Optimization Letters
https://doi.org/10.1007/s11590-022-01925-2

ORIGINAL PAPER

General variable neighborhood search for the parallel machine scheduling problem with two common servers

Abdelhak Elidrissi[1] · **Rachid Benmansour**[2] · **Angelo Sifaleras**[3]

Received: 14 July 2021 / Accepted: 20 August 2022

Abstract

We address in this paper the parallel machine scheduling problem with a shared loading server and a shared unloading server. Each job has to be loaded by the loading server before being processed on one of the available machines and unloaded immediately by the unloading server after its processing. The objective function involves the minimization of the overall completion time, known as the makespan. This important problem raises in flexible manufacturing systems, automated material handling, healthcare, and many other industrial fields, and has been little studied up to now. To date, research on it has focused on the case of two machines. The regular case of this problem is considered. A mixed integer programming formulation based on completion time variables is suggested to solve small-sized instances of the problem. Due to its $\mathcal{NP}$-hardness, we propose two greedy heuristics based on the minimization of the loading, respectively unloading, server waiting time, and an efficient General Variable Neighborhood Search (GVNS) algorithm. In the computational experiments, the proposed methods are compared using 120 new and publicly available instances. It turns out that, the proposed GVNS with an initial solution-finding mechanism based on the unloading server waiting time minimization significantly outperforms the other approaches.

Keywords Parallel machine scheduling · Scheduling with two common servers · Mixed integer program · General variable neighborhood search · Greedy heuristics

✉ Angelo Sifaleras
sifalera@uom.gr

Abdelhak Elidrissi
abdelhak.elidrissi@uir.ac.ma

Rachid Benmansour
r.benmansour@insea.ac.ma

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

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

[3] Department of Applied Informatics, University of Macedonia, School of Information Sciences, 156 Egnatias Str., 54636 Thessaloniki, Greece

Published online: 10 September 2022

1 Introduction

Parallel Machine Scheduling problem with a Single Server (PMSSS problem) has been widely studied in the last two decades (see [6, 11, 16, 18, 27]). In the PMSSS problem, a server which can represent a team of setup workers or a single operator, etc, is considered to be in charge of the setup operation (or loading operation) of jobs. The setup time resulting from the setup operation can be defined as the time required to prepare the necessary resources to perform a task [3]. The PMSSS problem has various applications, e.g., in supply chain [9, 35], in plastic injection industry [6], in printing industry [24], in semiconductor industry [27], and recently in health care [23].

Unlike the PMSSS problem, the problem considering both loading and unloading operations has been less studied in the literature. Indeed, in the latter problem, 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 few papers have been proposed in the literature for this variant of the problem. Xie et al. [37] 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 the largest processing time heuristic generated a tight worst-case bound of $3/2 - 1/2m$, where m is the number of available machines. Later, Jiang et al. [25] 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 ratios.

For the case with several servers, Kravchenko and Werner addressed the problem with k servers, where $k > 1$ in order to minimize the makespan [29]. They showed that, the problem is unary $\mathcal{NP}$-hard for each $k < m$ and developed a pseudo-polynomial time algorithm. Later in 2010, the same authors showed that, the problem with k servers with an objective function involving the minimization of the makespan is binary $\mathcal{NP}$-hard [36]. In addition, they conducted a worst case analysis of two list scheduling algorithms for makespan minimization. Furthermore, in the case of unloading operations involving multiple servers it is assumed that, a job starts its processing immediately on 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. [33] 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 two and proposed other heuristic algorithms as well as a branch-and-bound algorithm to solve the problem. The authors stated that, this problem was motivated by the milk run operations of a logistics company that faces limited unloading docks at the warehouse.

In this paper, we address a non-preemptive parallel machine scheduling problem with two dedicated servers by taking into account loading and unloading operations. The objective is to minimize the makespan. It is assumed that, the information about the problem is available before the scheduling starts (i.e., static case). Following the standard $\alpha|\beta|\gamma$ classification scheme for scheduling problems known as the Graham triplet [17], the considered problem can be denoted as $P, S2|s_j, t_j|C_{max}$, where P represents identical parallel machines, $S2$ represents two common servers, s_j is the loading time of job j, t_j is the unloading time of job j and C_{max} is the objective to be minimised (i.e., the makespan). We consider in this paper the regular case of the problem $P, S2|s_j, t_j|C_{max}$, where $\forall i, j \quad p_i < s_j + p_j + t_j$. In contrast, a set of jobs is called general if it is not respecting the regularity constraint (see [27] and [16]). It can be noticed that the regularity constraint was studied for the parallel machine scheduling problem involving a single server by [28] and [1].

In the scheduling literature, only a limited number of papers tackled the problem $P, S2|s_j, t_j|C_{max}$. Among them, Jiang et al. [26] addressed the problem $P2, S2|s_j = 1, t_j = 1|C_{max}$ with unit loading time, unit unloading time, and two identical parallel machines. The authors showed that, the classical list scheduling and largest processing time heuristics have worst-case ratios of 8/5 and 6/5, respectively. Recently, Benmansour and Sifaleras [8] proposed an MIP formulation and a general variable neighborhood search metaheuristic for the general case of the problem $P2, S2|s_j, t_j|C_{max}$ with only two identical parallel machines. To the best of our knowledge, no solution methods have been proposed in the literature for the regular case of problem $P, S2|s_j, t_j|C_{max}$ with an arbitrary number of machines. We fill this important gap in the literature by making the following contributions:

- we introduce a novel MIP formulation based on completion time variables for the regular case of the studied problem, and we improve it with a valid inequality.
- we propose four lower bounds and two greedy heuristics for the regular case of the problem.
- we design an efficient general variable neighborhood search metaheuristic with different initial solution-finding mechanisms for solving medium and large-sized instances of the regular case of the problem.
- we provide numerical results for reasonable computing times (with respect to the literature and to the industrial practice), including a comparison with the MIP formulation using a well-known commercial solver.

The rest of the paper is organized as follows: Section 2 provides a formal description, a mixed-integer-programming formulation, as well as a lower bound for the studied problem. Then, two greedy heuristics are presented in Sect. 3. In Sect. 4, a general variable neighborhood search approach and the neighborhood structures that are used, are described. Numerical experiments are performed in Sect. 5. Finally, concluding remarks are made in Sect. 6.

2 Problem formulation and lower bounds

In the problem $P, S2|s_j, t_j|C_{max}$ we consider that, a set $N = \{1, 2, \ldots, n\}$ of n independent jobs has to be processed on a set $M = \{1, 2, \ldots, m\}$ of m identical parallel machines with two common servers. The first server (loading server) is dedicated to the loading operation of jobs on the machines, while the second server (unloading server) is used to unload the jobs after their processing. Each job $j \in N$ is available at the beginning of the scheduling period and has a known integer processing time $p_j > 0$. Before its processing, each job j has to be loaded by the loading server, and the loading time is $s_j > 0$. After its processing, a job has to be unloaded from the machine by the unloading server, and the unloading time is $t_j > 0$. The processing operation starts immediately after the end of the loading operation, and the unloading operation starts immediately after the end of the processing operation. During the loading (respectively unloading) operation, both the machine and the loading server (respectively unloading server) are occupied and after loading (respectively unloading) a job, the loading server (respectively unloading server) becomes available for loading (respectively unloading) the next job. Furthermore, there is no precedence constraints among jobs, and preemption is not allowed. The objective is to find a feasible schedule that minimizes the makespan. In this paper, we consider the regular case of the problem $P, S2|s_j, t_j|C_{max}$, where $\forall i, j \in N \quad p_i < s_j + p_j + t_j$.

2.1 Mixed integer programming formulation

In this section, we present a refined version of the MIP formulation proposed in [8] which is based on Completion Time Variables (CSV) for the problem $P, S2|s_j, t_j|C_{max}$ with a regular job set. CSV formulation known also as natural-date variables formulation was initially used by Balas [5] to model a job shop scheduling problem. This formulation has been also used to model different $\mathcal{NP}$-hard scheduling problems (see [4, 7]).

The decision variables are defined as follows:

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

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

C_i be the completion time of job i.

Let L be a large positive integer, computed as $L = \sum_{i \in N}(s_i + p_i + t_i)$.

The problem $P, S2|s_j, t_j|C_{max}$ with a regular job set can be formulated as the following MIP. We recall that in [8], the authors considered the general case of the problem $P2, S2|s_j, t_j|C_{max}$.

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

$$s.t. \quad C_{max} \geq C_i \quad \forall i \in N \tag{2}$$

$$\sum_{k=1}^{m} x_{ik} = 1 \quad \forall i \in N \tag{3}$$

$$C_i \geq s_i + p_i + t_i \quad \forall i \in N \tag{4}$$

$$C_i + s_j + p_j + t_j \leq C_j + L(3 - x_{ik} - x_{jk} - z_{ij}) \quad \forall i, j \in N, i \neq j, \forall k \in M \tag{5}$$

$$C_i + s_j + p_j + t_j \leq C_j + p_i + t_i + L(1 - z_{ij}) \quad \forall i, j \in N, i \neq j \tag{6}$$

$$C_i + t_j \leq C_j + L(1 - z_{ij}) \quad \forall i, j \in N, i \neq j \tag{7}$$

$$z_{ij} + z_{ji} \geq 1 \quad \forall i, j \in N, i \neq j \tag{8}$$

$$x_{ik} \in \{0, 1\} \quad \forall i \in N, \forall k \in M \tag{9}$$

$$z_{ij} \in \{0, 1\} \quad \forall i, j \in N, i \neq j \tag{10}$$

In this formulation, the objective function (1) indicates that the makespan has to be minimized. Constraints (2) state that the makespan of an optimal schedule is greater than or equal to the completion time of all executed jobs. In order to guarantee that each job is scheduled on exactly one machine, constraint set (3) is added to the formulation. The completion time C_i which is at least greater than or equal to the sum of the loading, the processing, and the unloading times of the job i is calculated according to constraints (4). Constraints (5)–(8) show that a job can be processed only if the loading server, the machines, and the unloading server are available. Constraints (5) indicate that no two jobs scheduled on the same machine, can overlap in time. Constraints (6) state that the loading server can load at most one job at a time. While, the set of constraints (7) states that the unloading server can unload at most one job at a time. The constraints (8) impose that for each couple of jobs (i, j), one must be processed before the other. Finally, constraints (9) and (10) define binary variables x_{ik} and z_{ij}.

2.2 Strengthening the formulation

Proposition 1 *The following constraints*

$$C_{max} \geq \sum_{i=1}^{n} (s_i + p_i + t_i) x_{ik} \quad \forall k \in M \tag{11}$$

are valid for MIP.

Proof Assume the last job (denoted as job n) is executed on machine k.

The quantity $\sum_{i=1}^{n} (s_i + p_i + t_i)x_{ik}$ represents the total time of use of the machine k (since time 0), without counting the idle times. So it is obvious that $C_n = C_{max} \geq \sum_{i=1}^{n} (s_i + p_i + t_i)x_{ik}$. Hence inequalities (11) hold. □

Note that, we refer to Eqs. (1)−(10) as MIP1 and by considering the set of constraints Eq. (11) we refer to Eqs. (1)–(11) as MIP2.

2.3 Illustrative example

We now illustrate the previous formulation for an instance of $n = 5$ jobs and $m = 3$ machines. The processing time p_j, the loading time s_j and the unloading time t_j are given in Table 1. It takes 0.14 seconds to solve the instance using the above MIP1 formulation on IBM ILOG CPLEX 12.6. The optimal objective-function value is 20, and the obtained schedule of the problem is given in Fig. 1.

Table 1 Instance with $n = 5$ and $m = 3$

	Job 1	Job 2	Job 3	Job 4	Job 5
p_j	7	6	5	3	1
s_j	1	1	1	4	5
t_j	1	2	3	2	3

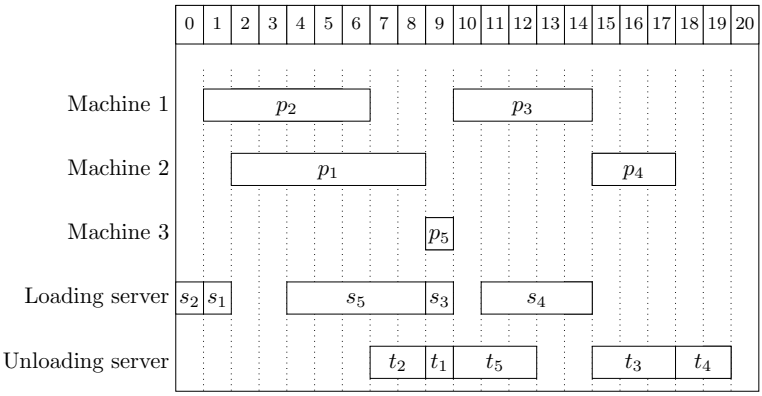

Fig. 1 Optimal schedule for the considered instance with 5 jobs and 3 machines

2.4 Lower bounds

In this section, we present a lower bound (LB) on the optimal makespan value for the problem $P, S2|s_j, t_j|C_{max}$. It can be useful in order to evaluate the quality of the greedy heuristics and the metaheuristic suggested in Sects. 3 and 6, respectively. $LB = \max(LB_1, LB_2, LB_3, LB_4)$, where LB_1, LB_2, LB_3, and LB_4 are given in Propositions 2, 3, 4, and 5 , respectively.

Proposition 2

$$LB_1 = \frac{1}{m}\sum_{i=1}^{n}(s_i + p_i + t_i)$$

is a valid lower bound for the problem $P, S2|s_j, t_j|C_{max}$.

Proof One can see that the total load to be executed by the m machines represents the sum of all loading, processing, and unloading times (i.e., $\sum_{i=1}^{n}(s_i + p_i + t_i)$). Then, it is sufficient to divide this total load by the number of machines to obtain the aforementioned lower bound. □

Proposition 3

$$LB_2 = \sum_{i=1}^{n} t_i + \min_{i \in N}(s_i + p_i)$$

is a valid lower bound for the problem $P, S2|s_j, t_j|C_{max}$.

Proof Let C^*_{max} denote the objective function value of an optimal solution of the problem $P, S2|s_j, t_j|C_{max}$. In addition, the unloading server waiting time of the job in position i, corresponds to the time between the end of the unloading operation of the job in position i and the start time of the unloading operation of the job in position $i + 1$. Indeed, if there is no unloading server waiting time in an optimal schedule of the problem $P, S2|s_j, t_j|C_{max}$, then C^*_{max} will be equal to the sum of all unloading times plus the shortest sum of the loading and processing times, which corresponds to $\min_{i \in N}(s_i + p_i)$. Hence, LB_2 is valid. □

Proposition 4

$$LB_3 = \sum_{i=1}^{n} s_i + \min_{i \in N}(p_i + t_i)$$

is a valid lower bound for the problem $P, S2|s_j, t_j|C_{max}$.

Proof Let start by defining the Loading Server Waiting Time (LSWT). LSWT of the job in position i, corresponds to the time between the end of the loading operation of the job in position i and the start time of the loading operation of the job in position $i + 1$. Indeed, if there is no loading server waiting time in an optimal schedule of the problem $P, S2|s_j, t_j|C_{max}$, then C^*_{max} will be equal to the sum of all loading times plus the shortest sum of the processing and unloading times, which corresponds to $\min_{i \in N}(p_i + t_i)$. Hence, the aforementioned lower bound (LB_3) is valid. □

Proposition 5

$$LB_4 = \max_{i \in N}(s_i + p_i + t_i)$$

is a valid lower bound for the problem $P, S2|s_j, t_j|C_{max}$.

Proof One can see that the completion time of the job with the maximal duration of the sum of loading, processing and unloading times, represents a valid lower bound for the problem $P, S2|s_j, t_j|C_{max}$. □

3 Greedy heuristics

In this section, we present two greedy heuristics that aim to minimize the loading/unloading server waiting time for the problem $P, S2|s_j, t_j|C_{max}$ with a regular job set. The basic objective at each step of the proposed heuristic is to avoid the generation of server loading/unloading waiting times. The idea of the proposed heuristics relies on the works of [2, 22], and [14] for the problems $P2, S1|s_j, p_j|C_{max}$ and $P, S1|s_j, p_j|C_{max}$, involving only one single server. Indeed, Abdekhodaee et al. [2] proposed a forward and a backward heuristics to minimize the server waiting time and the machine idle time, respectively for the problem $P2, S1|s_j, p_j|C_{max}$. Also, Hasani et al. [22] suggested two other heuristics for the same problem. Later, El Idrissi et al. [14] generalized the precedent suggested heuristics for the problem $P, S1|s_j, p_j|C_{max}$, with an arbitrary number of machines and a single server. It can be noticed that, the unloading server has not been considered in the precedent works.

3.1 Greedy heuristic (USWT)

In this heuristic, jobs are scheduled according to the availability of machines, the loading and the unloading servers. In addition, the job with the shortest sum of the loading and processing times is considered as the first job to be scheduled in the final sequence. It follows the structure of the lower bound LB_2 presented in Proposition 3. The steps of the first proposed heuristic, called USWT (Unloading Server Waiting Time), are as follows.

Heuristic USWT

- Step 1: Sort the list of jobs $\pi = \{\pi_1, \ldots, \pi_k, \ldots, \pi_n\}$ in increasing order of the sum of loading and processing times.
- Step 2: Sequence the first job of the list π on the earliest available machine.
- Step 3: Set Γ to the difference between the end of the loading and the unloading time of last sequenced job.
- Step 4: From the unsequenced jobs, find a job with a sum of the loading and processing time less than or equal to Γ. If there is no such job, select the first one from the list π. Schedule the selected job to the first available machine at the earliest possible time taking in consideration the loading/unloading server constraints.
- Step 5: Repeat Steps 3 and 4 until all jobs are sequenced.

3.2 Greedy heuristic (LSWT)

Contrary to the heuristic USWT, in this section, we present a constructive heuristic that aims to minimize the loading server waiting time. In this heuristic, the job with the shortest sum of the processing and unloading time is considered as the last job to be scheduled in the final list sequence. It follows the structure of the lower bound (LB_3) presented in Proposition 4. The steps of the second proposed heuristic, called LSWT (Loading Server Waiting Time), are as follows.

Heuristic LSWT

- Step 1: Sort the list of jobs π in increasing order of the sum of processing and unloading times.
- Step 2: Sequence the first job of the list π in the final position, and sequence the second job of the list π in the first position on the earliest available machine.
- Step 3: Set Θ to the difference between the end of the loading and the unloading time of last sequenced job.
- Step 4: From the unsequenced jobs, find a job with a sum of the loading and processing time greater than or equal to Θ. If there is no such job, select the first one from the list π. Schedule the selected job to the first available machine at the earliest possible time taking in consideration the loading/unloading server constraints.
- Step 5: Repeat Steps 3 and 4 until all jobs are sequenced.

Note that USWT and LSWT are both complementary, and are used to generate initial solutions for the metaheuristic presented in the next Sect. 4.

4 General variable neighborhood search (GVNS) metaheuristic

Variable Neighborhood Search (VNS) is a single solution metaheuristic method proposed by Mladenović and Hansen [31] that uses local search procedure as its basic building block. Basic variant of VNS (BVNS) involves three main steps: Shaking, Local Search (LS), and Change Neighborhood (Move or Not). Shaking step represents the diversification (perturbation) phase whose role is to ensure escaping from local optima traps. It is always applied to the current best solution and consists of random perturbation in the given neighborhood. The obtained (shaken, perturbed) solution represents a starting point to the LS step. The role of LS is to improve shaken solution by examining its neighbors (in one or more neighborhoods). When a local optimum is obtained by LS, BVNS performs the final step (Move or Not). Within this step local optimum is compared with the current best solution. If an improvement is obtained, the search concentrates around the newly found best solution which means that the current best solution and the neighborhood index are properly updated. In the case that current best solution was not improved, the search is expanded to wider part of solution space (if possible) by increasing the

neighborhood index for Shaking. The main BVNS steps are repeated until a pre-specified stopping criterion is satisfied [19].

Since 1997, VNS has been widely used and many variants and successful applications can be found in the relevant literature [20, 32]. The simplest VNS variant is Reduced VNS (RVNS), consisting only of Shaking and Move or Not Steps. 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) [13]. It is a deterministic algorithm where LS is performed along multiple neighborhoods. Skewed VNS (SVNS) represents a variant of VNS that allows the acceptance of non-improving solutions in Move or Not step with a given probability [10]. In SVNS an additional parameter α is introduced to control the quality of these non-improving solutions, while General VNS (GVNS), contains all three main steps and explores VND instead of LS [19].

Here, the GVNS metaheuristic is implemented for the problem $P, S2|s_j, t_j|C_{max}$ with a regular job set. The details about this implementation are provided in the remainder of this section.

4.1 Implementation details

A solution of the considered scheduling problem $P, S2|s_j, t_j|C_{max}$ 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 k^{th} position. This representation is in fact indirect as it requires actual scheduling of jobs to machines taking into consideration the servers constrains (i.e., loading and unloading) in order to calculate value of the objective function (C_{max}). Having in mind that jobs are independent, all permutations ($n!$) are representing feasible solutions, and therefore, the search space is very large. On the other hand, several neighborhood structures could be defined for this representation.

4.1.1 Neighborhood structures

To obtain an efficient VNS metaheuristic we have to decide about the neighborhood structures to use. The following three neighborhood structures ($l_{max} = 3$) are proposed to explore the solution space for the problem at hand.

- $\mathcal{N}_1(\pi) = \text{Swap}(\pi)$. It consists of all solutions obtained from the solution π swapping two jobs of π.
- $\mathcal{N}_2(\pi) = \text{Insert}(\pi)$: It consists of all solutions obtained from the solution π by reinserting one of its job somewhere else in the sequence.
- $\mathcal{N}_3(\pi) = Reverse(\pi)$: It consists of all solutions obtained from the solution π reversing a sub-sequence 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}.

Note that, these neighborhood structures have been successfully used in different scheduling problems involving a single server (see [15, 21]).

4.1.2 Variable neighborhood descent

Now, we propose to use $\mathcal{N}_1$, $\mathcal{N}_2$ and $\mathcal{N}_3$ within VND (Algorithm 1). It starts with an initial solution π_0 and continuously tries to construct a new improved solution from the current solution π by exploring its neighborhood $\mathcal{N}_l(\pi)$. The search continues to generate neighboring solutions until no further improvement can be made. Furthermore, the performance of VND depends on the order in which neighborhoods are explored, and on how to switch from one neighborhood to another during the search. To switch from one neighborhood to another (Change neighborhood step), we propose to use basic sequential, pipe, and cyclic strategies. They are given in Algorithms 2–4, respectively. Exhaustive testing is performed in Sect. 5.3 in order to identify the best order of neighborhoods with respect to the suggested change neighborhood strategies.

Algorithm 1: Variable Neighborhood Descent

```
   Data: π, l_max
   Result: π
1  repeat
2  |  l ← 1
3  |  repeat
4  |  |  π' ← Local Search(π, N_l)
5  |  |  Change Neighborhood-type(π, π', l)
6  |  until l = l_max
7  until there is no improvement
8  return π
```

Algorithm 2: Change Neighborhood-Sequential

```
1 if C_max(π') < C_max(π) then
2 |  π ← π';
3 |  l ← 1;
4 else
5 |  l ← l + 1
6 end
```

Algorithm 3: Change Neighborhood-Pipe

```
if C_max(π') < C_max(π) then
    π ← π';
else
    l ← l + 1
end
```

Algorithm 4: Change Neighborhood-Cyclic

```
l ← l + 1;
if C_max(π') < C_max(π) then
    π ← π';
end
```

4.1.3 Shaking

For escaping local optimum solutions, a shaking procedure is proposed. It consists of sequentially generating k random jumps from the current solution π using the neighborhood structure $\mathcal{N}_3$. After preliminary experiments, the shaking method with more neighborhood structures reduced the quality of the results. Its pseudo-code is given in Algorithm 5.

Algorithm 5: Shaking

```
Data: π, k
Result: π
for j = 1 to k do
    Select π' ∈ N_3(π) at random
    π ← π'
end
return π
```

4.2 GVNS for the problem $P, S2|s_j, t_j|C_{max}$

In this section, we present the overall pseudocode of GVNS as it is implemented to solve the the problem $P, S2|s_j, t_j|C_{max}$ with a regular job set (see Algorithm 6). The diversification and intensification ability of GVNS relies on the shaking phase and VND, respectively. Shaking step of GVNS consists of one neighborhood structure $\mathcal{N}_3$. In the VND step, the three proposed neighborhood structures are used. The stopping criterion is a CPU time limit T_{max}. Since GVNS is a

trajectory-based metaheuristic, we need to start from a given solution. Therefore, we refer to GVNS I as GVNS using USWT heuristic as initial solution, GVNS II as GVNS using LSWT heuristic as initial solution, and GVNS III as GVNS using a random initial solution. Note that all GVNS variants are compared in the next Sect. 5.

Algorithm 6: General Variable Neighborhood Search

```
   Data: π, k_max, T_max
   Result: π
 1 π ← Initial Solution()
 2 repeat
 3 |   k ← 1
 4 |   while k ≤ k_max do
 5 |   |   π' ← Shaking(π, k)
 6 |   |   π'' ← VND(π')
 7 |   |   if C_max(π'') < C_max(π) then
 8 |   |   |   π ← π''
 9 |   |   |   k ← 1
10 |   |   else
11 |   |   |   k ← k + 1
12 |   |   end
13 |   end
14 until CPU > T_max
15 return π
```

5 Computational results

In this section, the computational experiments carried out to evaluate the performance of the MIP1 (1)-(10), MIP2 (1)-(11), USWT, LSWT, GVNS I, GVNS II, and GVNS III for the problem $P, S2|s_j, t_j|C_{max}$ with a regular job set, are presented. The MIP1 and MIP2 were solved using concert Technology library of CPLEX 12.6 version with default settings in C++, whereas LSWT, USWT, GVNS I, GVNS II and GVNS III were coded in the C++ language. 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-sized instances for which one run is sufficient, the metaheuristics were executed 10 times in all experiments reported in this section.

5.1 Benchmark instances

To the best of our knowledge, there are no publicly available benchmark instances from the literature regarding the problem $P, S2|s_j, t_j|C_{max}$ with a regular job set, so we decided to generate a new set of instances. This set was created by generalizing the scheme previously proposed by Silva et al. [34] and Benmansour and Sifaleras [8]. Indeed, to generate a regular job set, first we generate a general job set, where the processing time p_j, loading time s_j, and unloading time t_j of each job j were generated from the uniform distributions $U[10, 100]$, $U[5, 25]$ and $U[5, 25]$,

respectively. Then, we reduce this general job set into a regular one by adapting the Koulama's reduction algorithm (see Koulamas [28]) to our studied problem. In our generation scheme, we adopted the following values: $n \in \{10, 50, 100, 250\}$, and $m \in \{2, 3, 5\}$. Thus, leading to a total of 12 groups of instances. For each group of instances (n, m), ten instances were created, resulting in a total of 120 new instances. These instances are publicly available at: https://sites.google.com/view/data-set-for-thepmsssproblem/accueil.

5.2 Parameters setting

For the proposed GVNS metaheuristic, two parameters have to be tuned, k_{max} which represents the maximum level of perturbation and T_{max} which corresponds to the maximum time allowed to be used by the GVNS. After some preliminary tests, we decided to set k_{max} to 20 as it offered a reasonable trade-off between the quality of the solution and CPU time. For small-sized instances ($n = 10$ and $m \in \{2, 3, 5\}$), T_{max} is set to the computing time to find an optimal solution by CPLEX solver. For medium-sized instances ($n \in \{50, 100\}$ and $m \in \{2, 3, 5\}$), T_{max} is set to 100 seconds. Finally, for large-sized instances ($n = 250$ and $m \in \{2, 3, 5\}$), T_{max} is set to 200 seconds. In addition, for all instances, the time limit for CPLEX is set to 1h.

5.3 Comparison of VND variants

Now, we present a detailed comparison of three VND variants, namely: sequential VND, pipe VND and cyclic VND. These VND variants have been widely used for different optimization problems (see [12, 30]). The performance of the proposed VND variants depends on the sequence of the three neighborhood structures $(\mathcal{N}_1, \mathcal{N}_2, \mathcal{N}_3)$, and also on the search strategy (*first* or *best* improvement). Therefore, six different sequences of neighborhood structures are presented in Table 2. Note that, in this comparison each VND variant starts with a random permutation of the solution π.

Furthermore, 30 instances of size $n = 250$ with m ranging from 2 to 5 (large-sized instances), are used in this comparison. Table 3 presents the average results for the *first* improvement strategy (i.e., in each iteration, stop the generation of neighbor solutions as soon as the current solution can be improved), whereas Table 4 presents the average results for the *best* improvement strategy (i.e., in each iteration, generate all the neighbor solutions and pick up the best one). For each experiment, we indicate the average value of C_{max} and the average computing time (in seconds).

Table 2 Possible sequences for the neighborhood structures

Sequence 1	Sequence 2	Sequence 3	Sequence 4	Sequence 5	Sequence 6
Insert	Insert	Swap	Swap	Reverse	Reverse
Swap	Reverse	Insert	Reverse	Insert	Swap
Reverse	Swap	Reverse	Insert	Swap	Insert

Table 3 Comparison of VNDs with different sequences with the *first* improvement strategy

	Sequence 1		Sequence 2		Sequence 3		Sequence 4		Sequence 5		Sequence 6	
	C_{max}	CPU	C_{max}	CPU	C_{max}	CPU	C_{max}	CPU	C_{max}	CPU	C_{max}	CPU
Sequential VND	6470.93	11.60	6463.77	11.49	6335.73	11.31	6334.63	10.69	6487.8	9.98	6446.7	10.27
Pipe VND	6460.03	8.65	6455.57	9.71	6333.77	12.50	6360.07	13.29	6485.93	9.64	6442.93	11.64
Cyclic VND	6464.57	6.60	6476.93	6.84	6347.9	8.57	6326.83	17.73	6418.67	11.19	6377.1	8.99

Table 4 Comparison of VNDs with different sequences with the *best* improvement strategy

	Sequence 1		Sequence 2		Sequence 3		Sequence 4		Sequence 5		Sequence 6	
	C_{max}	CPU	C_{max}	CPU	C_{max}	CPU	C_{max}	CPU	C_{max}	CPU	C_{max}	CPU
Sequential VND	6588.77	14.67	6589.7	17.76	6452.57	35.37	6400.2	41.11	6446.03	22.41	6441.63	21.96
Pipe VND	6564.3	18.52	6557.4	18.97	6456.3	43.80	6404.53	40.72	6431.27	14.57	6425.63	15.26
Cyclic VND	6571.8	23.58	6574.73	23.97	6564.37	16.52	6447.6	18.85	6439.37	20.72	6443.37	13.12

According to these results, we can observe that VND is sensitive to the neighborhood-structure sequence and the searching strategy. It can be noticed that the neighborhood-structure Cyclic, *Sequence 4*, combined with the first improvement strategy leads to the best results, since it returns the minimum value of the average makespan. Therefore, we will use these parameters within the proposed GVNS.

5.4 Results

Because of the $\mathcal{NP}$-hard nature of the problem $P, S2|s_j, t_j|C_{max}$, an optimal solution was only obtained for small-sized instances with up to 10 jobs and 5 machines. Therefore, LSWT, USWT, GVNS I, GVNS II and GVNS III were designed to solve medium and large-sized instances. Having in mind that in the proposed GVNS I, the initial solution is obtained by the greedy heuristic USWT. In GVNS II, the initial solution is obtained by the greedy heuristic LSWT. Finally, in GVNS III the initial solution is randomly generated.

5.4.1 Exact approaches

In Table 5, we compare the performance of MIP1 and MIP2 for all groups of instances (12 group of instances) for a time limit of 1 h. Each group of instances is characterized by the following information: the group number; a number n of jobs; a number m of machines. In addition, for each MIP formulation, the following information is given: *i*) the average time required to prove optimality, CPU, *ii*) the average percentage gap to optimality, $gap_{LB}(\%)$, *iii*) the average gap between the formulation's lower bound and the best upper bound, $gap_{LB-UB}(\%)$, and *iv*) the average gap between the linear programming (LP) relaxation lower bound and the best upper bound, $gap_{LP}(\%)$. The following observations can be made:

- For $n = 10$ and $m \in \{2, 3, 5\}$: Based on formulations MIP1 and MIP2, CPLEX is able to produce an optimal solution for any instance. It can be noticed that for the improved formulation MIP2, CPLEX is able to produce an optimal solution in significantly less computational time in comparison with the original formulation, except for group 2 and group 3.
- For $n \in \{50, 100, 250\}$ and $m \in \{2, 3, 5\}$: Based on MIP1 and MIP2, CPLEX is able to find a feasible solution for all groups. It can be noticed that the improved formulation MIP2 produce much smaller $gap_{LB}(\%)$ and $gap_{LB-UB}(\%)$ in comparison with MIP1. In addition, MIP2 reduced significantly the value of the linear programming relaxation.

The overall results showed that MIP2 outperforms, on average, the MIP1 formulation on almost all instances. Furthermore, the impact of the strengthening constraints in Eq. 11 is very positive as MIP2 produce more strict LP relaxation bounds than MIP1. In the following of the paper, we compare only MIP2 with

Table 5 Average results found by the different mathematical formulations

Group	n	m	MIP1				MIP2			
			CPU	$gap_{LB}(\%)$	$gap_{LB-UB}(\%)$	$gap_{LP}(\%)$	CPU	$gap_{LB}(\%)$	$gap_{LB-UB}(\%)$	$gap_{LP}(\%)$
1	10	2	689.77	0.00	0.00	0.00	267.35	0.00	0.00	0.00
2		3	30.64	0.00	0.00	0.00	48.54	0.00	0.00	0.00
3		5	6.47	0.00	0.00	0.00	50.48	0.00	0.00	0.00
4	50	2	3600	91.12	91.09	94.65	3600	1.01	0.97	0.97
5		3	3600	87.96	87.54	92.54	3600	9.41	9.30	9.30
6		5	3600	87.93	87.96	92.03	3600	42.71	42.16	42.16
7	100	2	3600	96.14	96.12	97.24	3600	4.18	3.19	3.19
8		3	3600	95.22	95.14	96.48	3600	15.66	15.18	15.18
9		5	3600	94.35	94.20	95.99	3600	42.54	42.54	42.54
10	250	2	3600	99.00	98.78	99.04	3600	16.14	16.14	22.00
11		3	3600	98.96	98.78	99.00	3600	41.06	41.06	45.65
12		5	3600	99.28	98.48	98.78	3600	51.44	56.86	56.86

the other approaches, as it produce the best results. Note that MIP2 was not able to prove optimality of 50-job instance, which is a limited size. Consequently, there is a need for meta/heuristics able to find an approximate solution if possible of high quality in a short computational time.

5.4.2 Approximate approaches

In Table 6, we compare the performance of the all proposed methods for small-sized instances ($n = 10$ and $m \in \{2, 3, 5\}$), where an optimal solution can be found by MIP2 within one hour. Each instance is characterized by the following information. First, an ID (the name of each instance, e.g., *nXmYkZ* denotes the Z^{th} instance with $n = X$ and $m = Y$); the lower bound LB computed as in Sect. 2.4; the optimal value C^*_{max} of C_{max} (found with the MIP2). Second, the obtained value of C_{max} is given for USWT and LSWT (not the computing time, as it is always below 0.0001). Finally, the computing time to find an optimal solution is given for the MIP2, GVNS I, GVNS II, and GVNS III. The last line of the table indicates average results. The following observations can be made :

- GVNS I, GVNS II, and GVNS III can reach an optimal solution for each instance in significantly less computing time than the MIP2.
- USWT and LSWT are not able to generate an optimal solution for all instances. On average, USWT produces solution of better quality than LSWT.
- The theoretical lower bound *LB* is on average 6.17% bellow C^*_{max}.
- GVNS I requires the smallest average computing time (0.70 second) to find optimal solutions in comparison with MIP2, GVNS II, and GVNS III.

