RAIRO-Oper. