random (resp. greedy) construction.

Algorithm 9: Construction Phase CP of GRASP

1 **Initialization:** Set $\pi = \emptyset$, $CL = N$, and α to a value generated randomly in interval $[0, 1]$.

2 **While** $CL \neq \emptyset$, **do**:

1. Evaluate the incremental cost $\Delta(j)$ of each job $j \in CL$.
2. Compute $\Delta^{min} = \min_{j \in CL} \Delta(j)$ and $\Delta^{max} = \max_{j \in CL} \Delta(j)$.
3. Set $RCL = \{j' \in CL \mid \Delta(j') \leq \Delta^{min} + \alpha(\Delta^{max} - \Delta^{min})\}$.
4. Select randomly a job $j \in RCL$ and schedule it on π on the 1st available machine at the earliest time.
5. Remove j from CL.

Return solution π.

4.4 Computational experiments

In this section, the performances of the MIP formulation, $H1$, GVNS, and GRASP are compared. The MIP formulation was solved using the concert Technology library of CPLEX 12.6 using default settings in C++, whereas $H1$, GVNS and GRASP were coded in C++. We use a personal computer Intel(R) Core(TM) with i7-4600M 2.90 GHz CPU and 16GB of RAM, running Windows 7. Except for the small instances for which one run is sufficient, the metaheuristics were executed 10 times in all experiments reported in this section, and average results are provided. The stopping conditions employed for all the methods are presented in Table 4.3.

Table 4.3: Results of MIP, $H1$, GVNS and GRASP for the small instances.

	Stopping condition		
Method	Small instances: $n \in \{8, 10\}$	Medium instances: $n \in \{12, 15, 20, 30\}$	Large instances: $n \in \{50, 75, 100\}$
MIP	Until an optimal solution is found	3600 seconds	3600 seconds
$H1$	One run (up to 0.0001 second)	One run (up to 0.001 second)	One run (up to 0.002 second)
GVNS / GRASP	Same computing time as the time required by MIP to find an optimal solution	60 seconds	120 seconds

4.4.1 Benchmark instances

As far as we know, there are no publicly available benchmark instances from the literature for the problem $P, S1|r_j|L_{max}$. Therefore, we have generated instances according to the existing literature as proposed in [14]. These instances are publicly available at https://sites.google.com/site/dataforpssproblem/data. The instances are characterized by the following features.

- The number of jobs $n \in \{8, 10, 12, 15, 20, 30, 50, 75, 100\}$.
- The number of machine $m \in \{2, 3, 4, 5\}$.
- The processing times p_j are uniformly distributed in the interval [10, 100].
- The setup times s_j are uniformly distributed in the interval [5, 50].
- Due dates are generated using two coefficients. The first coefficient is the due date tightness factor $\tau = 1 - \bar{d}/C_{max}$, where C_{max} is the estimated makespan and $\bar{d}$ is the average due date. The second coefficient is the due date range factor R, which is defined as $R = (d_{max} - d_{min})/C_{max}$, where d_{max} and d_{min} are the maximum and minimum due dates, respectively. R provides the measure of variability of the due dates. The higher the value of R is, the larger the range of due dates generated. Due dates d_j are generated from a uniform distribution based on τ and R: $d_j = r_j + s_j + p_j + U(\bar{d} - R\bar{d}, \bar{d})$. This guarantees that each job can be completed no later than $\bar{d}$. The τ value is taken as 0.65 and 0.8, in addition the value of R is 0.2 (see [14]). C_{max} corresponds to the lower bound of the problem $P, S1|p_j, s_j|C_{max}$, computed as $\max\left(\frac{\sum_{i \in N}(s_i + p_i)}{m}, \sum_{1 \leq i \leq n} s_i + \min_{1 \leq i \leq n} p_i\right)$.
- The release dates are generated as proposed by [13]: $r_j = U(0, \bar{d})$.

Due to the $\mathcal{NP}$-hard nature of the problem, optimal solutions were found for small-sized instances with $n \in \{8, 10\}$ and $m \in \{2, 3, 4\}$. A feasible solution was obtained for medium-sized instances with $n \in \{12, 15, 20, 30\}$ and $m \in \{2, 3, 4, 5\}$. No feasible solution was found for large-sized instances with $n \in \{50, 75, 100\}$ and $m \in \{2, 3, 4, 5\}$. Therefore, $H1$, GVNS and GRASP were designed to solve medium and large instances.

4.4.2 Comparison of MIP, $H1$, GVNS and GRASP for small-sized instances

For all the tables, it is noteworthy that an empty cell means that the same value as above is kept; all the computing times are indicated in seconds. In Table 4.4, we compare the performance of the methods for small-sized instances, where an optimal solution can be found by the MIP formulation within one hour. Each instance is characterized by the following information: an ID; a number n of jobs; a number m of machines; the due date tightness factor τ; the lower bound LB computed as in Section 4.2.3; the optimal value L_{max}^{opt} of L_{max} (found by the MIP). Next, the obtained value of L_{max} is given for $H1$ (but not the computing time, as it is always below 0.0001). Finally, the computing time to find an optimal solution is given for the MIP, GVNS and GRASP. The last line of the table indicates average results. The following observations can be made: GVNS and GRASP can reach an optimal solution for each instance in less computing time than the MIP; $H1$ is never able to find optimal solutions (its average gap to optimality is around 25%), except for instance I6; the lower bounds LB are far from L_{max}^{opt} (on average, LB is 45% below L_{max}^{opt}).

Table 4.4: Results of MIP, $H1$, GVNS and GRASP for the small instances.

Instance						$H1$	**MIP**	**GVNS**	**GRASP**
ID	n	m	τ	LB	L_{max}^{opt}	L_{max}	Time	Time	Time
I1	8	2	0.65	57	79	112	1.014	0.1176	0.0396
I2			0.8	82	101	151	0.873	0.0038	0.003
I3		3	0.65	7.66	101	128	1.519	0.002	0.0051
I4			0.8	49	95	144	0.636	0.0887	0.0013
I5		4	0.65	0	49	50	0.758	0.001	0.0008
I6			0.8	12	115	115	0.995	0.0001	0.0008
I7	10	2	0.65	111.5	125	141	74.464	0.1132	0.0187
I8			0.8	174	185	222	78.662	2.9117	3.9069
I9		3	0.65	18	77	105	9.574	0.0127	0.5092
I10			0.8	102.66	122	160	5.828	0.0784	1.373
I11		4	0.65	0	62	62	3.204	0.0001	0.0041
I12			0.8	47	112	134	21.173	0.3862	0.0146
Average				**55.07**	**101.92**	**127**	**16.56**	**0.31**	**0.49**

4.4.3 Comparison of MIP, $H1$, GVNS and GRASP for medium-sized instances

Table 4.5 presents the results for the medium instances. The instance characteristics are first indicated. For the MIP, the following information is given: the lower bound LB_{MIP}, the upper bound UB_{MIP}, the percentage gap to optimality, and the time requested to

prove optimality (if below 3600 seconds). Note that for instances I31 to I33, no information is provided as CPLEX is not able to return a feasible solution. Next, the found value of L_{max} is given for $H1$. Finally, the following results are shown for GVNS and GRASP: the best (resp. average) objective-function value over 10 runs denoted as $L^{\star}_{max}$ (resp. L^{avg}_{max}). The following observations can be made.

First, CPLEX (i.e., the MIP formulation) is able to find optimal solutions for instances I16 to I18. Moreover, the limitations of CPLEX seem to start from $n = 20$ jobs, and they are obvious from $n = 30$. Second, GVNS and GRASP outperform significantly the MIP formulation both in terms of quality (i.e., objective-function values) and speed (i.e., time requested to find competitive solutions). For example, for the instances I16 to I18, GVNS and GRASP are also able to find optimal solutions, but within 60 seconds. Third, the results of $H1$ allow to measure the benefit of GVNS, as the latter employs the former to generate an initial solution. Last but not least, GVNS and GRASP have a similar performance, and they appear to be robust, as the difference between L^{avg}_{max} and $L^{\star}_{max}$ is very small (often below one unit).

Table 4.5: Results of MIP, $H1$, GVNS and GRASP for the medium instances.

Instance					**MIP**				$H1$	**GVNS**		**GRASP**	
ID	n	m	τ	LB	LB_{MIP}	UB_{MIP}	Gap(%)	Time	L_{max}	$L^{\star}_{max}$	L^{avg}_{max}	$L^{\star}_{max}$	L^{avg}_{max}
I13	12	2	0.65	158	3.75	170	97.79	3600	223	170	170	170	170
I14			0.8	217.5	59	233	74.68	3600	294	231	231	231	231
I15		3	0.65	41.33	78	115	32.17	3600	137	115	115	115	115
I16			0.8	121.33	163.98	164	0	3223.07	217	164	164	164	164.2
I17		4	0.65	93	59	59	0	79.93	93	59	60.8	59	59
I18			0.8	58.5	102	102	0	425.73	158	102	102	102	102
I19	15	2	0.65	101	0	120	100	3600	193	120	120	120	120.5
I20			0.8	270	0.92	314	99.71	3600	359	307	308.5	307	308.5
I21		3	0.65	93	0	138	100	3600	173	137	138.3	137	138.1
I22			0.8	199.67	27.92	226	87.65	3600	278	217	218.3	217	218.7
I23		4	0.65	0	0	76	100	3600	106	76	76	76	76
I24			0.8	96	23.35	206	88.66	3600	259	206	206	206	206
I25	20	2	0.65	252.5	0	283	100	3600	335	281	283.4	281	284.7
I26			0.8	467.5	0	503	100	3600	563	488	490.3	487	490.9
I27		3	0.65	57.33	0	172	100	3600	235	172	172	172	172.1
I28			0.8	201	9.55	278	96.57	3600	350	275	278.6	277	282.1
I29		4	0.65	0	0	175	100	3600	189	175	175	175	175
I30			0.8	156.5	1	323	99.69	3600	370	323	323	323	323
I31	30	2	0.65	318	-	-	-	-	488	358	362.8	358	364.4
I32			0.8	658	-	-	-	-	792	684	690.2	685	691
I33		3	0.65	184	-	-	-	-	371	292	297.3	289	297.9
I34			0.8	448.33	0	520	100	3600	649	508	513.2	512	519
I35		4	0.65	0	0	244	100	3600	262	244	244	244	244
I36			0.8	206.75	0	495	100	3600	496	491	491	491	491
I37		5	0.65	0	0	202	100	3600	228	200	200	200	200
I38			0.8	117.4	0	460	100	3600	493	454	454	454	454
Average				**173.72**	-	-	-	-	**319.65**	**263.42**	**264.8**	**263.54**	**265.31**

4.4.4 Comparison of $H1$, GVNS and GRASP for large-sized instances

Table 4.6 presents the results for the large-sized instances. It has the same structure as Table 4.5, but the MIP formulation is not considered as it is not able to produce feasible solutions. The same observations as above are true, but it can be observed that the difference between L_{max}^{avg} and $L_{max}^{\star}$ grows with n and m, which is likely to indicate the robustness degradation of GVNS and GRASP with the increase of the instance size.

Table 4.6: Results of $H1$, GVNS and GRASP for the large instances.

Instance					$H1$	**GVNS**		**GRASP**	
ID	n	m	τ	LB	L_{max}	$L_{max}^{\star}$	L_{max}^{avg}	$L_{max}^{\star}$	L_{max}^{avg}
I39	50	2	0.65	608	780	672	686.6	665	684.3
I40			0.8	1064.5	1202	1108	1121.3	1111	1125
I41		3	0.65	386.667	668	494	512	495	513.4
I42			0.8	713.333	1077	874	889.8	880	888.9
I43		4	0.65	146.5	507	419	434.6	430	437
I44			0.8	509.25	889	801	801.8	800	801.6
I45		5	0.65	901.5	1179	948	1010	974	995.2
I46			0.8	1787.5	1967	1838	1859.2	1851	1869.3
I47	75	2	0.65	377.333	930	680	734.9	688	719.8
I48			0.8	1244.67	1617	1450	1462.4	1434	1459.4
I49		3	0.65	22.25	770	731	738.6	731	739.8
I50			0.8	699	1319	1219	1225.4	1219	1224.1
I51		4	0.65	1223	2961	1476	1553.6	1524	1572
I52			0.8	2459.5	2758	2559	2597.1	2605	2616.9
I53		5	0.65	895.667	1433	1226	1276.4	1205	1314.4
I54			0.8	1568.33	2133	1864	1902.9	1892	1921.5
I55	100	2	0.65	172.75	939	822	838.9	782	810.5
I56			0.8	1074.25	1780	1641	1670.3	1624	1648.8
I57		3	0.65	0	489	475	475	475	475
I58			0.8	234.6	805	805	805	805	805
I59		4	0.65	0	639	627	635.4	627	650.1
I60			0.8	328.6	1340	1329	1336.4	1329	1330.2
I61		5	0.65	0	938	896	909.7	891	912.3
I62			0.8	548.8	1599	1574	1579.4	1574	1577.6
Average				**706.92**	**1279.96**	**1105.33**	**1127.36**	**1108.79**	**1128.84**

4.5 Conclusions

In this chapter, we investigated the problem of scheduling a set of jobs that are released over time on an arbitrary number of identical parallel machines with a single server. The objective function involved the minimization of the maximum lateness. We proposed a mixed integer programming (MIP) formulation based on network variables to solve optimally small instances (up to 10 jobs). Due to its $\mathcal{NP}$-hard nature, a constructive heuristic ($H1$) and two metaheuristics were designed to obtain solutions for larger instances with up to 100 jobs. The first metaheuristic is a General Variable Neighborhood

Search (GVNS), whereas the second is a Greedy Randomized Adaptive Search Procedure (GRASP). Both algorithms use a Variable Neighborhood Descent (VND) for the intensification phase. In line with the good practice of the related literature, we have built 62 benchmark instances to compare the four proposed methods. The results show the superiority of GVNS and GRASP over $H1$ and the MIP formulation, in terms of quality and/or speed. This study is the first to integrate release dates and to use GVNS and GRASP for scheduling problems involving a single server. An avenue of research would be to adapt the proposed methods to other machine environments, such as dedicated and unrelated parallel machines.

Conclusion and prospects

Nowadays, in modern production management systems such as just-in-time, and cellular manufacturing, it's important to separate setup times from processing times in scheduling decisions for improving resource utilization and also benefits. Indeed, in printed circuit board assembly, 50% of production capacity can be lost due to setup operations [99]. Consequently, integrating setup times in scheduling studies, as well as reducing setup times become of great importance. In this thesis, we studied the integration of setup operations in scheduling on an arbitrary number of identical parallel machines. These setup operations are performed by a single shared server. Two problems are studied, namely: the first one is about scheduling jobs on identical parallel machine with a single server to minimize the makespan ($P, S1|p_j, s_j|C_{max}$). The second one is about scheduling jobs on identical parallel machine scheduling with a single server and release dates to minimize the maximum lateness ($P, S1|r_j|L_{max}$). The objective functions C_{max} and L_{max} are studied in order to satisfy the production plants and clients respectively.

The main contributions of this study are as follows. In Chapter 2, we considered the problem $P, S1|p_j, s_j|C_{max}$, and proposed its exact resolution. First, we provided four mathematical formulations for the general case of the problem, namely: completion time variables formulation, time-indexed variables formulation, network-variables formulation and linear ordering variables formulation, as well as sets of valid inequalities to reduce the resolution time and to improve the LP relaxation lower bound for those formulations. Second, we proposed a mathematical formulation for the regular case of the problem (i.e., $\forall i, j \quad p_j + s_j \geq p_i$) based on some mathematical properties. The computational results, obtained on benchmark instances from the literature demonstrate the efficiency of our proposed formulations in comparison with the literature. Further, the contributions on the approximate resolution of the problem $P, S1|p_j, s_j|C_{max}$ have been presented in Chapter 3. Namely, we suggested two greedy heuristics, as well as a Variable Neighborhood Search (VNS) metaheuristic. The two greedy heuristics are proposed to minimize the server waiting time and the machine idle time. The computational results, obtained

on benchmark instances from the literature demonstrate the efficiency of the proposed VNS for the problem $P, S1|p_j, s_j|C_{max}$. Chapter 4 is devoted to the contributions on the resolution of the problem $P, S1|r_j|L_{max}$. In the view that only complexity results have been proposed for this $\mathcal{NP}$-hard problem, we presented a mathematical formulation based on network variables for its exact resolution. Since the mathematical formulation is able to produce feasible solutions for only instances with up to 30 jobs, heuristic-based methods are required. Therefore, a lower bound, a constructive heuristics and two metaheuristics based on General Variable Neighborhood Search (GVNS), and Greedy Randomized Adaptive Search Procedure (GRASP) with Variable Neighborhood Descent (VND) are suggested for its approximate resolution. Exhaustive computational testing on a set of randomly generated instances in line with the literature have been performed. Our findings demonstrate the efficiency of GVNS and GRASP with an efficient investment of computational effort.

The proposed methods for solving two problems of scheduling on an arbitrary number of identical parallel machines with a single server, give us four future work directions. *i*) An avenue of research would be to model the problem $P, S1|p_j, s_j|C_{max}$ discussed in Chapter 2 and the problem $P, S1|r_j|L_{max}$ discussed in Chapter 4 using other mathematical formulations such as set covering formulation [103] and arc flow formulation [111]. To the best of our knowledge, these two formulations have never been proposed for modeling the parallel machine scheduling problem with a single server and its variants. Kramer et al. [66] suggested this two formulations for the parallel machine scheduling problem with family dependent setup times and total weighted completion time minimization. *ii*) Many variants of the parallel machine scheduling problem with a single server such as: scheduling with multiple servers and scheduling with a single server and loading/unloading operations have received less attention in the scheduling literature. In addition, despite their relevant applications in industry, there is no published paper in the literature proposing solution methods for this variants. *iii*) It will be interesting to consider the online (dynamic) version of the two studied problems. In online scheduling, it is assumed that the decision-maker has an extremely limited amount of information at his disposal, and has no information with regard to the future of the process. However, in a just in time environment, proposing new approaches for the online scheduling on parallel machines using a single server, will be of great interest. As far as we known, there is no published work considering an online scheduling on an arbitrary number of identical machines involving a single server. *iv*) The efficiency of the time-indexed formulation presented in Chapter 2 represents a good starting point for developing a new strategy for the approximate resolution of the problem $P, S1|p_j, s_j|C_{max}$, by means of machine learning techniques such

as Neural Networks [57], Support Vector Machines [86], Random Forest [23] etc. Indeed, each of these algorithms would be trained with optimal schedules "over a large set of" problem instances using the time-indexed formulation and these trained algorithms are then used to solve unseen problems. Training these algorithms with exact methods can be very helpful for solving large-sized instances and/or for developing good initial solutions for metaheuristics techniques. In optimization problems, machine learning techniques are used mainly for driving metaheuristics in three levels: Problem-level (in modeling the optimization problem to be solved), Low-level (driving search component such as the initialization of solution(s), and the search variation operators) and High-level (selection and generation of metaheuristics) [93]. Therefore, there are only few papers suggesting machine learning techniques as solution methods.

Bibliography

[1] A. H. Abdekhodaee and A. Wirth. Scheduling parallel machines with a single server: some solvable cases and heuristics. *Computers & Operations Research*, 29(3):295–315, 2002.

[2] A. H. Abdekhodaee, A. Wirth, and H. S. Gan. Equal processing and equal setup time cases of scheduling parallel machines with a single server. *Computers & Operations Research*, 31(11):1867–1889, 2004.

[3] A. H. Abdekhodaee, A. Wirth, and H.-S. Gan. Scheduling two parallel machines with a single server: the general case. *Computers & Operations Research*, 33(4):994–1009, 2006.

[4] I. Alharkan, M. Saleh, M. A. Ghaleb, H. Kaid, A. Farhan, and A. Almarfadi. Tabu search and particle swarm optimization algorithms for two identical parallel machines scheduling problem with a single server. *Journal of King Saud University-Engineering Sciences*, 2019.

[5] A. Allahverdi and H. Soroush. The significance of reducing setup times/setup costs. *European Journal of Operational Research*, 187(3):978–984, 2008.

[6] B. E. Anderson, J. D. Blocher, K. M. Bretthauer, and M. A. Venkataramanan. An efficient network-based formulation for sequence dependent setup scheduling on parallel identical machines. *Mathematical and Computer Modelling*, 57(3-4):483–493, 2013.

[7] V. A. Armentano and M. F. de Franca Filho. 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, 2007.

[8] J.-P. Arnaout. Heuristics for the two-machine scheduling problem with a single server. *International Transactions in Operational Research*, 24(6):1347–1355, 2017.

[9] S. Báez, F. Angel-Bello, A. Alvarez, and B. Melián-Batista. A hybrid metaheuristic algorithm for a parallel machine scheduling problem with dependent setup times. *Computers & Industrial Engineering*, 131:295–305, 2019.

[10] K. R. Baker and B. Keller. Solving the single-machine sequencing problem using integer programming. *Computers & Industrial Engineering*, 59(4):730–735, 2010.

[11] K. R. Baker and D. Trietsch. *Principles of sequencing and scheduling*. John Wiley & Sons, 2013.

[12] E. Balas. On the facial structure of scheduling polyhedra. In *Mathematical Programming Essays in Honor of George B. Dantzig Part I*, pages 179–218. Springer, 1985.

[13] J. Bank and F. Werner. Heuristic algorithms for unrelated parallel machine scheduling with a common due date, release dates, and linear earliness and tardiness penalties. *Mathematical and computer modelling*, 33(4-5):363–383, 2001.

[14] G. Bektur and T. Saraç. A mathematical model and heuristic algorithms for an unrelated parallel machine scheduling problem with sequence-dependent setup times, machine eligibility restrictions and a common server. *Computers & Operations Research*, 103:46–63, 2019.

[15] R. Benmansour, O. Braun, and A. Artiba. On the single-processor scheduling problem with time restrictions. In *2014 International Conference on Control, Decision and Information Technologies (CoDIT)*, pages 242–245. IEEE, 2014.

[16] R. Benmansour, O. Braun, and S. Hanafi. The single-processor scheduling problem with time restrictions: complexity and related problems. *Journal of Scheduling*, pages 1–7, 2018.

[17] R. Benmansour, O. Braun, S. Hanafi, and N. Mladenovic. Using a variable neighborhood search to solve the single processor scheduling problem with time restrictions. In *International Conference on Variable Neighborhood Search*, pages 202–215. Springer, 2018.

[18] B. Bettayeb, I. Kacem, and K. H. Adjallah. 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, 2008.

[19] M. Bierlaire, M. Thémans, and N. Zufferey. A heuristic for nonlinear global optimization. *INFORMS Journal on Computing*, 22(1):59–70, 2010.

[20] E. K. Bish. A multiple-crane-constrained scheduling problem in a container terminal. *European Journal of Operational Research*, 144(1):83–107, 2003.

[21] J. Blazewicz, M. Dror, and J. Weglarz. Mathematical programming formulations for machine scheduling: A survey. *European Journal of Operational Research*, 51(3):283–300, 1991.

[22] O. Braun, F. Chung, and R. Graham. Single-processor scheduling with time restrictions. *Journal of Scheduling*, 17(4):399–403, 2014.

[23] L. Breiman. Random forests. *Machine learning*, 45(1):5–32, 2001.

[24] P. Brucker. *Scheduling Algorithms*. Springer-Verlag, Berlin, Heidelberg, 3rd edition, 2001.

[25] P. Brucker, C. Dhaenens-Flipo, S. Knust, S. A. Kravchenko, and F. Werner. Complexity results for parallel machine problems with a single server. *Journal of Scheduling*, 5(6):429–457, 2002.

[26] J. Carlier. Scheduling jobs with release dates and tails on identical machines to minimize the makespan. *European Journal of Operational Research*, 29(3):298–306, 1987.

[27] K. Chakhlevitch and C. A. Glass. Scheduling reentrant jobs on parallel machines with a remote server. *Computers & operations research*, 36(9):2580–2589, 2009.

[28] T. Cheng, S. A. Kravchenko, and B. M. Lin. Preemptive parallel-machine scheduling with a common server to minimize makespan. *Naval Research Logistics (NRL)*, 64(5):388–398, 2017.

[29] T. Cheng, S. A. Kravchenko, and B. M. Lin. Server scheduling on parallel dedicated machines with fixed job sequences. *Naval Research Logistics (NRL)*, 66(4):321–332, 2019.

[30] S. Dauzere-Peres. An efficient formulation for minimizing the number of late jobs in single-machine scheduling. In *1997 IEEE 6th International Conference on Emerging Technologies and Factory Automation Proceedings, EFTA'97*, pages 442–445. IEEE, 1997.

[31] S. Dauzère-Pérès and M. Sevaux. Using lagrangean relaxation to minimize the weighted number of late jobs on a single machine. *Naval Research Logistics (NRL)*, 50(3):273–288, 2003.

[32] T. Davidović, P. Hansen, and N. Mladenović. Permutation-based genetic, tabu, and variable neighborhood search heuristics for multiprocessor scheduling with communication delays. *Asia-Pacific Journal of Operational Research*, 22(03):297–326, 2005.

[33] J. V. De Carvalho. Exact solution of bin-packing problems using column generation and branch-and-bound. *Annals of Operations Research*, 86:629–659, 1999.

[34] M. R. De Paula, M. G. Ravetti, G. R. Mateus, and P. M. Pardalos. Solving parallel machines scheduling problems with sequence-dependent setup times using variable neighbourhood search. *IMA Journal of Management Mathematics*, 18(2):101–115, 2007.

[35] M. E. Dyer and L. A. Wolsey. Formulating the single machine sequencing problem with release dates as a mixed integer program. *Discrete Applied Mathematics*, 26(2-3):255–270, 1990.

[36] A. El Idrissi, M. Benbrahim, R. Benmansour, and D. Duvivier. Greedy heuristics for identical parallel machine scheduling problem with single server to minimize the makespan. In *MATEC Web of Conferences*, volume 200, page 00001. EDP Sciences, 2018.

[37] A. Elidrissi, M. Benbrahim, R. Benmansour, and D. Duvivier. Variable neighborhood search for identical parallel machine scheduling problem with a single server. In *International Conference on Variable Neighborhood Search*, pages 112–125. Springer, 2019.

[38] A. Elidrissi, R. Benmansour, M. Benbrahim, and D. Duvivier. Mip formulations for identical parallel machine scheduling problem with single server. In *2018 4th International Conference on Optimization and Applications (ICOA)*, pages 1–6. IEEE, 2018.

[39] A. Elidrissi, R. Benmansour, M. Benbrahim, and D. Duvivier. Mathematical formulations for the parallel machine scheduling problem with a single server. *International Journal of Production Research*, pages 1–19, 2020.

[40] T. A. Feo and M. G. Resende. Greedy randomized adaptive search procedures. *Journal of global optimization*, 6(2):109–133, 1995.

[41] H.-S. Gan, A. Wirth, and A. Abdekhodaee. A branch-and-price algorithm for the general case of scheduling parallel machines with a single server. *Computers & Operations Research*, 39(9):2242–2247, 2012.

[42] C. A. Glass, Y. M. Shafransky, and V. A. Strusevich. Scheduling for parallel dedicated machines with a single server. *Naval Research Logistics (NRL)*, 47(4):304–328, 2000.

[43] R. L. Graham, E. L. Lawler, J. K. Lenstra, and A. R. Kan. Optimization and approximation in deterministic sequencing and scheduling: a survey. In *Annals of discrete mathematics*, volume 5, pages 287–326. Elsevier, 1979.

[44] N. G. Hall, C. N. Potts, and C. Sriskandarajah. Parallel machine scheduling with a common server. *Discrete Applied Mathematics*, 102(3):223–243, 2000.

[45] A. Hamzadayi and G. Yildiz. Event driven strategy based complete rescheduling approaches for dynamic m identical parallel machines scheduling problem with a common server. *Computers & Industrial Engineering*, 91:66–84, 2016.

[46] A. Hamzadayi and G. Yildiz. Hybrid strategy based complete rescheduling approaches for dynamic m identical parallel machines scheduling problem with a common server. *Simulation Modelling Practice and Theory*, 63:104–132, 2016.

[47] A. Hamzadayi and G. Yildiz. Modeling and solving static m identical parallel machines scheduling problem with a common server and sequence dependent setup times. *Computers & Industrial Engineering*, 106:287–298, 2017.

[48] P. Hansen, N. Mladenović, and J. A. M. Pérez. Variable neighbourhood search: methods and applications. *Annals of Operations Research*, 175(1):367–407, 2010.

[49] P. Hansen, N. Mladenović, R. Todosijević, and S. Hanafi. Variable neighborhood search: basics and variants. *EURO Journal on Computational Optimization*, 5(3):423–454, 2017.

[50] K. Hasani, S. A. Kravchenko, and F. Werner. Block models for scheduling jobs on two parallel machines with a single server. *Computers & Operations Research*, 41:94–97, 2014.

[51] K. Hasani, S. A. Kravchenko, and F. Werner. A hybridization of harmony search and simulated annealing to minimize mean flow time for the two-machine scheduling problem with a single server. *Int J Oper Res*, 3(1):9–26, 2014.

[52] K. Hasani, S. A. Kravchenko, and F. Werner. Minimising interference for scheduling two parallel machines with a single server. *International Journal of Production Research*, 52(24):7148–7158, 2014.

[53] K. Hasani, S. A. Kravchenko, and F. Werner. Minimizing total weighted completion time approximately for the parallel machine problem with a single server. *Information Processing Letters*, 114(9):500–503, 2014.

[54] K. Hasani, S. A. Kravchenko, and F. Werner. Simulated annealing and genetic algorithms for the two-machine scheduling problem with a single server. *International Journal of Production Research*, 52(13):3778–3792, 2014.

[55] K. Hasani, S. A. Kravchenko, and F. Werner. Minimizing the makespan for the two-machine scheduling problem with a single server: Two algorithms for very large instances. *Engineering Optimization*, 48(1):173–183, 2016.

[56] S. Huang, L. Cai, and X. Zhang. Parallel dedicated machine scheduling problem with sequence-dependent setups and a single server. *Computers & Industrial Engineering*, 58(1):165–174, 2010.

[57] A. K. Jain, J. Mao, and K. M. Mohiuddin. Artificial neural networks: A tutorial. *Computer*, 29(3):31–44, 1996.

[58] Y. Jiang, J. Dong, and M. Ji. Preemptive scheduling on two parallel machines with a single server. *Computers & Industrial Engineering*, 66(2):514–518, 2013.

[59] Y. Jiang, F. Yu, P. Zhou, and J. Hu. Online algorithms for scheduling two parallel machines with a single server. *International Transactions in Operational Research*, 22(5):913–927, 2015.

[60] Y. Jiang, P. Zhou, H. Wang, and J. Hu. Scheduling on two parallel machines with two dedicated servers. *ANZIAM Journal*, 58:314–323, 2016.

[61] A. B. Keha, K. Khowala, and J. W. Fowler. Mixed integer programming formulations for single machine scheduling problems. *Computers & Industrial Engineering*, 56(1):357–367, 2009.

[62] M.-Y. Kim and Y. H. Lee. Mip models and hybrid algorithm for minimizing the makespan of parallel machines scheduling problem with a single server. *Computers & Operations Research*, 39(11):2457–2468, 2012.

[63] G. Kirlik and C. Oguz. A variable neighborhood search for minimizing total weighted tardiness with sequence dependent setup times on a single machine. *Computers & Operations Research*, 39(7):1506–1520, 2012.

[64] C. P. Koulamas. Scheduling two parallel semiautomatic machines to minimize machine interference. *Computers & Operations Research*, 23(10):945–956, 1996.

[65] A. Kramer, M. Dell'Amico, and M. Iori. Enhanced arc-flow formulations to minimize weighted completion time on identical parallel machines. *European Journal of Operational Research*, 275(1):67–79, 2019.

[66] A. Kramer, M. Iori, and P. Lacomme. Mathematical formulations for scheduling jobs on identical parallel machines with family setup times and total weighted completion time minimization. *European Journal of Operational Research*, pages to appear, available online 5 July 2019, 2019.

[67] S. A. Kravchenko and F. Werner. Parallel machine scheduling problems with a single server. *Mathematical and Computer Modelling*, 26(12):1–11, 1997.

[68] S. A. Kravchenko and F. Werner. Scheduling on parallel machines with a single and multiple servers. *Otto-von-Guericke-Universitat Magdeburg*, 30(98):1–18, 1998.

[69] S. A. Kravchenko and F. Werner. A heuristic algorithm for minimizing mean flow time with unit setups. *Information Processing Letters*, 79(6):291–296, 2001.

[70] H. Krim, R. Benmansour, D. Duvivier, and A. Artiba. A variable neighborhood search algorithm for solving the single machine scheduling problem with periodic maintenance. *RAIRO-Operations Research*, 53(1):289–302, 2019.

[71] J. B. Lasserre and M. Queyranne. Generic scheduling polyhedra and a new mixed-integer formulation for single-machine scheduling. *In Proceedings of the second IPCO conference, Carnegie-Mellon University Pittsburgh*, pages 136–149, 1992.

[72] G.-S. Liu, J.-J. Li, H.-D. Yang, and G. Q. Huang. Approximate and branch-and-bound algorithms for the parallel machine scheduling problem with a single server. *Journal of the Operational Research Society*, 70(9):1554–1570, 2019.

[73] J. Lohmer and R. Lasch. Production planning and scheduling in multi-factory production networks: a systematic literature review. *International Journal of Production Research*, pages 1–27, 2020.

[74] G. McMahon and M. Florian. On scheduling with ready times and due dates to minimize maximum lateness. *Operations research*, 23(3):475–482, 1975.

[75] L. P. Michael. *Scheduling: theory, algorithms, and systems.* Springer, 2018.

[76] N. Mladenović and P. Hansen. Variable neighborhood search. *Computers & operations research*, 24(11):1097–1100, 1997.

[77] J. Ou, X. Qi, and C.-Y. Lee. Parallel machine scheduling with multiple unloading servers. *Journal of Scheduling*, 13(3):213–226, 2010.

[78] A. Pessoa, E. Uchoa, M. P. De Aragão, and R. Rodrigues. Exact algorithm over an arc-time-indexed formulation for parallel machine scheduling problems. *Mathematical Programming Computation*, 2(3-4):259–290, 2010.

[79] M. Queyranne. Structure of a simple scheduling polyhedron. *Mathematical Programming*, 58(1-3):263–285, 1993.

[80] M. Queyranne and A. S. Schulz. *Polyhedral approaches to machine scheduling.* TU, Fachbereich 3, Berlin, 1994.

[81] M. Queyranne and Y. Wang. Single-machine scheduling polyhedra with precedence constraints. *Mathematics of Operations Research*, 16(1):1–20, 1991.

[82] M. Ratli, R. Benmansour, R. Macedo, S. Hanafi, and C. Wilbaut. Mathematical programming and heuristics for scheduling problems with early and tardy penalties. *Metaheuristics for Production Scheduling*, pages 183–223, 2013.

[83] J. Respen, N. Zufferey, and P. Wieser. Three-level inventory deployment for a luxury watch company facing various perturbations. *Journal of the Operational Research Society*, 68(10):1195–1210, 2017.

[84] M. Schneider, A. Stenger, and J. Hof. An adaptive vns algorithm for vehicle routing problems with intermediate stops. *Or Spectrum*, 37(2):353–387, 2015.

[85] D. Schnitzler. Production scheduling in small and medium sized companies with additional setup operator constraints. 2016.

[86] B. Scholkopf and A. J. Smola. *Learning with kernels: support vector machines, regularization, optimization, and beyond.* Adaptive Computation and Machine Learning series, 2018.

[87] J. M. Schutten and R. Leussink. Parallel machine scheduling with release dates, due dates and family setup times. *International journal of production economics*, 46:119–125, 1996.

[88] J. M. P. Silva, E. Teixeira, and A. Subramanian. Exact and metaheuristic approaches for identical parallel machine scheduling with a common server and sequence-dependent setup times. *Journal of the Operational Research Society*, pages 1–15, 2019.

[89] K. Šorić. A cutting plane algorithm for a single machine scheduling problem. *European Journal of Operational Research*, 127(2):383–393, 2000.

[90] F. Sourd. New exact algorithms for one-machine earliness-tardiness scheduling. *INFORMS Journal on Computing*, 21(1):167–175, 2009.

[91] J. P. Sousa and L. A. Wolsey. A time indexed formulation of non-preemptive single machine scheduling problems. *Mathematical programming*, 54(1-3):353–367, 1992.

[92] C. Su. Online lpt algorithms for parallel machines scheduling with a single server. *Journal of Combinatorial Optimization*, 26(3):480–488, 2013.

[93] E.-G. Talbi. Machine learning into metaheuristics: A survey and taxonomy of data-driven metaheuristics. 2020.

[94] S. Thevenin and N. Zufferey. Learning variable neighborhood search for a scheduling problem with time windows and rejections. *Discrete Applied Mathematics*, 261:344–353, 2019.

[95] S. Thevenin, N. Zufferey, and R. Glardon. Model and metaheuristics for a scheduling problem integrating procurement, sale and distribution decisions. *Annals of Operations Research*, 259(1-2):437–460, 2017.

[96] S. Thevenin, N. Zufferey, and M. Widmer. Order acceptance and scheduling with earliness and tardiness penalties. *Journal of Heuristics*, 22(6):849–890, 2016.

[97] R. Todosijević, R. Benmansour, S. Hanafi, N. Mladenović, and A. Artiba. Nested general variable neighborhood search for the periodic maintenance problem. *European Journal of Operational Research*, 252(2):385–396, 2016.

[98] L. Torjai and F. Kruzslicz. Mixed integer programming formulations for the biomass truck scheduling problem. *Central European Journal of Operations Research*, 24(3):731–745, 2016.

[99] S. C. Trovinger and R. E. Bohn. Setup time reduction for electronics assembly: Combining simple (smed) and it-based methods. *Production and operations management*, 14(2):205–217, 2005.

[100] Y. Unlu and S. J. Mason. Evaluation of mixed integer programming formulations for non-preemptive parallel machine scheduling problems. *Computers & Industrial Engineering*, 58(4):785–800, 2010.

[101] J. Van den Akker, C. A. Hurkens, and M. W. Savelsbergh. Time-indexed formulations for machine scheduling problems: Column generation. *INFORMS Journal on Computing*, 12(2):111–124, 2000.

[102] J. Van den Akker, C. Van Hoesel, and M. W. P. Savelsbergh. A polyhedral approach to single-machine scheduling problems. *Mathematical Programming*, 85(3):541–572, 1999.

[103] J. M. van Den Akker, J. A. Hoogeveen, and S. L. van de Velde. Parallel machine scheduling by column generation. *Operations Research*, 47(6):862–872, 1999.

[104] G. Wang and T. E. Cheng. An approximation algorithm for parallel machine scheduling with a common server. *Journal of the Operational Research Society*, 52(2):234–237, 2001.

[105] H. Wang, H. Li, Y. Zhao, D. Lin, and J. Li. Genetic algorithm for scheduling reentrant jobs on parallel machines with a remote server. *Transactions of Tianjin University*, 19(6):463–469, 2013.

[106] S. Wang, R. Wu, F. Chu, and J. Yu. Variable neighborhood search-based methods for integrated hybrid flow shop scheduling with distribution. *Soft Computing*, pages 1–20, 2019.

[107] F. Werner and S. A. Kravchenko. Scheduling with multiple servers. *Automation and Remote Control*, 71(10):2109–2121, 2010.

[108] C. Wilbaut, R. Benmansour, S. Hanafi, and O. Braun. Iterative relaxation-based heuristic for the single-processor scheduling problem with time restrictions. In *2015 International Conference on Industrial Engineering and Systems Management (IESM)*, pages 496–501. IEEE, 2015.

[109] D. Wolpert. The lack of a priori distinctions between learning algorithms. *Neural Computation*, 8:1341–1390, 1996.

[110] D. H. Wolpert and W. G. Macready. No free lunch theorems for optimization. *IEEE Transactions on Evolutionary Computation*, 1(1):67–82, 1997.

[111] L. A. Wolsey. Valid inequalities, covering problems and discrete dynamic programs. In *Annals of Discrete Mathematics*, volume 1, pages 527–538. Elsevier, 1977.

[112] X. Xie, Y. Li, H. Zhou, and Y. Zheng. Scheduling parallel machines with a single server. In *Proceedings of 2012 International Conference on Measurement, Information and Control*, volume 1, pages 453–456. IEEE, 2012.

[113] J. C. Yepes-Borrero, F. Villa, F. Perea, and J. P. Caballero-Villalobos. Grasp algorithm for the unrelated parallel machine scheduling problem with setup times and additional resources. *Expert Systems with Applications*, 141:112959, 2020.

[114] A. Zhang, H. Wang, Y. Chen, and G. Chen. Scheduling jobs with equal processing times and a single server on parallel identical machines. *Discrete Applied Mathematics*, 213:196–206, 2016.

[115] L. Zhang and A. Wirth. On-line scheduling of two parallel machines with a single server. *Computers & operations research*, 36(5):1529–1553, 2009.