In addition, Table 7 presents the performance of the three metaheuristic approaches in terms of the percentage deviation from the best-known solutions for each group of instances (the best one over all the runs of all the metaheuristics, and the one obtained by the considered metaheuristic). For each metaheuristic, the following information is given: the minimum value of the percentage deviation over all instance's group, Min, the average value of the percentage deviation over all instance's group, Avg, and the maximum value of the percentage deviation over all instance's group, Max. The last line of the table indicates average results. The results show that GVNS I, on average, obtained a superior performance in terms of minimum, average and maximum gaps for each group of instance, when compared to GVNS II and GVNS III.

Furthermore, the detailed results of the MIP2, the two greedy heuristics, and the three metaheuristics for the remaining instances is given in Appendix 1. Tables 8, 9, 10 present the performance of the all approaches for medium and large-sized instances with $n \in \{50, 100, 250\}$ and $m \in \{2, 3, 5\}$, where only a feasible solution can be found by MIP2 within one hour (the best results are indicated in bold). The instance characteristics are first indicated. First for the MIP2, the following information is given: the lower bound LB_{MIP2}, the upper bound UB_{MIP2}, the

Table 6 Detailed results found by the proposed approaches for instances with $n = 10$ and $m \in \{2, 3, 5\}$

Instance			USWT	LSWT	MIP2	GVNS I	GVNS II	GVNS III
ID	*LB*	C^*_{max}	C_{max}	C_{max}	CPU	CPU	CPU	CPU
10*n*2*m*1	280	291	312	314	234.99	0.01	0.01	0.00
10*n*2*m*2	333.5	341	377	369	83.90	0.02	0.01	0.00
10*n*2*m*3	348.5	364	381	381	480.23	0.00	0.00	0.00
10*n*2*m*4	293	306	322	349	239.04	0.01	0.07	0.01
10*n*2*m*5	299	310	325	333	421.74	0.00	0.00	0.00
10*n*2*m*6	380	391	410	416	314.58	0.12	0.19	0.19
10*n*2*m*7	265.50	274	286	305	73.97	0.00	0.00	0.00
10*n*2*m*8	323	334	364	365	691.69	0.11	0.07	0.00
10*n*2*m*9	358	367	387	406	68.83	2.77	2.45	2.11
10*n*2*m*10	302	310	339	349	64.48	0.01	0.00	0.00
10*n*3*m*1	235	248	270	311	92.72	0.90	0.39	1.48
10*n*3*m*2	216	246	272	300	21.75	4.12	7.52	7.76
10*n*3*m*3	228	251	274	300	13.28	1.73	0.58	2.71
10*n*3*m*24	209.67	236	270	293	13.63	0.66	0.33	0.19
10*n*3*m*25	240.33	283	304	310	61.89	0.07	0.02	0.17
10*n*3*m*6	249	271	303	294	75.67	0.01	0.01	0.04
10*n*3*m*7	247	275	288	326	41.69	0.07	0.01	0.01
10*n*3*m*8	219	239	282	264	38.40	0.18	0.12	0.02
10*n*3*m*9	219	253	273	299	89.87	1.36	0.54	0.44
10*n*3*m*10	200	226	272	262	36.52	0.06	0.01	0.02
10*n*5*m*1	242	251	293	322	53.28	0.01	0.01	0.01
10*n*5*m*2	232	235	245	298	77.50	0.00	0.01	0.00
10*n*5*m*3	214	230	271	266	18.23	7.27	9.16	7.42
10*n*5*m*4	214	238	263	304	57.85	0.13	0.02	0.06
10*n*5*m*5	223	231	270	255	25.84	0.01	0.11	0.02
10*n*5*m*6	211	225	254	300	22.63	0.19	1.21	0.53
10*n*5*m*7	205	214	251	273	66.79	0.07	0.20	0.01
10*n*5*m*8	192	216	264	237	32.52	0.01	0.01	0.01
10*n*5*m*9	235	243	286	321	30.69	0.40	0.92	0.10
10*n*5*m*10	212	228	262	286	119.46	0.64	0.40	0.52
Avg.	254.18	270.9	299	313.6	122.12	0.70	0.81	0.79

percentage gap to optimality $Gap_{LB}(\%)$, and the time requested to prove optimality (CPU). Second, the obtained value of C_{max} is given for USWT and LSWT (not the computing time, as it is always below 0.001). Finally, for GVNS I, GVNS II and GVNS III : the best (respectively average) objective-function value over 10 runs denoted as Best (respectively Avg). In addition, the average computing times are

Table 7 Gap from the best solution found

Group	n	m	Gap(%)								
			GVNS I			GVNS II			GVNS III		
			Min.	Avg.	Max.	Min.	Avg.	Max.	Min.	Avg.	Max.
1	10	2	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
2		3	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
3		5	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
4	50	2	0.00	0.05	0.20	0.00	0.05	0.13	0.00	0.02	0.07
5		3	0.00	0.28	0.79	0.00	0.20	0.72	0.00	0.15	0.53
6		5	0.00	0.64	1.29	0.00	0.63	1.87	0.00	0.09	0.88
7	100	2	0.00	0.03	0.10	0.00	0.05	0.14	0.00	0.05	0.20
8		3	0.00	0.17	0.89	0.00	0.46	0.98	0.00	0.22	0.62
9		5	0.00	0.66	1.64	0.00	0.37	1.68	0.00	0.62	2.16
10	250	2	0.00	0.00	0.00	0.00	0.10	0.10	0.00	0.07	0.07
11		3	0.00	0.05	0.05	0.00	0.50	0.50	0.05	0.57	0.57
12		5	0.00	0.34	0.34	0.00	0.93	0.93	0.00	0.75	0.75
Avg.			0.00	0.19	0.44	0.00	0.27	0.59	0.00	0.21	0.49

also presented (computed over the 10 runs). Note that the computing time of a run corresponds to the time at which the best visited solution is found. According to these results, overall, out of 120 instances, GVNS I found 73 best solution (60.83%), whereas GVNS II and GVNS III found only 42 (35%) and 71 (59.16%), respectively. Figure 2 depicts the tradeoff between the quality of solution versus the time expended, for 10 instances of size $n = 250$ and $m = 5$ for GVNS I. We have chosen

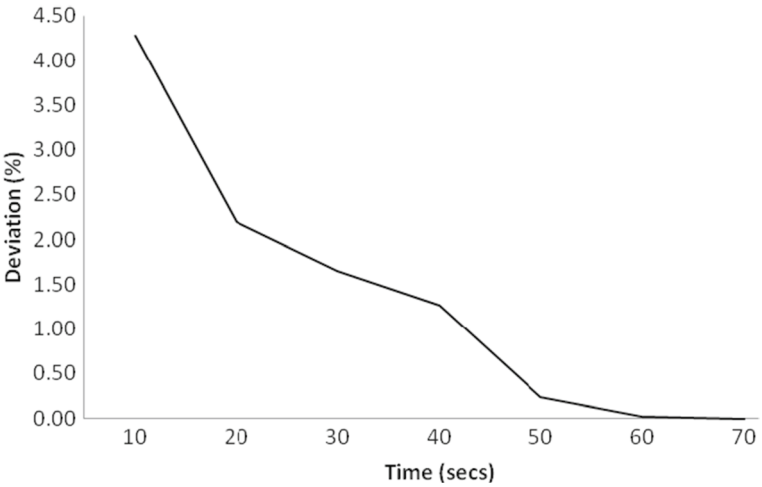

Fig. 2 Average quality percent deviation from best found solutions versus computational effort for GVNS I

Table 8 Detailed results found by the proposed approaches for instances with $n = 50$ and $m \in \{2, 3, 5\}$

Instance		MIP2				USWT	LSWT	GVNS I			GVNS II			GVNS III		
ID	LB	LB_{MIP2}	UB_{MIP2}	$Gap_{LB}(\%)$	CPU	C_{max}	C_{max}	Best	Avg	CPU	Best	Avg	CPU	Best	Avg	CPU
50*n*2*m*1	1516.5	1516.5	1529	0.82	3600	1569	1620	1526	1527.1	35.80	1526	1527.4	36.14	**1525**	1526.1	50.95
50*n*2*m*2	1649.5	1649.5	1666	0.99	3600	1696	1752	**1656**	1656.6	46.75	1656	1656.8	25.37	**1656**	1656.5	43.38
50*n*2*m*3	1514.5	1514.5	1534	1.27	3600	1578	1623	**1522**	1522.6	24.22	1521	1523	44.91	**1522**	1523.1	56.47
50*n*2*m*4	1490	1490	1500	0.67	3600	1527	1565	**1496**	1497.9	36.68	1497	1497.6	43.03	**1496**	1497.9	28.32
50*n*2*m*5	1514	1514	1528	0.92	3600	1560	1600	1524	1525.1	37.65	1523	1524.9	31.55	**1523**	1524.3	35.65
50*n*2*m*6	1514	1514	1532	1.17	3600	1567	1613	1523	1523.2	34.07	1523	1523.8	21.51	**1522**	1524	30.12
50*n*2*m*7	1631.5	1631.5	1642	0.64	3600	1671	1723	**1638**	1639.4	27.46	1639	1639.6	41.19	1639	1639.5	32.30
50*n*2*m*8	1495	1495	1518	1.52	3600	1568	1632	1503	1503.8	38.70	1502	1502.6	36.38	**1500**	1502.4	42.92
50*n*2*m*9	1612.5	1612.5	1630	1.07	3600	1663	1740	1620	1620.9	29.28	1620	1620.3	35.83	**1619**	1620.5	40.86
50*n*2*m*10	1564	1564	1581	1.08	3600	1620	1741	**1571**	1573	36.74	1571	1572.3	49.16	1572	1573	28.87
50*n*3*m*1	1008.67	1008.67	1115	9.54	3600	1159	1268	1089	1092.1	43.29	1083	1091.2	31.82	**1086**	1092.2	29.61
50*n*3*m*2	1075.33	1075.33	1156	6.98	3600	1218	1282	**1134**	1139.6	51.46	1134	1138.6	52.60	1135	1138.5	31.64
50*n*3*m*3	1003	1003	1142	12.17	3600	1182	1277	1107	1114.6	43.97	1111	1115.6	47.37	**1103**	1111.5	5.38
50*n*3*m*4	1049.33	1049.33	1141	8.03	3600	1192	1288	1122	1126.7	42.65	1119	1124.5	35.81	**1119**	1125.6	37.42
50*n*3*m*5	1118.33	1118.33	1187	5.78	3600	1235	1323	**1169**	1174.4	33.97	1169	1173.9	49.13	1170	1175.4	36.91
50*n*3*m*6	1065	1065	1206	11.69	3600	1230	1316	1140	1146.7	38.26	1140	1144.8	58.30	**1138**	1144	33.82
50*n*3*m*7	1065.33	1065.33	1189	10.40	3600	1211	1295	1147	1150.8	46.38	1147	1150.3	40.58	**1141**	1150.3	21.03
50*n*3*m*8	1060.33	1060.33	1167	9.14	3600	1209	1289	**1126**	1134.1	58.50	1131	1133.9	51.08	1132	1136.1	22.96
50*n*3*m*9	1046.33	1046.33	1194	12.37	3600	1211	1304	1144	1147	18.65	1135	1144.7	43.20	**1141**	1146.7	27.31
50*n*3*m*10	1008.33	1008.33	1096	8.00	3600	1134	1233	1067	1073.4	30.68	1067	1071.6	50.53	**1066**	1070.6	33.86
50*n*5*m*1	975	631	1049	39.85	3600	1101	1146	**1020**	1025.5	48.97	1014	1025.5	46.09	1023	1028	28.29
50*n*5*m*2	941	549	1021	46.23	3600	1101	1223	999	1010.5	32.02	999	1008.6	51.17	**990**	1005	52.69
50*n*5*m*3	1009	635.4	1087	41.55	3600	1142	1230	1059	1063.9	42.02	1057	1063.3	50.61	**1046**	1058.2	30.49
50*n*5*m*4	1026	653	1101	40.69	3600	1148	1301	1082	1091.1	53.06	1083	1088.4	46.32	**1080**	1087.9	39.89

Table 8 (continued)

Instance		MIP2				USWT	LSWT	GVNS I			GVNS II			GVNS III		
ID	LB	LB_{MIP2}	UB_{MIP2}	$Gap_{LB}(\%)$	CPU	C_{max}	C_{max}	Best	Avg	CPU	Best	Avg	CPU	Best	Avg	CPU
50*n*5*m*5	972	625	1082	42.24	3600	1142	1297	1033	1044.2	40.85	1038	1046.4	38.28	**1032**	1043.8	28.90
50*n*5*m*6	1058	618.2	1090	43.28	3600	1146	1311	1088	1094	47.88	1085	1092.4	37.10	**1083**	1092.7	50.54
50*n*5*m*7	946	626.6	1030	39.17	3600	1088	1231	1001	1011.2	31.41	996	1006.7	53.59	**991**	1004.1	43.17
50*n*5*m*8	943	593.6	1127	47.33	3600	1107	1263	1008	1015.6	40.20	1014	1018.9	39.38	**995**	1013.7	31.23
50*n*5*m*9	952	575.6	1038	44.55	3600	1114	1233	1051	1055.2	50.51	1045	1056.6	50.47	**1044**	1053.2	30.83
50*n*5*m*10	922	591.4	1023	42.19	3600	1079	1280	**997**	1009.6	41.98	1006	1011.5	63.36	997	1005.1	12.61
Avg.	1191.52	1070.01	1263.37	17.71	3600	1305.60	1399.97	1238.73	1243.7	39.47	1238.37	1243.19	43.40	1236.20	1242.33	33.95

Table 9 Detailed results found by the proposed approaches for instances with $n = 100$ and $m \in \{2, 3, 5\}$

Instance		MIP2				USWT	LSWT	GVNS I			GVNS II			GVNS III		
ID	LB	LB_{MIP2}	UB_{MIP2}	$Gap_{LB}(\%)$	CPU	C_{max}	C_{max}	Best	Avg	CPU	Best	Avg	CPU	Best	Avg	CPU
100*n*2*m*1	2914.5	2914.5	3132	6.94	3600	3016	3055	2931	2934.3	37.17	**2929**	2934.2	34.6302	2931	2933.9	41.28
100*n*2*m*2	3185.5	3185.5	3283	2.97	3600	3266	3278	3196	3198.4	17.86	**3195**	3198.7	42.1734	3196	3198.2	45.98
100*n*2*m*3	3124.5	3124.5	3284	4.86	3600	3199	3236	3134	3137.4	22.22	3134	3136.8	37.2746	**3133**	3136.5	31.20
100*n*2*m*4	3011	3011	3148	4.35	3600	3110	3083	3027	3029.2	17.14	**3024**	3027.9	41.7487	3027	3028.9	40.81
100*n*2*m*5	2912.5	2912.5	3027	3.78	3600	3036	3057	**2925**	2930.8	33.59	2929	2933.5	39.1949	2931	2932.6	33.68
100*n*2*m*6	2968.5	2968.5	3117	4.76	3600	3043	3139	**2984**	2986.6	38.02	2985	2988.4	40.3402	**2984**	2986.7	51.93
100*n*2*m*7	3183.5	3183.5	3261	2.38	3600	3249	3293	**3194**	3197.1	34.98	3196	3197.5	35.2199	3196	3197.6	27.26
100*n*2*m*8	3053	3053	3171	3.72	3600	3119	3155	3067	3068.1	43.22	3067	3068.6	33.9656	**3066**	3068.1	51.39
100*n*2*m*9	2972.5	2972.5	3100	4.11	3600	3022	3089	**2986**	2990.6	33.22	2989	2990.8	42.5326	**2986**	2989.9	32.39
100*n*2*m*10	3148.5	3148.5	3278	3.95	3600	3236	3268	**3160**	3163.3	44.18	3162	3163.7	36.7731	3162	3163.3	39.48
100*n*3*m*1	2012.33	2012.33	2462	18.26	3600	2315	2437	**2178**	2194.8	33.22	2197	2201.7	47.3413	2183	2197.2	46.51
100*n*3*m*2	2104.33	2104.33	2478	15.08	3600	2350	2471	**2222**	2242.6	36.08	2244	2252.6	33.4319	2229	2250.3	44.26
100*n*3*m*3	2091.67	2091.67	2470	15.32	3600	2355	2532	**2252**	2270.7	18.69	2265	2272.7	59.4172	2259	2272.5	29.87
100*n*3*m*4	2133.33	2133.33	2468	13.56	3600	2389	2500	**2261**	2276	36.93	2269	2277.6	32.0511	2272	2279.6	49.45
100*n*3*m*5	2042.33	2042.33	2425	15.78	3600	2331	2452	**2205**	2224.3	32.35	2216	2226.3	34.7098	2210	2219	50.26
100*n*3*m*6	2094	2094	2463	14.98	3600	2396	2513	2257	2270.2	37.19	2264	2270.6	60.6821	**2250**	2266.1	35.91
100*n*3*m*7	2108	2108	2460	14.31	3600	2346	2505	2252	2259.8	27.98	**2232**	2258	45.1023	2246	2257.9	49.15
100*n*3*m*8	1973.33	1973.33	2432	18.86	3600	2296	2440	2164	2178.8	24.25	2168	2180.4	29.8381	**2157**	2173.4	25.85
100*n*3*m*9	2055.33	2055.33	2547	19.30	3600	2377	2512	2217	2230.6	29.23	2216	2231.6	56.0288	**2213**	2236.8	40.48
100*n*3*m*10	2132.67	2132.67	2401	11.18	3600	2379	2484	2263	2275.1	35.68	2264	2275.3	33.7019	**2262**	2273.2	31.26
100*n*5*m*1	1953	1263.4	2257	44.02	3600	2273	2400	2136	2149.6	40.09	**2109**	2137.1	37.0417	2121	2148.8	52.05
100*n*5*m*2	1832	1217.4	2122	42.63	3600	2173	2313	1978	2006.1	38.35	**1960**	2005.4	41.9943	1977	2000.3	20.20
100*n*5*m*3	1874	1218	2198	44.59	3600	2141	2323	**1985**	2026.7	22.69	2019	2034.1	31.0121	2000	2028.7	48.67
100*n*5*m*4	1893	1181.4	2229	47.00	3600	2169	2328	**2064**	2088.6	15.06	2071	2089.4	32.1701	2069	2095.2	31.71

Table 9 (continued)

Instance		MIP2				USWT	LSWT	GVNS I			GVNS II			GVNS III		
ID	LB	LB_{MIP2}	UB_{MIP2}	$Gap_{LB}(\%)$	CPU	C_{max}	C_{max}	Best	Avg	CPU	Best	Avg	CPU	Best	Avg	CPU
100*n*5*m*5	1811	1290.4	2140	39.70	3600	2161	2344	1966	1986.2	46.27	1963	1979.2	32.8473	**1956**	1981.9	28.95
100*n*5*m*6	1844	1195.6	2195	45.53	3600	2148	2395	2021	2044.8	48.12	**1992**	2023.3	28.3215	2036	2044.1	34.08
100*n*5*m*7	1888	1232.2	2172	43.27	3600	2172	2379	2022	2043	37.59	2021	2036.3	41.1486	**2009**	2043.5	36.23
100*n*5*m*8	1860	1332.6	2135	37.58	3600	2177	2294	2015	2031.8	34.54	1994	2020.2	28.9616	**1982**	2015.5	19.30
100*n*5*m*9	1875	1278.8	2156	40.69	3600	2214	2312	2036	2060	20.02	**2032**	2055.4	33.4116	2041	2072.1	33.96
100*n*5*m*10	1898	1293.6	2172	40.44	3600	2209	2339	**2050**	2082.4	28.37	2052	2076	28.4028	2074	2085.5	30.31
Avg.	2331.64	2124.16	2606.10	20.80	3600	2555.57	2664.20	2438.27	2452.60	32.14	2438.60	2451.44	38.38	2438.60	2452.58	37.80

Table 10 Detailed results found by the proposed approaches for instances with $n = 250$ and $m \in \{2, 3, 5\}$

Instance		MIP2				USWT	LSWT	GVNS I			GVNS II			GVNS III		
ID	LB	LB_{MIP2}	UB_{MIP2}	$Gap_{LB}(\%)$	CPU	C_{max}	C_{max}	Best	Avg	CPU	Best	Avg	CPU	Best	Avg	CPU
250*n*2*m*1	7547	7547	8957	29.15	3600	7716	7843	**7570**	7581.1	67.17	7587	7590	83.98	7578	7586	59.87
250*n*2*m*2	7761.5	7761.5	11920	11.80	3600	7894	8085	7786	7791.6	31.39	7792	7794.7	61.31	**7785**	7792.4	77.87
250*n*2*m*3	7563	7563	9624	33.15	3600	7681	7849	**7581**	7587.7	19.34	7595	7601.2	107.28	7591	7599	59.45
250*n*2*m*4	7533.5	7533.5	7940	17.24	3600	7674	7831	7556	7563	27.78	7564	7570.4	72.02	**7555**	7568	110.09
250*n*2*m*5	7674	7674	9206	15.16	3600	7832	7950	7696	7699.1	14.36	**7695**	7704.6	100.22	**7695**	7702.8	74.49
250*n*2*m*6	7428.5	7428.5	8268	18.42	3600	7604	7695	**7454**	7457.8	23.70	7455	7463	89.09	7457	7461.1	66.80
250*n*2*m*7	7860	7860	9398	16.19	3600	7974	8146	**7879**	7883.6	17.50	7884	7889.7	65.87	7882	7889.2	85.07
250*n*2*m*8	7658.5	7658.5	8689	25.36	3600	7808	7925	**7675**	7683.1	11.65	7685	7688.8	110.06	7680	7688.4	61.46
250*n*2*m*9	7580	7580	9113	14.91	3600	7721	7878	**7598**	7606.4	36.23	7600	7616.6	73.22	7608	7614.9	96.39
250*n*2*m*10	7688	7688	9660	38.56	3600	7822	7948	**7715**	7720.8	19.16	7723	7729.3	75.05	7728	7732.5	75.81
250*n*3*m*1	5065	5065	11103	33.62	3600	5775	6114	5523	5545	56.74	**5516**	5551.6	83.87	5519	5558	53.54
250*n*3*m*2	5095	5095	8033	44.36	3600	5754	6125	**5499**	5547.4	21.61	5553	5586.2	118.98	5547	5579.1	67.71
250*n*3*m*3	5227	5227	14894	65.77	3600	5809	6151	**5536**	5589.7	49.97	5565	5615	95.48	5578	5603.8	88.59
250*n*3*m*4	5230.67	5230.67	7140	66.67	3600	5871	6207	**5604**	5643.8	21.33	5645	5671.4	110.54	5627	5657.4	72.42
250*n*3*m*5	5260	5260	14545	39.94	3600	5930	6344	**5667**	5712.3	69.68	5730	5756.2	79.25	5714	5751	65.19
250*n*3*m*6	5328	5328	15984	23.37	3600	5869	6145	**5629**	5659.8	41.86	5644	5690.1	135.60	5645	5681.3	79.83
250*n*3*m*7	5166.33	5166.33	7667	29.61	3600	5874	6222	**5601**	5634.3	45.72	5624	5655.5	101.49	5642	5662.9	120.24
250*n*3*m*8	5333.33	5333.33	8267	65.61	3600	5936	6300	**5689**	5731	62.06	5699	5731.4	87.07	5697	5745.7	78.28
250*n*3*m*9	5167.67	5167.67	8595	28.59	3600	5807	6186	**5551**	5612.7	65.00	5596	5631.1	93.41	5593	5640.6	83.08
250*n*3*m*10	4981.67	4981.67	6997	58.91	3600	5711	6181	5476	5514.3	49.05	**5453**	5532	73.87	5501	5540.4	82.97
250*n*5*m*1	4613	3103.4	6965	48.63	3600	5370	5729	5003	5078.8	82.88	5049	5082.3	109.14	**4965**	5045.4	72.26
250*n*5*m*2	4714	3096	5975	49.96	3600	5469	5962	5172	5211.1	49.06	5129	5224.4	92.38	**5126**	5221.1	101.82
250*n*5*m*3	4573	3078.6	5864	50.41	3600	5314	5740	**4961**	5073.7	74.60	5070	5100.9	86.26	5077	5118.5	97.97
250*n*5*m*4	4801	3190.4	11668	45.50	3600	5484	5850	5231	5257.4	52.27	5185	5262.2	78.64	**5184**	5252.7	72.16

Table 10 (continued)

Instance		MIP2				USWT	LSWT	GVNS I			GVNS II			GVNS III		
ID	LB	LB_{MIP2}	UB_{MIP2}	$Gap_{LB}(\%)$	CPU	C_{max}	C_{max}	Best	Avg	CPU	Best	Avg	CPU	Best	Avg	CPU
250*n*5*m*5	4681	3034.2	7318	53.96	3600	5364	5815	5112	5172.8	45.63	**5085**	5205.7	112.25	5112	5216.4	76.25
250*n*5*m*6	4600	3053.6	6729	53.94	3600	5418	5805	**5047**	5147.7	83.82	5169	5209.5	80.22	5103	5153.6	84.68
250*n*5*m*7	4710	3133	6052	45.43	3600	5531	5909	**5116**	5219.2	85.89	5189	5258.5	102.16	5175	5240	71.04
250*n*5*m*8	4720	3108.8	6747	75.28	3600	5404	5954	**5102**	5174.7	59.60	5124	5210	82.61	5188	5225.7	100.61
250*n*5*m*9	4712	3141.8	6222	44.90	3600	5518	5984	5186	5237.5	88.34	5209	5263.9	90.02	**5170**	5232	91.54
250*n*5*m*10	4560	3122.8	15614	46.36	3600	5375	5795	**5011**	5081	73.85	5033	5081.9	86.56	5050	5107.7	97.01
Avg.	5827.76	5307.04	9171.80	39.69	3600	6343.63	6655.60	6107.53	6147.3	48.24	6128.23	6165.60	91.26	6125.73	6162.25	80.82

to study these instances in particular because they are difficult to solve. On average, the time to produce a solution within 4% of the best-found solution is equal to 10.7 s over 10 instances.

As it was shown in the previous Tables 8, 9, 10, for GVNS I, GVNS II and GVNS III, the difference between Best and Avg grows with n and m, which is likely to indicate the robustness degradation with the increase of the instance size. It can be noticed that the proposed theoretical lower bound (LB) is equal to the MIP2 formulation's lower bound LB_{MIP2} for 90 instances. Furthermore, the proposed GVNS I, GVNS II, and GVNS III outperformed CPLEX in terms of quality (of the obtained solutions) and speed (i.e., time needed to generate efficient solutions). In addition, GVNS I produced, on average, better results is small computational time in comparison with GVNS II, and GVNS III. This success can be explained by the quality of the initial solution, as the unloading server waiting time minimization strategy contribute significantly to the minimization of the makespan. Hence, one can conclude that the GVNS I is the best method for computing good quality solutions in a small amount of time for the problem $P, S2|s_j, t_j|C_{max}$ with a regular job set.

6 Conclusions and future work

In this paper, the identical parallel machine scheduling problem with two common servers was addressed. Each job has to be loaded by a loading server and unloaded by an unloading server, respectively, immediately before and after being processed on one of the m available machines. The objective function involved makespan minimization. The regular case of this problem is considered, where $\forall i,j \quad p_i < s_j + p_j + t_j$. A mixed-integer-programming (MIP) formulation based on completion time variables, as well as a valid inequality were suggested to solve optimally small-sized instances with up to 10 jobs and 5 machines. In addition, four lower bounds are proposed. Due to the $\mathcal{NP}$-hard nature of the problem, two greedy heuristics based on the minimization of the loading, respectively unloading server waiting time, and a general variable neighborhood search (GVNS) algorithm with different initial solution-finding mechanisms were designed to obtained solution for large-sized instances with up to 250 jobs and 5 machines. Computational experiments were carried out on 120 new instances, divided into 12 groups. For small-sized instances, GVNS algorithm outperformed the MIP2 formulation in terms of the computing time to find an optimal solution. However, for medium and large-sized instances, the GVNS with an initial solution-finding mechanism based on the unloading server waiting time minimization yielded better results than the other approaches. The future work may include larger test instances with equal loading times ($s_j = s$) and/or equal unloading times ($t_j = t$), new neighborhood structures, and implementation of other metaheuristic methods for the problem $P, S2|s_j, t_j|C_{max}$ with a general job set. Additional constraints could also be considered, especially sequence-and-machine-dependent setup times and release dates.

Appendix

Detailed results of MIP2, USWT, LSWT, GVNS I, GVNS II and GVNS III for each instance

In this Appendix 1 the detailed results, on entire data sets for all methods studied in this paper, found by the proposed approaches for instances with $m \in \{2, 3, 5\}$ and $n = \{50, 100, 250\}$ are depicted in the following Tables 8, 9, and 10 , respectively.

Funding Open access funding provided by HEAL-Link Greece.

Data availability The authors declare that, the data set used in this study is available within the supplementary information files of the article.

References

1. Abdekhodaee, A.H., Wirth, A.: Scheduling parallel machines with a single server: some solvable cases and heuristics. Comput. Oper. Res. **29**(3), 295–315 (2002)
2. Abdekhodaee, A.H., Wirth, A., Gan, H.S.: Scheduling two parallel machines with a single server: the general case. Comput. Oper. Res. **33**(4), 994–1009 (2006)
3. Allahverdi, A., Soroush, H.: The significance of reducing setup times/setup costs. Eur. J. Oper. Res. **187**(3), 978–984 (2008)
4. Baker, K.R., Keller, B.: Solving the single-machine sequencing problem using integer programming. Comput. Ind. Eng. **59**(4), 730–735 (2010)
5. Balas, E.: On the facial structure of scheduling polyhedra. In: Mathematical Programming Essays in Honor of George B. Dantzig Part I, pp. 179–218. Springer (1985)
6. Bektur, G., Saraç, T,: 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 (2019)
7. Benmansour, R., Allaoui, H., Artiba, A., Hanafi, S.: Minimizing the weighted sum of maximum earliness and maximum tardiness costs on a single machine with periodic preventive maintenance. Comput. Oper. Res. **47**, 106–113 (2014)
8. Benmansour, R., Sifaleras, A.: Scheduling in parallel machines with two servers: the restrictive case. In: Variable Neighborhood Search: 8th International Conference, ICVNS 2021, Abu Dhabi, United Arab Emirates, 21–25 March, 2021, Proceedings 8, pp. 71–82. Springer International Publishing (2021)
9. Bish, E.K.: A multiple-crane-constrained scheduling problem in a container terminal. Eur. J. Oper. Res. **144**(1), 83–107 (2003)

10. Brimberg, J., Urošević, D., Mladenović, N,: Variable neighborhood search for the vertex weighted k-cardinality tree problem. Eur. J. Oper. Res. **171**(1), 74–84 (2006)
11. Cheng, T., Kravchenko, S.A., Lin, B.M.: Complexity of server scheduling on parallel dedicated machines subject to fixed job sequences. J. Oper. Res. Soc. 1–4 (2020)
12. Diana, R.O.M., de Souza, S.R.: Analysis of variable neighborhood descent as a local search operator for total weighted tardiness problem on unrelated parallel machines. Comput. Oper. Res. **117**, 104886 (2020)
13. Duarte, A., Mladenović, N., Sánchez-Oro, J., Todosijević, R.: Variable Neighborhood Descent. In: Martí, R., Panos, P., Resende, M.G. (eds.) Handbook of heuristics, pp. 1–27. Springer International Publishing, Cham (2016)
14. El Idrissi, A., Benbrahim, M., Benmansour, R., Duvivier, D.: Greedy heuristics for identical parallel machine scheduling problem with single server to minimize the makespan. In: MATEC Web of Conferences, vol. 200, p. 00001. EDP Sciences (2018)
15. Elidrissi, A., Benbrahim, M., Benmansour, R., Duvivier, D.: Variable neighborhood search for identical parallel machine scheduling problem with a single server. In: International Conference on Variable Neighborhood Search, pp. 112–125. Springer (2019)
16. Elidrissi, A., Benmansour, R., Benbrahim, M., Duvivier, D.: Mathematical formulations for the parallel machine scheduling problem with a single server. Int. J. Prod. Res. **59**(20), 6166–6184 (2021)
17. Graham, R.L., Lawler, E.L., Lenstra, J.K., Kan, A.R.: Optimization and approximation in deterministic sequencing and scheduling: a survey. In: Annals of Discrete Mathematics, vol. 5, pp. 287–326. Elsevier (1979)
18. Hamzadayi, A., Yildiz, G.: 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 (2017)
19. Hansen, P., Mladenović, N., Pérez, J.A.M.: Variable neighbourhood search: methods and applications. Ann. Oper. Res. **175**(1), 367–407 (2010)
20. Hansen, P., Mladenović, N., Todosijević, R., Hanafi, S.: Variable neighborhood search: basics and variants. EURO J. Comput. Optim. **5**(3), 423–454 (2017)
21. Hasani, K., Kravchenko, S.A., Werner, F.: Simulated annealing and genetic algorithms for the two-machine scheduling problem with a single server. Int. J. Prod. Res. **52**(13), 3778–3792 (2014)
22. Hasani, K., Kravchenko, S.A., Werner, F.: 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 (2016)
23. Hsu, P.Y., Lo, S.H., Hwang, H.G., Lin, B.M.: Scheduling of anaesthesia operations in operating rooms. Healthcare **9**(6), 640 (2021)
24. Huang, S., Cai, L., Zhang, X.: Parallel dedicated machine scheduling problem with sequence-dependent setups and a single server. Comput. Ind. Eng. **58**(1), 165–174 (2010)
25. Jiang, Y., Yu, F., Zhou, P., Hu, J.: Online algorithms for scheduling two parallel machines with a single server. Int. Trans. Oper. Res. **22**(5), 913–927 (2015)
26. Jiang, Y., Zhou, P., Wang, H., Hu, J.: Scheduling on two parallel machines with two dedicated servers. ANZIAM J. **58**(3–4), 314–323 (2017)
27. Kim, M.Y., Lee, Y.H.: 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 (2012)
28. Koulamas, C.P.: Scheduling two parallel semiautomatic machines to minimize machine interference. Comput. Oper. Res. **23**(10), 945–956 (1996)
29. Kravchenko, S.A., Werner, F.: Scheduling on parallel machines with a single and multiple servers. Otto-von-Guericke-Universitat Magdeburg **30**(98), 1–18 (1998)
30. Mjirda, A., Todosijević, R., Hanafi, S., Hansen, P., Mladenović, N.: Sequential variable neighborhood descent variants: an empirical study on the traveling salesman problem. Int. Trans. Oper. Res. **24**(3), 615–633 (2017)
31. Mladenović, N., Hansen, P.: Variable neighborhood search. Comput. Oper. Res. **24**(11), 1097–1100 (1997)
32. Mladenović, N., Sleptchenko, A., Sifaleras, A., Omar, M. (eds.): Variable neighborhood search. In: 8th International Conference, ICVNS 2021, Abu Dhabi, United Arab Emirates, 21–25 October, 2021, Revised Selected Papers, *LNCS*, vol. 12559. Springer, Cham (2021)
33. Ou, J., Qi, X., Lee, C.Y.: Parallel machine scheduling with multiple unloading servers. J. Sched. **13**(3), 213–226 (2010)

34. Silva, J.M.P., Teixeira, E., Subramanian, A.: Exact and metaheuristic approaches for identical parallel machine scheduling with a common server and sequence-dependent setup times. J. Oper. Res. Soc. **72**(2), 444–457 (2021)
35. Torjai, L., Kruzslicz, F.: Mixed integer programming formulations for the biomass truck scheduling problem. CEJOR **24**(3), 731–745 (2016)
36. Werner, F., Kravchenko, S.A.: Scheduling with multiple servers. Autom. Remote Control **71**(10), 2109–2121 (2010)
37. Xie, X., Li, Y., Zhou, H., Zheng, Y.: Scheduling parallel machines with a single server. In: Proceedings of 2012 International Conference on Measurement, Information and Control, vol. 1, pp. 453–456. IEEE (2012)