4OR (2024) 22:351–385
https://doi.org/10.1007/s10288-023-00547-3

RESEARCH PAPER

Minimization of maximum lateness on parallel machines with a single server and job release dates

Abdelhak Elidrissi[1] · Rachid Benmansour[2] · Nicolas Zufferey[3,4] · Mohammed Benbrahim[5] · David Duvivier[6]

Received: 3 March 2023 / Revised: 3 March 2023 / Accepted: 30 June 2023 /
Published online: 19 July 2023

Abstract

This paper addresses the problem of scheduling independent jobs with release dates on identical parallel machines with a single server. The goal consists in minimizing the maximum lateness. This is a realistic extension of the traditional parallel machine scheduling problem with a single server, in which all jobs are assumed to be available at the beginning of the schedule. This problem, referred to as $P, S1|r_j|L_{max}$, has various applications in practice. To date, research on it has focused on complexity analysis. To solve small-sized instances of the problem, we present two mixed-integer-programming formulations, along with a valid inequality. Due to the $\mathcal{NP}$-hard nature of the problem, we propose a constructive heuristic and two metaheuristics, namely a General Variable Neighborhood Search (GVNS) and a Greedy Randomized Adaptive Search Procedures, both using a Variable Neighborhood Descent as an intensification operator. In the experiments, the proposed algorithms are compared using a set of new instances generated randomly with up to 500 jobs, in line with the related literature. It turns out that GVNS outperforms by far the other approaches.

Keywords Heuristics · Metaheuristics · Production · Scheduling with a single server

✉ Abdelhak Elidrissi
abdelhak.elidrissi@uir.ac.ma

1 Rabat Business School, International University of Rabat, Parc Technopolis, Rabat-Shore, Morocco

2 National Institute of Statistics and Applied Economics (INSEA), Rabat, Morocco

3 Geneva School of Economics and Management, GSEM – University of Geneva, Uni-Mail, 1211 Geneva 4, Switzerland

4 Interuniversity Research Centre on Enterprise Networks, Logistics and Transportation (CIRRELT), Montreal, Canada

5 MOAD6 Team, MASI Laboratory, Mohammadia School of Engineers, Mohammed V University, Rabat, Morocco

6 LAMIH, UMR CNRS 8201, Univ. Polytechnique Hauts-de-France, 59313 Valenciennes, France

Mathematics Subject Classification 90B35

1 Introduction

Parallel machine scheduling problems have been widely studied in the literature (Mokotoff 2001). Among all these problems, the parallel machine scheduling with a single server (PSS) has received much attention over the last 2 decades. In the PSS problem, it is considered that the server is in charge of the setup operation of jobs. This setup operation can be also considered as a loading and/or unloading operation of a job on a particular machine (Bektur and Saraç 2019; Hamzadayi and Yildiz 2017; Kim and Lee 2012). Indeed, in the classical parallel machine scheduling problem, it is assumed that the jobs are ready to be executed without prior setup. However, this assumption is not always satisfied in practice where industrial systems are more flexible. For more information about scheduling with setup times, readers can refer to Allahverdi and Soroush (2008).

The PSS problem has many industrial applications. In *container terminals*, Bish (2003) 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 represent the server, each container corresponds to a job, and vehicles corresponds to machines. The objective is to minimize the maximum turnaround time of a ship, which can represent the makespan. In the *printing industry*, Huang et al. (2010) considered a set of printing machines that 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 and 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 2012). In *supply chain optimization*, Torjai and Kruzslicz (2016) studied a biomass truck scheduling problem that originated from a real-life herbaceous biomass supply chain. The authors considered it as an identical PSS problem, for which two objective functions have to be minimized: the number of machines and the idle times. The authors stated that the identical trucks represent the identical parallel machines in charge of delivering biomass from satellite storage locations to a central bio-refinery 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 the *plastic injection industry*, Bektur and Saraç (2019) considered a set of plastic injection machines, which involve a team of workers who 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 *healthcare*, Hsu et al. (2020) studied a scheduling problem of anaesthesia operations in operating rooms. The authors considered operating rooms as machines, and operations as jobs. Each operation consists of two parts, anaesthesia operation and surgical operation. They assumed that only a single anaesthetist is available for carrying out all anaesthesia operations across the available rooms. The objective is to minimize the makespan.

In this paper, we investigate an identical PSS (IPSS) problem by taking into account job release dates. The objective is to minimize the maximum lateness. Following the standard $\alpha|\beta|\gamma$ classification scheme for scheduling problems known as the Graham triplet (Graham et al. 1979), our problem can be denoted as $P, S1|r_j|L_{max}$, where P represents identical parallel machines, $S1$ represents the single server, r_j is the release date of job j, and L_{max} is the maximum lateness.

In the scheduling literature, only a limited number of works addressed the problem $P, S1|r_j|L_{max}$. Among them, Hall et al. (2000) showed that the problem $P2, S1|s_j = 1|L_{max}$ is unary $\mathcal{NP}$-hard and that the *earliest-due-date* rule can solve optimally the more general problem with an arbitrary number of machines with unit processing times ($P, S1|p_j = 1|L_{max}$) in $O(n \log n)$. Brucker et al. (2002) showed that the problem $P2, S1|r_j = 1|L_{max}$ is unary $\mathcal{NP}$-hard. However, regarding the problem $P|r_j|L_{max}$, without considering the single server, the studies have considered sequence-dependent setup times. We are not aware of any recent work suggesting solution methods for the problem $P, S1|r_j|L_{max}$. A goal of our paper aims at bridging this gap, and to generalize the problem $P|r_j|L_{max}$.

The problem $P, S1|r_j|L_{max}$ is $\mathcal{NP}$-hard since it is a generalization of the problem $P2, S1|p_j, r_j = 1|L_{max}$. However, only small-sized instances can be solved optimally, and meta/heuristics are generally required. We therefore suggest: (1) a constructive heuristic; (2) a Variable Neighborhood Descent (VND); (3) a metaheuristic based on General Variable Neighborhood Search (GVNS); (4) a metaheuristic relying on Greedy Randomized Adaptive Search Procedures (GRASP). Both GVNS and GRASP employ VND as an intensification operator. Such choices are in line with the fact that many metaheuristics have been proposed for different variants of the IPSS problem, namely: simulated annealing (Kim and Lee 2012; Hasani et al. 2014b; Hamzadayi and Yildiz 2017; Bektur and Saraç 2019), genetic algorithm (Abdekhodaee et al. 2006; Huang et al. 2010; Hamzadayi and Yildiz 2017), tabu search (Kim and Lee 2012; Alharkan et al. 2020; Bektur and Saraç 2019), ant colony optimization (Arnaout 2017), geometric particle swarm optimization (Alharkan et al. 2020), iterative local search (Silva et al. 2019), and worm optimization algorithm (Arnaout 2021). To the best of our knowledge, VND, GVNS and GRASP have not been adapted to scheduling problems involving a single server. This might appear as surprising, given the success of that kind of methods for industrial scheduling problems (Respen et al. 2016; Thevenin and Zufferey 2019). In particular, our GVNS and GRASP approaches are based on various neighborhood structures and inherent diversification mechanisms (namely, a construction phase for GRASP and a shaking phase for GVNS). Such features obviously favor the escape from local optima, and thus the search space is likely to be efficiently explored. Indeed, in line with the findings in Thevenin et al. (2017), a strong exploration ability appears to be appropriate for scheduling problems involving several parallel machines.

The main contributions of this paper are the following. First, we provide two mixed-integer-programming (MIP) formulations, along with a valid inequality for the problem $P, S1|r_j|L_{max}$. Second, we propose for the first time dedicated solution methods, namely a constructive heuristic, a VND algorithm, and two metaheuristics

(GVNS and GRASP). Finally, numerical results are provided for reasonable computing times (with respect to the literature and to the industrial practice), including a comparison with an exact method using a well-known commercial solver.

The remainder of this paper is organized as follows. Section 2 surveys the related literature. In Sect. 3, after introducing the problem, we present two Mathematical formulations along with a valid inequality. In Sect. 4, a streamline heuristic and two metaheuristics are proposed. Numerical experiments are performed in Sect. 5. Finally, concluding remarks are made in Sect. 6.

2 Literature review

Despite its importance in practice, the problem $P, S1|r_j|L_{max}$ has not received much attention in the recent publications. However, the literature on closely related problems gives us some insights on how to deal with the problem $P, S1|r_j|L_{max}$. We will see that the following ingredients have been widely employed: MIP formulations, greedy heuristics and metaheuristics, which motivates us to propose the same ingredients (see Sects. 3, 4) to tackle the problem $P, S1|r_j|L_{max}$.

In this section, we present a comprehensive literature review considering the PSS (PSS) problem, with different machine environments and objective functions. To date, most of the related works considering the PSS problem assume that all jobs are available at the beginning of the schedule (i.e., no release date is considered) and are limited to two machines. Table 1 summarizes the useful notation. The literature is characterized by three research streams, namely: complexity results (Sect. 2.1), the case of two machines (Sect. 2.2), and the case of an arbitrary number of machines (Sect. 2.3). Next, we discuss the literature related to the problem $P|r_j|L_{max}$ without a single server (Sect. 2.4). Finally, a classification of the papers related to the IPSS (Sect. 2.5).

Table 1 Notation used throughout the paper

Notation	Description	Notation	Description
P	Identical parallel machines	PD	Dedicated parallel machines
R	Unrelated parallel machines	$S1$	Scheduling with a single server
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
IT	Total machine idle time	C_{max}	Makespan
L_{max}	Maximum lateness	$\sum C_j$	Total completion time
$\sum w_jC_j$	Total weighted completion time	$\sum T_j$	Total tardiness
$\sum w_jT_j$	Total weighted tardiness	$\sum U_j$	Number of tardy jobs
$\sum w_jU_j$	Weighted number of tardy jobs		

2.1 Complexity results

Kravchenko and Werner (1997) showed that the problems $P, S1|p_j, s_j = 1|C_{max}$ and $P2, S1|p_j, s_j = s|IT$ are unary $\mathcal{NP}$-hard. Kravchenko and Werner (1998) proposed polynomial-time algorithms and some complexity results for the cases of one server, as well as k servers with $k < m$. Later, Hall et al. (2000) extended the previous studies and presented new complexity results and heuristics 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 $\sum U_j$. For the case of parallel dedicated machines, Glass et al. (2000) addressed the problem $PD, S1||C_{max}$. They showed that it is $\mathcal{NP}$-hard in the strong sense, even in the case where all setup times and all processing times are equal.

2.2 Case with two machines

Three objective functions (to be minimized) have been mainly considered, namely: makespan (C_{max}), total machine idle time (IT) and total completion time ($\sum C_j$). Koulamas (1996) showed that the problem $P2, S1|p_j, s_j|IT$ is $\mathcal{NP}$-hard in the strong sense, and proposed a beam search algorithm to solve it. Jiang et al. (2013) considered the problem $P2, S1|prmp|C_{max}$. They proposed an algorithm with a 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 (2002) addressed the problem $P2, S1|p_j, s_j|C_{max}$. They proposed a MIP formulation for the regular case and two greedy heuristics for the general case. Abdekhodaee et al. (2004) 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 heuristics. Abdekhodaee et al. (2006) developed 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, Gan et al. (2012) addressed the problem $P2, S1|p_j, s_j|C_{max}$. They presented two MIP formulations and two branch-and-price algorithms. Hasani et al. (2014a) proposed a MIP formulation 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 results showed that the proposed formulation outperformed all heuristics of Gan et al. (2012). Hasani et al. (2014b) addressed the problem $P2, S1|p_j, s_j|C_{max}$. They proposed two metaheuristics based on simulated annealing (SA) and genetic algorithm (GA). The results obtained are much better than all the previous algorithms proposed in Abdekhodaee et al. (2006) and Gan et al. (2012). Hasani et al. (2016) investigated the problem $P2, S1|p_j, s_j|C_{max}$. They proposed two greedy heuristics to solve very large-sized instances with up to 10,000 jobs. The results obtained are much better than the ones of all previous algorithms presented in the literature for very large-sized instances. Arnaout (2017) suggested a brand and 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 ones proposed by Hasani et al. (2014b) for large-sized instances. Benmansour et al. (2018) 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 Braun et al. (2014) and Benmansour et al. (2014). Recently, Alharkan et al. (2020) proposed two metaheuristics based on tabu search (TS) and geometric particle swarm optimization (GPSO) algorithms for the problem $P2, S1|p_j, s_j|C_{max}$. The results showed that the proposed metaheuristics outperformed the algorithms of Hasani et al. (2014b) for large-sized instances. Recently, Arnaout (2021) proposed a worm optimization algorithm for the problem $P2, S1|p_j, s_j|C_{max}$ involving two identical parallel machines.

2.3 Case with an arbitrary number of machines

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

Identical parallel machines Wang and Cheng (2001) proposed an approximation algorithm for the problem $P, S1|p_j, s_j|\sum w_j C_j$. Kim and Lee (2012) addressed the problem $P, S1|p_j, s_j|C_{max}$, and they suggested two MIP formulations and a hybrid heuristic algorithm. Zhang et al. (2016) proved that the SPT (shortest processing time) rule has a worst-case ratio of $1 + \frac{\sqrt{m-1}}{\sqrt{m}\sqrt{m-1}}$ for the problem $P, S1|p_j = p, s_j|\sum C_j$ (m being the number of machines). Hamzadayi and Yildiz (2017) considered the problem $P, S1|ST_{sd}|C_{max}$ with sequence-dependent setup time. They proposed a MIP formulation and two metaheuristics based on SA and GA. Cheng et al. (2017) considered the problems $P2, S1|prmp|C_{max}$ and $P, S1|prmp|C_{max}$, by taking into account job preemptions. They showed that the problems are $\mathcal{NP}$-hard and presented pseudo-polynomial-time algorithms to solve them. Liu et al. (2019) presented a branch-and-bound algorithm, a lower bound, and dominance properties for the exact resolution of the problem $P, S1|p_j, s_j|\sum w_j C_j$. Silva et al. (2019) proposed a MIP formulation based on arc-time-indexed variables and an iterative local-search metaheuristic for the problem $P, S1|ST_{sd}|C_{max}$. The results showed that the MIP formulation and the iterative local search outperformed the methods presented by Hamzadayi and Yildiz (2017). Elidrissi et al. (2021) proposed several 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 formulations suggested by Kim and Lee (2012). For the case of multiples servers: Xu et al. (2021) addressed a parallel machine scheduling problem involving multiple servers. The authors assumed that the servers are in charge of the setup and removal operations of the jobs. In addition, they considered slack times (between the setup operation and the removal operation), machine-server eligibility restrictions, and machine-server availability periods. To solve the problem, the authors proposed a MIP formulation along with some heuristics based on dispatching rules. Lee and Kim (2021) addressed a new variant of the IPSS problem, namely a parallel machine scheduling problem with job splitting, sequence-dependent setup times, and limited setup servers. The objective function involves the minimization of the makespan. To solve this new problem, the authors proposed a MIP formulation and a heuristic. Recently, Heinz et al. (2022) studied a parallel machine scheduling problem with sequence-dependent setup times, multiple servers, and makespan objective. To solve this new problem, the authors suggested a constraint programming model and constructive heuristics.

Dedicated parallel machines Huang et al. (2010) proposed a MIP formulation and a hybrid genetic algorithm for the problem $PD, S1|ST_{sd}|C_{max}$ with sequence-dependent setup times and dedicated parallel machines. Cheng et al. (2019) were the first to investigate the problem $PD, S1 \mid fixed - seq \mid C_{max}$ 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 duration or all the loading operations require the same time; two heuristics were proposed for the latter case with unit setup time for all the jobs. Recently, Cheng et al. (2021) addressed the parallel machine scheduling problem with a single server and fixed job sequence constraint in order to minimize the makespan. They showed that the problem becomes binary $\mathcal{NP}$-hard for $m = 3$ machines, and suggested a pseudo-polynomial algorithm to solve the problem with fixed m.

Unrelated parallel machines Bektur and Saraç (2019) were the first to address the problem $R, S1|M_j, ST_{sd}|\sum w_j T_j$. They presented a MIP formulation and two metaheuristics based on TS and SA by taking into consideration machine eligibility restrictions and sequence-dependent setup times.

Uniform parallel machines Kim and Lee (2021) addressed a new variant of the IPSS problem, namely a uniform parallel machine scheduling problem with machine eligibility, job splitting, sequence-dependent setup times, and limited setup servers. The objective function involves the minimization of the makespan. This new problem is motivated by a real application of piston manufacturing. To solve the problem, the authors proposed a MIP formulation and an efficient heuristic algorithm.

2.4 Studies on the problem $P|r_j|L_{max}$ without a single server

In this section, we present the papers related to the parallel machine scheduling with release dates to minimize the maximum lateness without a single server. Ovacik and Uzsoy (1995) addressed the problem $P|ST_{sd}, r_j|L_{max}$ with sequence-dependent setup times. They proposed a family of rolling horizon procedures. The results showed that proposed methods outperformed dispatching rules combined with local-search techniques. Schutten and Leussink (1996) proposed lower bounds and a branch-and-bound algorithm for the parallel machine scheduling problem with release dates and family setup times to minimize the maximum lateness ($P|r_j, s_j|L_{max}$). Centeno and Armacost (1997) suggested an algorithm for the problem $P|r_j, M_j|L_{max}$ for the special case where due dates are equal to release dates plus a constant. Haouari and Gharbi (2003) proposed lower bounds for the problem $P|r_j|L_{max}$. Kim and Shin (2003) presented a restricted TS for the problem $P|ST_{sd}, r_j|L_{max}$. Lee et al. (2010) developed a restricted SA for the problem $P|ST_{sd}, r_j|L_{max}$. Ying and Cheng (2010) suggested an iterated greedy heuristic for the problem $P|ST_{sd}, r_j|L_{max}$. Later, Lin et al. (2011) proposed an improved iterated greedy heuristic with a sinking temperature for the same problem. The metaheurictics of Lee et al. (2010) and Ying and Cheng (2010) have not been compared. It can be noticed that no solution algorithm has been designed for the problem $P|r_j|L_{max}$.

2.5 Studies on the IPSS problem

The papers considering the IPSS problem are summarized in Table 2, which also highlight our contributions (see the last line). This table reveals that only two papers addressed the problem $P, S1|r_j|L_{max}$. In the first one, Hall et al. (2000) showed that the problem $P, S1|p_j = 1, s_j|L_{max}$ (without release dates) can be solved in $O(n \log n)$, and that the problem $P2, S1|p_j, s_j = 1|L_{max}$ (with unit setup time) is unary $\mathcal{NP}$-hard. In the second one, Brucker et al. (2002) showed that the problem $P2, S1|p_j = 1, r_j|L_{max}$ with unit processing time is unary $\mathcal{NP}$-hard. Therefore, no solution algorithm has been designed for the general problem $P, S1|r_j|L_{max}$. In addition, no previous work on scheduling problems involving a single server proposed solution methods for the case where jobs are released over time. Note that for the problem $P|r_j|L_{max}$ without considering the single server, no solution algorithm has been suggested.

3 Problem formulation

To define the problem $P, S1|r_j|L_{max}$ addressed in this paper, let $M = \{1, 2, \dots, m\}$ be the set of m identical parallel machines that are available to process a set $N = \{1, 2, \dots, n\}$ 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 duration 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 $L_j < 0$, job j is early, and if $L_j > 0$, 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 (Respen et al. 2017; Thevenin et al. 2017).

3.1 Network variables formulation (MIP_1)

We present here a MIP formulation based on network variables for the problem $P, S1|r_j|L_{max}$. Network variables formulation, known also as traveling-salesman-problem variables formulation or tour-constraint formulation, was initially proposed

Table 2 An overview of the IPSS problem works with a single server

Publications	Jobs			Setup type		Number of machines		Objective	Approach (comments)
	r_j	d_j	$prmp$	Dependent	Independent	$m = 2$	$m \geq 2$		
Koulamas (1996)					✓	✓		IT	Complexity results and a beam search heuristic
Kravchenko and Werner (1997)					✓	✓	✓	C_{max}, IT	Complexity results, polynomially solvable cases and heuristics
Kravchenko and Werner (1998)					✓	✓	✓	C_{max}, IT	Complexity results for the case of one server and $m-1$ servers
Hall et al. (2000)					✓	✓	✓	C_{max}, L_{max}, etc	Complexity results, polynomial time algorithms for: $C_{max}, L_{max}, \sum w_jC_j, \sum C_j, \sum w_jT_j, \sum T_j, \sum w_jU_j, \sum U_j$
Wang and Cheng (2001)					✓		✓	$\sum w_jC_j$	An approximation algorithm
Brucker et al. (2002)	✓	✓			✓	✓	✓	C_{max}, L_{max}, etc	New complexity results for: $C_{max}, L_{max}, \sum w_jC_j, \sum C_j, \sum w_jT_j, \sum T_j, \sum w_jU_j, \sum U_j$
Abdekhodaee and Wirth (2002)					✓	✓		C_{max}	Complexity results, MIP formulation for the regular case
Abdekhodaee et al. (2004)					✓	✓		C_{max}	Complexity results, lower bound and heuristics (equal processing times)
Abdekhodaee et al. (2006)					✓	✓		C_{max}	Greedy heuristics and GA (general case)
Kim and Lee (2012)					✓		✓	C_{max}	MIP formulations and hybrid SA and TS
Gan et al. (2012)					✓	✓		C_{max}	MIP formulations and Branch and Price
Jiang et al. (2013)			✓		✓	✓		C_{max}	An algorithm with a tight worst-case ratio
Hasani et al. (2014a)					✓	✓		C_{max}	MIP formulation based on blocks
Hasani et al. (2014b)					✓	✓		C_{max}	SA and GA
Hasani et al. (2016)					✓	✓		C_{max}	Two greedy heuristics
Zhang et al. (2016)					✓		✓	$\sum C_j$	An approximate algorithm
Arnaout (2017)					✓	✓		C_{max}	Branch and bound, ACO
Hamzadayi and Yildiz (2017)				✓			✓	C_{max}	MIP formulation, SA and GA

Table 2 continued

Publications	Jobs			Setup type		Number of machines		Objective	Approach (comments)
	r_j	d_j	$prmp$	Dependent	Independent	$m = 2$	$m \geq 2$		
Cheng et al. (2017)			✓		✓	✓	✓	C_{max}	Complexity results, and a pseudo-polynomial time algorithm
Benmansour et al. (2018)					✓	✓		C_{max}	Equivalence with scheduling with time restriction (equal processing times)
Liu et al. (2019)					✓		✓	$\sum w_j C_j$	Branch and bound, lower bound
Alharkan et al. (2020)					✓	✓		C_{max}	TS and GPSO
Silva et al. (2019)				✓			✓	C_{max}	MIP formulation and an iterative local search metaheuristic
Elidrissi et al. (2021)					✓		✓	C_{max}	MIP formulations for general and regular job sets
This paper	✓	✓			✓		✓	L_{max}	MIP formulations, heuristic, GVNS and GRASP/VND

by Queyranne and Schulz (1994) to model the nonpreemptive 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 (Baker and Keller 2010; Anderson et al. 2013). 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 (Unlu and Mason 2010)].

The decision variables are defined as follows:

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

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

C_j: completion time of job j.

Let B be a sufficiently large positive integer, such as $B \geq \sum_{j \in N}(r_j + s_j + p_j)$. The problem $P, S1|r_j|L_{max}$ can be formulated as the following MIP_1.

$$\min \quad L_{max} \tag{1}$$

$$s.t. \quad L_{max} \geq C_j - d_j \qquad \forall j \in N \tag{2}$$

$$\sum_{j=1}^{n} x_{0,j} \leq m \tag{3}$$

$$\sum_{i=1}^{n} x_{i,0} \leq m \tag{4}$$

$$\sum_{j=0:j\neq i}^{n} x_{i,j} = 1 \qquad \forall i \in N \tag{5}$$

$$\sum_{i=0:i\neq j}^{n} x_{i,j} = 1 \qquad \forall j \in N \tag{6}$$

$$C_j \geq r_j + s_j + p_j \qquad \forall j \in N \tag{7}$$

$$C_i + s_j + p_j \leq C_j + B(1 - x_{i,j}) \qquad \forall i, j \in N, i \neq j \tag{8}$$

$$C_i + s_j + p_j \leq C_j + p_i + B(1 - z_{i,j}) \qquad \forall i, j \in N, i \neq j \tag{9}$$

$$z_{i,j} + z_{j,i} \geq \qquad \forall i, j \in N, i \neq j \tag{10}$$

$$x_{i,j} \in \{0, 1\} \qquad \forall i \in N \cup \{0\}, \forall j \in N \cup \{0\} \tag{11}$$

$$z_{i,j} \in \{0, 1\} \qquad \forall i, j \in N \tag{12}$$

The objective function (1) indicates that the maximum lateness has to be minimized. Constraints (2) stipulate that the maximum lateness is greater than or equal to the difference between C_j and d_j. Constraints (3) and (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 (5) and (6) guarantee that each job is scheduled on a particular machine. Constraints (7) state that each job j should starts after its release time. Constraints (8) indicate that no two jobs i and j, scheduled on the same machine, can overlap in time. Constraints (9) and (10) state that the server can set-up at most one job at a time. Constraints (11) and (12) define variables $x_{i,j}$ and $z_{i,j}$ as binaries.

3.2 Completion-time variables formulation (*MIP*$_2$)

We present here a MIP formulation based on completion-time variables for the problem $P, S1|r_j|L_{max}$. Completion-time variables, known also as natural-date variables, were initially used by Balas (1985) for a job shop scheduling problem. This formulation has been also used to model different $\mathcal{NP}$-hard scheduling problems (see Elidrissi et al. 2018; Krim et al. 2020; Elidrissi et al. 2022).

The decision variables are defined as follows:

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

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

The problem $P, S1|r_j|L_{max}$ can be formulated as the following MIP_2.

$$\min \quad L_{max} \tag{13}$$

$$s.t. \quad L_{max} \geq C_j - d_j \quad \forall j \in N \tag{14}$$

$$\sum_{k=1}^{m} y_{j,k} = 1 \quad \forall j \in N \tag{15}$$

$$C_j \geq r_j + s_j + p_j \quad \forall j \in N \tag{16}$$

$$C_i + s_j + p_j \leq C_j + B(3 - y_{i,k} - y_{j,k} - z_{i,j}) \quad \forall i, j \in N, i < j, \forall k \in M \tag{17}$$

$$C_j + s_i + p_i \leq C_i + B(2 - y_{i,k} - y_{j,k} + z_{i,j}) \quad \forall i, j \in N, i < j, \forall k \in M \tag{18}$$

$$C_i + s_j + p_j \leq C_j + p_i + B(1 - z_{i,j}) \quad \forall i, j \in N, i < j \tag{19}$$

$$C_j + s_i + p_i \leq C_i + p_j + Bz_{i,j} \quad \forall i, j \in N, i < j \tag{20}$$

$$y_{j,k} \in \{0, 1\} \quad \forall j \in N, \forall k \in M \tag{21}$$

$$z_{i,j} \in \{0, 1\} \quad \forall i, j \in N \tag{22}$$

In this formulation, the objective function (13) indicates that the maximum lateness has to be minimized. Constraints (14) indicate that the maximum lateness is greater than or equal to the difference between the completion time and the due date of each job. In order to guarantee that each job is scheduled on exactly one machine, constraints set (15) is added to the formulation. The completion time C_j is greater than or equal to the sum of the release, the setup, and the processing times of the job j, and is calculated according to constraints (16). Constraints (17) to (20) show that a job can be processed only if the server and the machines are available simultaneously. Constraints (17) and (18) indicate that no two jobs, scheduled on the same machine, can overlap in time. Constraints (19) and (20) state that the server can set-up at most one job at a time. Finally, constraints (21) and (22) define binary variables $x_{i,k}$ and $z_{i,j}$.

3.3 Valid inequalities

In order to reduce the time required to solve the problem by one of the proposed formulations MIP_1 and MIP_2, the following constraint set can be added. Let L^*_{max} denotes the objective-function value of an optimal solution of the problem $P, S1|r_j|L_{max}$. Let $\hat{L}^{H1}_{max}$ denotes the objective-function value of the solution provided by the constructive heuristic $H1$ for the problem $P, S1|r_j|L_{max}$. The constructive heuristic $H1$ is presented in Sect. 4.2.

Proposition 1 *The following constraints are valid for MIP_1 and MIP_2 formulations.*

$$C_j - C_i \geq r_j + s_j + p_j - (\hat{L}^{H1}_{max} + d_i) \qquad \forall i, j \in N, i \neq j \tag{23}$$

Proof The maximum lateness verifies the following equation:

$$C_j - d_j \leq L^*_{max} \leq \hat{L}^{H1}_{max} \qquad \forall j \in N \tag{24}$$

Combining the constraints sets 24 and 7, we obtain:

$$r_j + s_j + p_j \leq C_j \leq \hat{L}^{H1}_{max} + d_j \qquad \forall j \in N, \tag{25}$$

and accordingly, the constraints set is valid. □

MIP^*_1 refers to formulation MIP_1 with constraints set (23). MIP^*_2 refers to formulation MIP_2 with constraints set (23). A comparative study among MIP_1, MIP_2, MIP^*_1, and MIP^*_2 is performed in Sect. 5.

3.4 Illustrative example

We illustrate the previous formulation for an instance with $n = 6$, and $m = 3$. 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 3. It takes 0.44 s to solve the instance using the above MIP_1 formulation on IBM ILOG CPLEX 12.6. The optimal objective-function value is 6, and the obtained schedule is given in Fig. 1.

4 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 $H1$.

Table 3 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

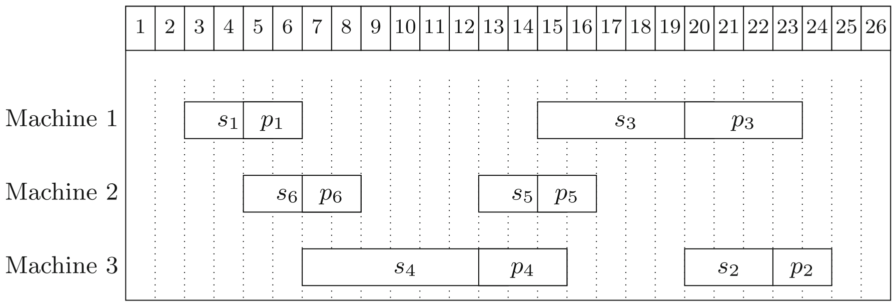

Fig. 1 Optimal schedule for the considered instance with 6 jobs and 3 machines

Finally, we present two metaheuristics: GVNS and GRASP, both using VND as an intensification operator. Note that $H1$ is used to generate initial solutions for GVNS.

4.1 Solution representation and objective-function calculation

A schedule of the problem $P, S1|r_j|L_{max}$ can be represented 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 kth position by the server (see Hasani et al. 2014b; Elidrissi et al. 2019). Any permutation of all jobs defines a feasible schedule, where each job will be scheduled as soon as it is released, and only if a machine and the server are available simultaneously. This is an indirect solution representation as it requires the scheduling of the jobs on the machines (while taking into account the server constraint) in order to compute the value of the maximum lateness (L_{max}) and the completion times of the jobs. Having in mind that all jobs are independent, all permutations ($n!$) represent feasible solutions, and therefore the search space is very large.

Additional notation is defined in Table 4.

In order to compute the objective function of a given sequence π of jobs, denoted as $L_{max}(\pi)$, the following Proposition 2 is used.

Proposition 2 *Given a sequence π of jobs, S_{π_k} is computed as follows:*

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

Table 4 Employed notation

Terms	Definition
E_{t,π_k}	Incumbent completion time of machine $t \in M$ (knowing the job we want to schedule at position π_k)
S_{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

Proof The first job will start immediately at its release date. Thus, $S_{\pi_k} = r_{\pi_k}$ if $k = 1$.

Second, each job in position $k \in \{2, \dots, m\}$ 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 $\{1, \dots, m\}$ available machines. This is trivial because in such cases, at least one machine will be available for processing this job. Therefore: $S_{\pi_k} = \max\left(r_{\pi_k}, S_{\pi_{k-1}} + s_{\pi_{k-1}}\right)$, $\forall k \in \{2, \dots, m\}$.

Third, suppose that we want to schedule a job in position $k \geq m + 1$. Its start time S_{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 $S_{\pi_k} \geq r_{\pi_k}$.

Moreover, the server will be available to perform the setup operation of the job at position k if $S_{\pi_k} \geq S_{\pi_{k-1}} + s_{\pi_{k-1}}$. 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, $t = \arg\min_{1\leq t'\leq m}\left(E_{t',\pi_k}\right)$.

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}, $S_{\pi_{k-1}} + s_{\pi_{k-1}}$ and $\min\limits_{1\leq t\leq m} E_{t,\pi_k}$.

□

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

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

It can be noticed that Proposition 2 can be used as a dispatching rule for other scheduling problems with a single server involving different objective functions.

4.2 Constructive heuristic (*H*1)

The idea of the heuristic $H1$ relies on the work of Lin et al. (2011) for the problem $P|ST_{sd}, r_j|L_{max}$, considering parallel machines with job release dates, sequence-dependent setup times and maximum lateness minimization. Indeed, in Lin et al. (2011), the authors developed a simple heuristic in order to generate an initial solution for an iterated greedy algorithm. Thus, we adapt this heuristic to our problem $P, S1|r_j|L_{max}$ by taking into account the single server, and sequence-independent setup times constraints. 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. 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 (if such a job does not exist, the job with the smallest release date is chosen). 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 ($k > m$) 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 the position k must be released before a given time $\max(T_{\pi_{k-1}} + s_{\pi_{k-1}}, E_{t,\pi_k})$, where $t = \arg\min_{h \in M}(E_{h,\pi_k})$. 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 General Variable Neighborhood Search (GVNS)

Variable Neighborhood Search (VNS) is a metaheuristic initially proposed by Mladenović and Hansen (1997). 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 (Hansen et al. 2008; Bierlaire et al. 2010; Perron et al. 2010; Mladenović et al. 2013; Schneider et al. 2015; Thevenin and Zufferey 2019; Todosijević et al. 2016). 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 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 $i = 1$ and $\pi = \pi''$.

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, the new current solution becomes π'' (as we set $\pi = \pi''$ in step (3)), and 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 $\mathcal{N}_1$).

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 (Hansen et al. 2017). In General VNS (GVNS), VND is used as a local search (Hansen et al. 2010). Following this research line, in this paper, 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.

4.3.1 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 $P, S1|r_j|L_{max}$. These neighborhood structures have been widely used in the literature [e.g., for the two parallel machines scheduling problem with a single server (Hasani et al. 2014b; Alharkan et al. 2020), for other scheduling problems on parallel machines (Respen et al. 2017; Thevenin et al. 2016; Thevenin and Zufferey 2019)].

- $\mathcal{N}_1(\pi) = \text{Swap}(\pi)$. 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-1} with π_j and π_i with π_{j+1}.

4.3.2 Variable Neighborhood Descent (VND)

We propose to use $\mathcal{N}_1$, $\mathcal{N}_2$ and $\mathcal{N}_3$ in a VND framework. The output solution will be a local optimum with respect to all the proposed neighborhood structures. The employed VND pseudocode is presented in Algorithm 1. Preliminary experiments led to the following settings. Such experiments are not reported here as they concern minor parameters with respect to the overall proposed approaches. First, the following sequence of the neighborhood structures is the most efficient: $\mathcal{N}_3, \mathcal{N}_1, \mathcal{N}_2$. Second, when generating a solution $\pi' \in \mathcal{N}_i$ in step (1), the first-improvement process (i.e., in each iteration, stop the generation of neighbor solutions as soon as the current solution can be improved) is better than the best-improvement process (i.e., in each iteration, generate all the neighbor solutions and pick up the best one). Third, and in line with Hansen et al. (2017), for step (3) of VND, we have tested three different techniques for changing the employed neighborhood structure. The techniques are summarized below, and Pipe turns out to be the best technique in our context.

– *Cyclic* set $k = k + 1$. In other words, the search mechanism employs always the next neighborhood structure of the list.
– *Pipe* set $k = k + 1$ if there is no improvement in step (2). In this case, the search mechanism keeps the same neighborhood structure as long as it is successful.
– *Sequential* set $k = k + 1$ if there is no improvement in step (2); otherwise set $k = 1$. It means that an improvement imposes the search mechanism to come back to the first neighborhood structure of the list.

```
Algorithm 1: VND
  Initialization: Construct a solution $\pi$ with $H1$ (or read the input solution), and
  set $i = 3$.
  While $\pi$ can be improved, do:
    1. Generate a solution $\pi' \in \mathcal{N}_i$.
    2. If $\pi'$ is better than $\pi$, set $\pi = \pi'$.
    3. Change the neighborhood type.
  Return the local optimum $\pi$ with respect to $\mathcal{N}_1, \mathcal{N}_2, \mathcal{N}_3$.
```

4.3.3 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 $\mathcal{N}_3$ (we have selected $\mathcal{N}_3$ because it modifies the structure of the considered solution more deeply than $\mathcal{N}_1$ and $\mathcal{N}_2$, which is in line with the role of a shaking mechanism). In other words, h iterations are performed in $\mathcal{N}_3$. In each iteration, a single neighbor solution is generated at random. The overall pseudocode of GVNS is given in Algorithm 2. The stopping criterion is a CPU time limit T. The diversification (resp. intensification) ability of GVNS relies on the shaking phase (resp. VND).

In line with the VNS literature, we have decided to employ increasing values of h if step 3 of GVNS does not lead to an improvement of the current solution π. More precisely, we start with $h = h_{min}$. Next, if there is no improvement in step 3, we augment h by Δh (as long as h does not exceed h_{max}) because more modifications seem to be required to escape from the current local optimum (and we impose the threshold h_{max} for h in order to control the search process and not make the shaking process resemble a restart mechanism). In contrast, anytime there is an improvement of π in step 3, we come back to smaller modifications by setting $h = h_{min}$ again (to allow the search process exploiting this new promising region of the solution space). After preliminary experiments that are not reported here, we have decided to set $(h_{min}, h_{max}, \Delta h) = (2, 10, 1)$.

```
Algorithm 2: GVNS
  Initialization: construct an initial solution $\pi$ with $H1$.
  While the computing time limit $T$ is not reached, do:
    1. Apply the shaking procedure to generate a solution $\pi'$ from $\pi$.
    2. Apply VND on $\pi'$, and let $\pi''$ be the resulting solution.
    3. If $\pi''$ is better than $\pi$, set $\pi = \pi''$.
  Return the local optimum $\pi$ with respect to $\mathcal{N}_1, \mathcal{N}_2, \mathcal{N}_3$.
```

4.4 Greedy randomized adaptive search procedures (GRASP) with VND

GRASP (Feo and Resende 1995) is a multi-start metaheuristic used to solve hard optimization problems. It has been applied for different scheduling problems (Yepes-Borrero et al. 2020; Báez et al. 2019; Armentano and de Franca Filho 2007). As presented in Algorithm 3, it consists of two phases which are repeated in turn as long a stopping condition is not met: (1) a greedy randomized construction phrase CP (presented below in Algorithm 4); (2) an improvement procedure (VND in our case). The diversification (resp. intensification) ability of GRASP relies on CP (resp. VND).

```
Algorithm 3: GRASP
  While the computing time limit $T$ is not reached, do:
    1. Apply the construction phase $CP$ to generate a solution $\pi$.
    2. Apply VND on $\pi$.
  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 Eq. (26), where α is the randomness parameter. The incremental cost is employed 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 $\Delta(j) \leq \Delta^{min} + \alpha(\Delta^{max} - \Delta^{min})$.

$$\Delta(j) = \max(T_{\pi_r} + s_{\pi_r}, \min_{1 \leq t \leq m} E_{t,\pi_k}, r_j) + s_j + p_j - d_j \tag{26}$$

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 4: Construction Phase CP of GRASP

```
Initialization: Set $\pi = \emptyset$, $CL = N$, and $\alpha$ to a value generated randomly in interval [0, 1].
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 in $\pi$ on the 1st available machine at the earliest time.
    5. Remove $j$ from $CL$.
Return solution $\pi$.
```

5 Computational experiments

In this section, the performances of MIP_1, MIP_1^*, MIP_2, MIP_2^*, $H1$, VND, GVNS, and GRASP are compared. The MIP formulations were solved using the concert technology library of CPLEX 12.6 using default settings in C++, whereas $H1$, VND, GVNS and GRASP were coded in C++. We use a personal computer Intel(R) Core(TM) with i7-4600 M 2.90 GHz CPU and 16GB of RAM, running Windows 7. Except for the small-sized instances for which one run is sufficient, the metaheuristics were executed 10 times in all experiments, and average results are provided. The stopping conditions employed for all the methods are presented in Table 5. Up to one hour of computing time is in line with the current practice in the job-scheduling industry (Respen et al. 2016; Thevenin et al. 2016, 2017). Moreover, for the problem $P2, S1|p_j, s_j|C_{max}$, Gan et al. (2012) proposed a stopping time of $\frac{300}{8} \cdot n$, and 3600 s for instances with $n \geq 100$. In this paper, we use the same limits for the computation times.

5.1 Benchmark instances

As far as we know, there are no publicly available benchmark instances in the literature for the problem $P, S1|r_j|L_{max}$. Therefore, we have generated a set of instances according to the existing literature, as proposed by Bektur and Saraç (2019). These instances are publicly available at https://sites.google.com/view/dataforps1rjlmax/accueil.

The instances are characterized by the following features:

- The number of jobs $n \in \{8, 10, 12, 15, 20, 25, 30, 50, 75, 100, 250, 500\}$.
- The number of machines $m \in \{2, 3, 4\}$.
- The integer processing times p_j are uniformly distributed in the interval [10, 100] .
- The integer setup times s_j are uniformly distributed in the interval [5, 50].
- Due dates d_j are uniformly generated based on 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),

Table 5 Stopping conditions of the MIP formulations, $H1$, VND, GVNS and GRASP

Method	Stopping condition		
	Small-sized instances $n \in \{8, 10\}$	Medium-sized instances $n \in \{12, 15, 20, 25, 30\}$	Large-sized instances $n \in \{50, 75, 100, 250, 500\}$
MIP formulations	Until an optimal solution is found	3600 s	3600 s
$H1$	One run (up to 0.0001 s)	One run (up to 0.001 s)	One run (up to 0.002 s)
GVNS	Same computing time	$\frac{300}{8} \cdot n$ s	$\frac{300}{8} \cdot n$ s (but 3600 s for $n \geq 100$)
GRASP	as the time required by MIP		
VND	to find an optimal solution		

and the due-date range factor R such that $d_j = r_j + s_j + p_j + U(\bar{d} - R\bar{d}, \bar{d})$. In line with Bektur and Saraç (2019), we have set $\tau \in \{0.65, 0.8\}$ and $R = 0.2$.
- The release dates are generated with a uniform distribution: $r_j = U(0, \bar{d})$.

For each combination of n, $m \in \{2, 3, 4\}$ and $\tau \in \{0.65, 0.8\}$, an instance was created, resulting to 6 instances (composing a group of instances) for each n. Therefore, we have a total of 72 instances, leading to a total of 12 groups of instances. The parameters of the test problems are given in Table 6. Due to the $\mathcal{NP}$-hard nature of the problem, and using the MIP formulations (MIP_1, MIP_1^*, MIP_2 and MIP_2^*), optimal solutions were found for small-sized instances ($n \in \{8, 10\}$), feasible solutions were obtained for medium-sized instances ($n \in \{12, 15, 20, 25, 30\}$), but no feasible solution was found for large-sized instances ($n \in \{50, 75, 100, 250, 500\}$). Therefore, $H1$, VND, GVNS and GRASP were designed to solve medium-sized and large-sized instances.

5.2 Results of the exact methods

In Table 7, we compare the CPLEX performance of models MIP_1, MIP_1^*, MIP_2, and MIP_2^* for 7 instance groups ($g1, \ldots, g7$) with a time limit of 1 h (since no feasible solution is found for the instance groups $g8$ to $g12$). For each MIP formulation, the following information is given: the number of instances solved to optimality within 1 h, #opt; the average computing time for these optimally solved instances, CPU; the number of instances unsolved within 1 h (instances with feasible solutions), #fs; and the average optimality gap for the instances which could not be solved within 1 h, gap (%).

The following observations can be made:

Table 6 Parameters of the test problems

Group no	Instance no	n	m	τ	Group no	Instance no	n	m	τ	Group no	Instance no	n	m	τ
$g1$	I1	8	2	0.65	$g5$	I25	20	2	0.65	$g9$	I49	75	2	0.65
	I2			0.80		I26			0.80		I50			0.80
	I3		3	0.65		I27		3	0.65		I51		3	0.65
	I4			0.80		I28			0.80		I52			0.80
	I5		4	0.65		I29		4	0.65		I53		4	0.65
	I6			0.80		I30			0.80		I54			0.80
$g2$	I7	10	2	0.65	$g6$	I31	25	2	0.65	$g10$	I55	100	2	0.65
	I8			0.80		I32			0.80		I56			0.80
	I9		3	0.65		I33		3	0.65		I57		3	0.65
	I10			0.80		I34			0.80		I58			0.80
	I11		4	0.65		I35		4	0.65		I59		4	0.65
	I12			0.80		I36			0.80		I60			0.80
$g3$	I13	12	2	0.65	$g7$	I37	30	2	0.65	$g11$	I61	250	2	0.65
	I14			0.80		I38			0.80		I62			0.80
	I15		3	0.65		I39		3	0.65		I63		3	0.65
	I16			0.80		I40			0.80		I64			0.80
	I17		4	0.65		I41		4	0.65		I65		4	0.65
	I18			0.80		I42			0.80		I66			0.80
$g4$	I19	15	2	0.65	$g8$	I43	50	2	0.65	$g12$	I67	500	2	0.65
	I20			0.80		I44			0.80		I68			0.80
	I21		3	0.65		I45		3	0.65		I69		3	0.65
	I22			0.80		I46			0.80		I70			0.80
	I23		4	0.65		I47		4	0.65		I71		4	0.65
	I24			0.80		I48			0.80		I72			0.80

- For the instance group $g1$, formulations MIP_1, MIP_1^*, MIP_2, and MIP_2^* are able to produce an optimal solution for all instances. For the formulations MIP_2 and MIP_2^*, CPLEX is able to produce an optimal solution in less computational time in comparison with the formulations MIP_1 and MIP_1^*.
- For the instance group $g2$, formulations MIP_1, MIP_1^*, MIP_2, and MIP_2^* are able to produce an optimal solution for all instances. MIP_2^* is able to produce an optimal solution in less computational time than the other formulations.
- For the instance group $g3$, MIP_1, MIP_2, and MIP_2^* are able to produce an optimal solution for all instances. MIP_1^* is able to produce an optimal solution for only 3 instances. Again, MIP_2^* works faster than the other formulations.
- For the instance group $g4$, MIP_2 and MIP_2^* are able to find an optimal solution for only one instance. MIP_2^* produces, on average, much smaller gaps than MIP_2. In addition, MIP_1 and MIP_1^* produced a feasible solution for all instances.
- For the instance groups $g5$ to $g7$ no formulation is able to find an optimal solution for all instances. On average, formulation MIP_2^* produces much smaller gaps than the other formulations.

Overall, the results showed that MIP_2^* outperforms, on average, the other formulations on almost all instances. This highlights the positive impact of the strengthening constraints in Eq. (23). Therefore, in the next experiments, we compare only MIP_2^* with the other approaches.

5.3 Results of the solution methods

5.3.1 Comparison of MIP_2^*, $H1$, VND, GVNS and GRASP for the small-sized instances

Note first that for all tables, an empty cell means that the same value as the above one is kept, and all the computing times are indicated in seconds. In Table 8, we compare the performance of MIP_2^*, $H1$, VND, GVNS and GRASP for small-sized instances, where an optimal solution can be found by MIP_2^* within 1 h. 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 optimal value L_{max}^{opt} of L_{max} (found with the formulation MIP_2^*). Next, the obtained value of L_{max} is given for $H1$ (but not the computing time, as it is always below 0.0001 s) and VND (but not the computing time, as it is always below 0.01 s). Finally, the computing time to find an optimal solution is given for the MIP_2^*, 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 significantly less computing time than the formulation MIP_2^*.
- VND is able to find an optimal solution for four instances, namely: I1, I5, I6, I11.
- $H1$ is never able to find an optimal solution (its average gap to optimality is around 25%), except for instance I6.
- GVNS requires the smallest average computing time (0.31 s) to find optimal solutions.

Table 7 Results of MIP_1, MIP_1^*, MIP_2, and MIP_2^*

	MIP_1				MIP_1^*				MIP_2				MIP_2^*			
	#opt	CPU	#fs	gap(%)	#opt	CPU	#fs	gap(%)	#opt	CPU	#fs	gap(%)	#opt	CPU	#fs	gap(%)
$g1$	6	0.97	0	0	6	0.94	0	0	6	0.92	0	0	6	0.92	0	0
$g2$	6	32.15	0	0	6	24.21	0	0	6	7.01	0	0	6	6.08	0	0
$g3$	3	1242.91	3	68.21	3	1159.77	3	79.15	6	397	0	0	6	81.42	0	0
$g4$	0	–	6	96	0	–	6	90.99	1	317.62	5	53.26	1	288.38	5	40.37
$g5$	0	–	6	99.38	0	–	5	100	0	–	6	87.79	0	–	6	80.03
$g6$	0	–	2	100	0	–	2	100	0	–	5	96.82	0	–	5	95.54
$g7$	0	–	3	100	0	–	3	100	0	–	4	100	0	–	4	100

Table 8 Results of MIP_2^*, $H1$, VND, GVNS and GRASP for the small-sized instances

Instance					$H1$	VND	MIP_2^*	GVNS	GRASP
ID	n	m	τ	L_{max}^{opt}	L_{max}		Time		
I1	8	2	0.65	79	112	79	0.97	0.12	0.04
I2			0.80	101	151	111	0.87	0.00	0.00
I3		3	0.65	101	128	107	1.18	0.00	0.01
I4			0.80	95	144	102	0.64	0.09	0.00
I5		4	0.65	49	50	49	0.85	0.00	0.00
I6			0.80	115	115	115	0.96	0.00	0.00
I7	10	2	0.65	125	141	133	7.48	0.11	0.02
I8			0.80	185	222	201	12.30	2.91	3.91
I9		3	0.65	77	105	82	3.55	0.01	0.51
I10			0.80	122	160	134	4.11	0.08	1.37
I11		4	0.65	62	62	62	1.21	0.00	0.00
I12			0.80	112	134	113	7.80	0.39	0.01
Avg.				101.92	127	107.33	3.49	**0.31**	0.49

The best average results are indicated in bold face

5.3.2 Comparison of MIP_2^*, $H1$, VND, GVNS and GRASP for the medium-sized instances

Table 9 presents the results for the medium-sized instances (the best results are indicated in bold face). The instance characteristics are first indicated. For MIP_2^*, the following information is given: the lower bound $LB_{MIP_2^*}$, the upper bound $UB_{MIP_2^*}$, the percentage gap to optimality $Gap_{MIP_2^*}(\%)$, and the time requested to prove optimality (if below 1 h). Note that for the instances I35 to I37, no information is provided as CPLEX is not able to return a feasible solution. Next, the found value of L_{max} is given for $H1$ and VND (with its associated computing time). Finally, the following results are showed for GVNS and GRASP: the best (resp. average) objective-function value over 10 runs denoted as $L_{max}^{\star}$ (resp. L_{max}^{avg}). The average computing times are also provided (computed over the 10 runs, and the computing time of a run corresponds to the time at which the best visited solution is found).

The following observations can be made:

- CPLEX (i.e., the MIP_2^* formulation) is able to prove the optimality of the obtained solutions for the instances I16 to I18.
- The limitations of CPLEX seem to start from $n = 15$, and they are obvious from $n = 30$.
- GVNS and GRASP outperform significantly MIP_2^* both in terms of quality (i.e., objective-function values) and speed (i.e., time requested to find competitive solutions). Moreover, for the instances I16 to I18, GVNS and GRASP are also able to find optimal solutions, but within 25 s.
- The results of $H1$ allow to measure the benefit of GVNS, as the latter employs the former to generate an initial solution.

Table 9 Results of MIP_2^*, $H1$, VND, GVNS and GRASP for the medium-sized instances

Instance				MIP_2^*				$H1$	VND		GVNS			GRASP		
ID	n	m	τ	$LB_{MIP_2^*}$	$UB_{MIP_2^*}$	$Gap_{MIP_2^*}(\%)$	Time	L_{max}	L_{max}	Time	$L_{max}^{\star}$	L_{max}^{avg}	Time	$L_{max}^{\star}$	L_{max}^{avg}	Time
I13	12	2	0.65	170	170	0	40.03	223	190	0.01	**170**	170	0.29	**170**	170	2.49
I14			0.80	231	231	0	80.73	294	246	0.01	**231**	231	1.40	**231**	231	10.83
I15		3	0.65	115	115	0	181.70	137	130	0.01	**115**	115	0.06	**115**	115	0.49
I16			0.80	164	164	0	50.86	217	185	0.01	**164**	164	1.23	**164**	164	22.70
I17		4	0.65	59	59	0	20.07	93	60	0.02	**59**	60.8	0.10	**59**	59	0.17
I18			0.80	102	102	0	115.08	158	107	0.04	**102**	102	0.25	**102**	102	3.27
I19	15	2	0.65	93	120	22.50	3600	193	137	0.05	**120**	120	11.67	**120**	120	40.75
I20			0.80	125	307	59.28	3600	359	330	0.03	**307**	307	172.09	**307**	307	220.65
I21		3	0.65	96	136	29.41	3600	173	158	0.02	**136**	136.70	144.67	**136**	137	185.86
I22			0.80	143	217	34.10	3600	278	234	0.03	**215**	216.40	203.21	**215**	216.70	196.08
I23		4	0.65	76	76	0	288.38	106	85	0.03	**76**	76	0.15	**76**	76	0.49
I24			0.80	89.5	206	56.55	3600	259	209	0.02	**206**	206	0.04	**206**	206	0.58
I25	20	2	0.65	0	279	100	3600	335	316	0.06	278	280.20	212.56	**277**	280.40	308.76
I26			0.80	132.2	486	72.8	3600	563	509	0.11	**484**	485.20	572.56	485	487	307.50
I27		3	0.65	0	172	100	3600	235	194	0.10	**172**	172	7.97	**172**	172	15.25
I28			0.80	88.1	277	68.18	3600	350	300	0.10	272	274.10	347.87	**271**	275.60	269.73
I29		4	0.65	0	175	100	3600	189	175	0.06	**175**	175	0.17	**175**	175	0.97
I30			0.80	132	323	59.14	3600	370	328	0.07	**323**	323	0.07	**323**	323	0.09

Table 9 continued

Instance				MIP_2^*				$H1$	VND		GVNS			GRASP		
ID	n	m	τ	$LB_{MIP_2^*}$	$UB_{MIP_2^*}$	$Gap_{MIP_2^*}(\%)$	Time	L_{max}	L_{max}	Time	$L^\star_{max}$	L^{avg}_{max}	Time	$L^\star_{max}$	L^{avg}_{max}	Time
I31	25	2	0.65	0	302	100	3600	372	376	0.38	**300**	302	532.85	302	303.8	259.87
I32			0.80	–	–	–	–	660	648	0.31	**605**	607.7	343.95	607	609.5	301.10
I33		3	0.65	0	231	100	3600	342	305	0.12	**234**	239.90	312.66	237	240.70	496.92
I34			0.80	64	403	84.11	3600	474	432	0.18	**392**	397	340.18	397	399.70	437.44
I35		4	0.65	0	209	100	3600	221	222	0.16	**209**	209	0.162	**209**	209	1.06
I36			0.80	27.2	423	93.56	3600	441	423	0.11	**423**	423	0.141	**423**	423	0.68
I37	30	2	0.65	–	–	–	–	488	395	0.54	**354**	356.80	326.65	**354**	358.40	360.27
I38			0.80	–	–	–	–	792	721	0.42	**672**	679.60	663.26	678	681.70	460.08
I39		3	0.65	0	296	100	3600	371	313	0.47	**275**	284.10	642.47	283	288.20	599.65
I40			0.80	0	528	100	3600	649	544	0.36	**500**	505.60	564.86	503	509.90	606.34
I41		4	0.65	0	210	100	3600	228	227	0.45	**203**	204	421.98	205	207.60	306.49
I42			0.80	0	505	100	3600	822	505	0.33	**485**	503	0.27	505	505	0.46

- GVNS and GRASP outperform VND significantly (except for the instances I29 and I34). This shows the benefit of all the ingredients (e.g., diversification mechanisms) added to VND to derive GVNS and GRASP.
- GVNS and GRASP have a similar performance. In addition, the difference between L_{max}^{avg} and $L_{max}^{\star}$ is very small (often below one unit).
- GVNS is able to obtain the best results for all instances but two (with $n = 20$).

5.3.3 Comparison of VND, GVNS and GRASP for the large-sized instances

Table 10 has the same structure as Table 9, and the best average results are indicated in bold face. It presents the results for VND, GVNS and GRASP, which are the method specifically designed for tackling the large-sized instances (indeed, MIP_2^* cannot find a feasible solution for such instances, and the role of $H1$ is simply to generate an initial solution for GVNS).

The following observations can be made:

- Overall, for the provided instances, the methods can be ranked as follows: GVNS $>$ GRASP $>$ VND. This ranking is very obvious for $n \in \{250, 500\}$.
- GVNS and GRASP have comparable objective-function values for $n \in \{50, 75, 100\}$. However, for such instances, GVNS requires generally less computing time to find its best solutions. In contrast, GVNS significantly outperforms GRASP for the larger instances ($n \in \{250, 500\}$).
- GVNS has a best $L_{max}^{\star}$ value for 26 instances, whereas GRASP has best values for only 8 instances.
- For GVNS and GRASP, the difference between L_{max}^{avg} and $L_{max}^{\star}$ grows with n and m, which is likely to indicate the robustness degradation with the increase of the instance size (i.e., with the increase of the problem complexity). This degradation is however smaller for GVNS when compared to GRASP.

5.4 Discussion

Table 11 presents the performance of VND, GVNS and GRASP in terms of the percentage deviation from the best-known solutions according to the due date tightness factor τ and the number n of jobs. In order to compute each percentage deviation, two values of $L_{max}^{\star}$ are compared: the best one over all the runs of all the methods, and the one obtained by the considered method. Again, we can easily observe the superiority of GVNS over GRASP (in particular for the large-sized instances), and both methods outperforms their intensification operator VND. Furthermore, VND and GRASP perform better with $\tau = 0.85$ rather than with $\tau = 0.65$ (with respect to the deviation from the best-known solutions). This might indicate that these methods are less efficient with tighter due dates.

Table 10 Results of VND, GVNS and GRASP for the large-sized instances

Instance				VND		GVNS			GRASP		
ID	n	m	τ	L_{max}	Time	$L^{\star}_{max}$	L^{avg}_{max}	Time	$L^{\star}_{max}$	L^{avg}_{max}	Time
I43	50	2	0.65	734	3.54	**649**	657	859.36	**649**	658.1	900.27
I44			0.80	1144	3.34	**1099**	1108.5	1329.31	1104	1110.4	772.58
I45		3	0.65	592	3.35	**474**	485.4	1219.91	489	494	1006.52
I46			0.80	919	5.13	**860**	870.1	792.88	861	873.9	1156.24
I47		4	0.65	449	3.80	413	419.6	985.90	**412**	418.1	852.97
I48			0.80	835	3.93	**799**	799.1	304.97	**799**	799.1	739.58
I49	75	2	0.65	1074	20.88	959	972.3	1132.61	**953**	973.1	1694.09
I50			0.80	1882	16.27	**1822**	1833.6	1771.70	1824	1834	1751.26
I51		3	0.65	740	32.10	**657**	668.5	876.64	662	675.2	1677.99
I52			0.80	1472	25.00	1410	1422	1577.48	**1391**	1412.9	1261.26
I53		4	0.65	760	28.75	**731**	731	374.05	**731**	731.3	835.65
I54			0.80	1240	21.72	**1219**	1219	597.15	**1219**	1219	644.50
I55	100	2	0.65	1519	88.29	**1377**	1418.6	1886.62	1390	1418.6	2099.04
I56			0.80	2584	57.70	**2510**	2525.8	1599.41	2531	2542.4	1609.93
I57		3	0.65	1302	90.09	1179	1200.9	1392.69	**1172**	1201	1661.77
I58			0.80	1901	112.35	**1811**	1844.1	1272.00	1826	1840.4	1084.43
I59		4	0.65	833	111.06	**763**	774.7	1546.62	770	784.8	1610.01
I60			0.80	1638	82.13	**1617**	1628.7	1789.72	1620	1624.6	1863.66

Table 10 continued

Instance				VND		GVNS			GRASP		
ID	n	m	τ	L_{max}	Time	$L^{\star}_{max}$	L^{avg}_{max}	Time	$L^{\star}_{max}$	L^{avg}_{max}	Time
I61	250	2	0.65	6450	362.54	**3665**	3849.7	2430.31	5542	5646.8	2706.14
I62			0.80	7411	328.92	**6215**	6267.7	2498.25	6855	7029	3104.64
I63		3	0.65	4470	647.90	**2817**	2979.2	2515.00	3913	4205.8	2448.19
I64			0.80	5551	365.19	**4618**	4745.8	2756.68	4973	5304.9	3002.08
I65		4	0.65	3870	446.75	**2350**	2449.1	1266.34	3393	3527.6	2448.71
I66			0.80	4933	290.70	**4071**	4136.6	1949.93	4480	4634.3	2153.93
I67	500	2	0.65	12,829	1011.50	**7887**	7986.8	2190.17	12,135	12,639.3	2702.47
I68			0.80	16,050	1002.93	**13,570**	13,728.6	3607.65	15,920	16,334.8	2393.68
I69		3	0.65	10,014	1205.16	**6801**	6939.8	2160.92	9651	10,068.5	2523.17
I70			0.80	12,333	1045.49	**10,618**	10,706.4	3242.32	11,535	11,922.5	1972.36
I71		4	0.65	8244	1208.49	**4822**	4901.3	2161.68	7656	8103.6	2281.71
I72			0.80	10,407	1130.14	**8755**	8857.9	2521.52	9802	10,069.9	2510.19

Table 11 Comparison of VND, GVNS and GRASP in terms of percentage deviation from the best-known solutions

n	τ	Percentage deviation from the best solutions		
		VND	GVNS	GRASP
8	0.65	1.98	**0**	**0**
	0.80	5.76	**0**	**0**
10	0.65	4.30	**0**	**0**
	0.80	6.46	**0**	**0**
12	0.65	8.83	**0**	**0**
	0.80	8.07	**0**	**0**
15	0.65	14.06	**0**	**0**
	0.80	5.93	**0**	**0**
20	0.65	8.96	0.12	**0**
	0.80	5.80	0.12	**0.07**
25	0.65	18.28	**0**	0.64
	0.80	5.10	**0**	0.64
30	0.65	9.41	**0**	0.97
	0.80	5.05	**0**	1.40
50	0.65	11.79	**0.06**	0.79
	0.80	3.87	**0**	0.14
75	0.65	10.98	**0.16**	0.19
	0.80	3.26	0.34	**0.03**
100	0.65	7.81	**0.15**	0.47
	0.80	2.32	**0**	0.46
250	0.65	215.75	**0**	163.05
	0.80	23.74	**0**	12.60
500	0.65	409.24	**0**	354.14
	0.80	28.56	**0**	22.52
Avg.		34.39	**0.04**	23.25

The best average results are indicated in bold face

6 Conclusions and future works

In this work, 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 two mixed-integer-programming (MIP) formulations to solve optimally instances with up to 12 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 500 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, which involved three well-known neighborhood structures for such problems.

In line with the good practice of the related literature, we have built 72 benchmark instances to compare the proposed methods. For small-sized instances, GVNS and GRASP outperformed the MIP formulations in terms of the computing time to find an optimal solution. However, for medium and large-sized instances, the results show the very good performance of GVNS with respect to quality (of the obtained solutions), speed (i.e., time needed to generate efficient solutions) and some robustness indicators (e.g., deviation percentage from the best-known results, difference between the best and average solution values). This success can be explained by the good balance between the inherent diversification and intensification features of GVNS, namely the alternate and fine-tuned use of the VND operator for intensifying the search, and the shaking phase for diversifying the search. It also outlines that the exploration mechanism of GRASP (i.e., building another solution from scratch) might be too strong when compared to the shaking phase of GVNS. Indeed, the latter preserves a significant portion of the incumbent solution (a mechanism has been added to avoid the shaking phase resemble a restart mechanism, which results in well-controlled trajectory when exploring the solution space).

An avenue of research would be to adapt the proposed methods to other machine environments, such as dedicated and unrelated parallel machines, taking into account sequence-and-machine-dependent setup times. Further works could propose tight bounds on the machine impact for the problem $P, S1|r_j|L_{max}$ as done by Rustogi and Strusevich (2013) for the parallel machine scheduling problem to minimize the makespan and the total flow time.

Declarations

Conflict of interest No potential conflict of interest was reported by the authors.

References

Abdekhodaee AH, Wirth A (2002) Scheduling parallel machines with a single server: some solvable cases and heuristics. Comput Oper Res 29(3):295–315

Abdekhodaee AH, Wirth A, Gan HS (2004) Equal processing and equal setup time cases of scheduling parallel machines with a single server. Comput Oper Res 31(11):1867–1889

Abdekhodaee AH, Wirth A, Gan HS (2006) Scheduling two parallel machines with a single server: the general case. Comput Oper Res 33(4):994–1009

Alharkan I, Saleh M, Ghaleb MA, Kaid H, Farhan A, Almarfadi A (2020) Tabu search and particle swarm optimization algorithms for two identical parallel machines scheduling problem with a single server. J King Saud Univ-Eng Sci 32(5):330–338

Allahverdi A, Soroush H (2008) The significance of reducing setup times/setup costs. Eur J Oper Res 187(3):978–984

Anderson BE, Blocher JD, Bretthauer KM, Venkataramanan MA (2013) An efficient network-based formulation for sequence dependent setup scheduling on parallel identical machines. Math Comput Model 57(3–4):483–493

Armentano VA, de Franca Filho MF (2007) Minimizing total tardiness in parallel machine scheduling with setup times: an adaptive memory-based GRASP approach. Eur J Oper Res 183(1):100–114

Arnaout JP (2017) Heuristics for the two-machine scheduling problem with a single server. Int Trans Oper Res 24(6):1347–1355

Arnaout JP (2021) Worm optimisation algorithm to minimise the makespan for the two-machine scheduling problem with a single server. Int J Oper Res 41(2):270–281

Báez S, Angel-Bello F, Alvarez A, Melián-Batista B (2019) A hybrid metaheuristic algorithm for a parallel machine scheduling problem with dependent setup times. Comput Ind Eng 131:295–305

Baker KR, Keller B (2010) Solving the single-machine sequencing problem using integer programming. Comput Ind Eng 59(4):730–735

Balas E (1985) On the facial structure of scheduling polyhedra. In: Mathematical programming essays in honor of George B. Dantzig Part I. Springer, pp 179–218

Bektur G, Saraç T (2019) 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. Comput Oper Res 103:46–63

Benmansour R, Braun O, Artiba A (2014) On the single-processor scheduling problem with time restrictions. In: 2014 international conference on control, decision and information technologies (CoDIT). IEEE, pp 242–245

Benmansour R, Braun O, Hanafi S (2018) The single-processor scheduling problem with time restrictions: complexity and related problems. J Sched 22:1–7

Bierlaire M, Thémans M, Zufferey N (2010) A heuristic for nonlinear global optimization. INFORMS J Comput 22(1):59–70

Bish EK (2003) A multiple-crane-constrained scheduling problem in a container terminal. Eur J Oper Res 144(1):83–107

Braun O, Chung F, Graham R (2014) Single-processor scheduling with time restrictions. J Sched 17(4):399–403

Brucker P, Dhaenens-Flipo C, Knust S, Kravchenko SA, Werner F (2002) Complexity results for parallel machine problems with a single server. J Sched 5(6):429–457

Centeno G, Armacost RL (1997) Parallel machine scheduling with release time and machine eligibility restrictions. Comput Ind Eng 33(1–2):273–276

Cheng T, Kravchenko SA, Lin BM (2017) Preemptive parallel-machine scheduling with a common server to minimize makespan. Naval Res Logist 64(5):388–398

Cheng T, Kravchenko SA, Lin BM (2019) Server scheduling on parallel dedicated machines with fixed job sequences. Naval Res Logist (NRL) 66(4):321–332

Cheng T, Kravchenko SA, Lin BM (2021) Complexity of server scheduling on parallel dedicated machines subject to fixed job sequences. J Oper Res Soc 72(10):2286–2289

Elidrissi A, Benmansour R, Benbrahim M, Duvivier D (2018) MIP formulations for identical parallel machine scheduling problem with single server. In: 2018 4th international conference on optimization and applications (ICOA). IEEE, pp 1–6

Elidrissi A, Benbrahim M, Benmansour R, Duvivier D (2019) Variable neighborhood search for identical parallel machine scheduling problem with a single server. In: International conference on variable neighborhood search. Springer, pp 112–125

Elidrissi A, Benmansour R, Benbrahim M, Duvivier D (2021) Mathematical formulations for the parallel machine scheduling problem with a single server. Int J Prod Res 59(20):6166–6184

Elidrissi A, Benmansour R, Sifaleras A (2022) General variable neighborhood search for the parallel machine scheduling problem with two common servers. Optim Lett 1–31

Feo TA, Resende MG (1995) Greedy randomized adaptive search procedures. J Glob Optim 6(2):109–133

Gan HS, Wirth A, Abdekhodaee A (2012) A branch-and-price algorithm for the general case of scheduling parallel machines with a single server. Comput Oper Res 39(9):2242–2247

Glass CA, Shafransky YM, Strusevich VA (2000) Scheduling for parallel dedicated machines with a single server. Naval Res Logist 47(4):304–328

Graham RL, Lawler EL, Lenstra JK, Kan AR (1979) Optimization and approximation in deterministic sequencing and scheduling: a survey. Ann Discrete Math 5:287–326

Hall NG, Potts CN, Sriskandarajah C (2000) Parallel machine scheduling with a common server. Discrete Appl Math 102(3):223–243

Hamzadayi A, Yildiz G (2017) Modeling and solving static m identical parallel machines scheduling problem with a common server and sequence dependent setup times. Comput Ind Eng 106:287–298

Hansen P, Mladenović N, Moreno Perez JA (2008) Variable neighbourhood search: methods and applications. 4OR 6(4):319–360

Hansen P, Mladenović N, Pérez JAM (2010) Variable neighbourhood search: methods and applications. Ann Oper Res 175(1):367–407

Hansen P, Mladenović N, Todosijević R, Hanafi S (2017) Variable neighborhood search: basics and variants. EURO J Comput Optim 5(3):423–454

Haouari M, Gharbi A (2003) An improved max-flow-based lower bound for minimizing maximum lateness on identical parallel machines. Oper Res Lett 31(1):49–52

Hasani K, Kravchenko SA, Werner F (2014a) Block models for scheduling jobs on two parallel machines with a single server. Comput Oper Res 41:94–97

Hasani K, Kravchenko SA, Werner F (2014b) Simulated annealing and genetic algorithms for the two-machine scheduling problem with a single server. Int J Prod Res 52(13):3778–3792

Hasani K, Kravchenko SA, Werner F (2016) Minimizing the makespan for the two-machine scheduling problem with a single server: two algorithms for very large instances. Eng Optim 48(1):173–183

Heinz V, Novák A, Vlk M, Hanzálek Z (2022) Constraint programming and constructive heuristics for parallel machine scheduling with sequence-dependent setups and common servers. Comput Ind Eng 172:108586

Hsu PY, Lo SH, Hwang HG, Lin BM (2020) Scheduling of anaesthesia operations in operating rooms. Healthcare 9(6):3527–3533

Huang S, Cai L, Zhang X (2010) Parallel dedicated machine scheduling problem with sequence-dependent setups and a single server. Comput Ind Eng 58(1):165–174

Jiang Y, Dong J, Ji M (2013) Preemptive scheduling on two parallel machines with a single server. Comput Ind Eng 66(2):514–518

Kim MY, Lee YH (2012) MIP models and hybrid algorithm for minimizing the makespan of parallel machines scheduling problem with a single server. Comput Oper Res 39(11):2457–2468

Kim HJ, Lee JH (2021) Scheduling uniform parallel dedicated machines with job splitting, sequence-dependent setup times, and multiple servers. Comput Oper Res 126:108586

Kim CO, Shin HJ (2003) Scheduling jobs on parallel machines: a restricted Tabu search approach. Int J Adv Manuf Technol 22(3):278–287

Koulamas CP (1996) Scheduling two parallel semiautomatic machines to minimize machine interference. Comput Oper Res 23(10):945–956

Kravchenko SA, Werner F (1997) Parallel machine scheduling problems with a single server. Math Comput Model 26(12):1–11

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

Krim H, Benmansour R, Duvivier D, Aït-Kadi D, Hanafi S (2020) Heuristics for the single machine weighted sum of completion times scheduling problem with periodic maintenance. Comput Optim Appl 75(1):291–320

Lee JH, Kim HJ (2021) A heuristic algorithm for identical parallel machine scheduling: splitting jobs, sequence-dependent setup times, and limited setup operators. Flex Serv Manuf J 33:992–1026

Lee ZJ, Lin SW, Ying KC (2010) Scheduling jobs on dynamic parallel machines with sequence-dependent setup times. Int J Adv Manuf Technol 47(5):773–781

Lin SW, Lee ZJ, Ying KC, Lu CC (2011) Minimization of maximum lateness on parallel machines with sequence-dependent setup times and job release dates. Comput Oper Res 38(5):809–815

Liu GS, Li JJ, Yang HD, Huang GQ (2019) Approximate and branch-and-bound algorithms for the parallel machine scheduling problem with a single server. J Oper Res Soc 70:1554–1570

Mladenović N, Hansen P (1997) Variable neighborhood search. Comput Oper Res 24(11):1097–1100

Mladenović N, Urošević D, Hanafi S (2013) Variable neighborhood search for the travelling deliveryman problem. 4OR 11(1):57–73

Mokotoff E (2001) Parallel machine scheduling problems: a survey. Asia-Pac J Oper Res 18(2):193

Ovacik IM, Uzsoy R (1995) Rolling horizon procedures for dynamic parallel machine scheduling with sequence-dependent setup times. Int J Prod Res 33(11):3173–3192

Perron S, Hansen P, Le Digabel S, Mladenović N (2010) Exact and heuristic solutions of the global supply chain problem with transfer pricing. Eur J Oper Res 202(3):864–879

Queyranne M, Schulz AS (1994) Polyhedral approaches to machine scheduling. TU, Fachbereich 3, Berlin

Respen J, Zufferey N, Amaldi E (2016) Metaheuristics for a job scheduling problem with smoothing costs relevant for the car industry. Networks 67(3):246–261

Respen J, Zufferey N, Wieser P (2017) Three-level inventory deployment for a luxury watch company facing various perturbations. J Oper Res Soc 68(10):1195–1210

Rustogi K, Strusevich VA (2013) Parallel machine scheduling: impact of adding extra machines. Oper Res 61(5):1243–1257

Schneider M, Stenger A, Hof J (2015) An adaptive VNS algorithm for vehicle routing problems with intermediate stops. OR Spectrum 37(2):353–387

Schutten JM, Leussink R (1996) Parallel machine scheduling with release dates, due dates and family setup times. Int J Prod Econ 46:119–125

Silva JMP, Teixeira E, Subramanian A (2019) Exact and metaheuristic approaches for identical parallel machine scheduling with a common server and sequence-dependent setup times. J Oper Res Soc 72:1–15

Thevenin S, Zufferey N (2019) Learning variable neighborhood search for a scheduling problem with time windows and rejections. Discrete Appl Math 261:344–353

Thevenin S, Zufferey N, Widmer M (2016) Order acceptance and scheduling with earliness and tardiness penalties. J Heuristics 22(6):849–890

Thevenin S, Zufferey N, Glardon R (2017) Model and metaheuristics for a scheduling problem integrating procurement, sale and distribution decisions. Ann Oper Res 259(1–2):437–460

Todosijević R, Benmansour R, Hanafi S, Mladenović N, Artiba A (2016) Nested general variable neighborhood search for the periodic maintenance problem. Eur J Oper Res 252(2):385–396

Torjai L, Kruzslicz F (2016) Mixed integer programming formulations for the biomass truck scheduling problem. Cent Eur J Oper Res 24(3):731–745

Unlu Y, Mason SJ (2010) Evaluation of mixed integer programming formulations for non-preemptive parallel machine scheduling problems. Comput Ind Eng 58(4):785–800

Wang G, Cheng TE (2001) An approximation algorithm for parallel machine scheduling with a common server. J Oper Res Soc 52(2):234–237

Xu D, Li G, Zhang F (2021) Scheduling an automatic IoT manufacturing system with multiple servers. Comput Ind Eng 157:107343

Yepes-Borrero JC, Villa F, Perea F, Caballero-Villalobos JP (2020) Grasp algorithm for the unrelated parallel machine scheduling problem with setup times and additional resources. Expert Syst Appl 141:112959

Ying KC, Cheng HM (2010) Dynamic parallel machine scheduling with sequence-dependent setup times using an iterated greedy heuristic. Expert Syst Appl 37(4):2848–2852

Zhang A, Wang H, Chen Y, Chen G (2016) Scheduling jobs with equal processing times and a single server on parallel identical machines. Discrete Appl Math 213:196–206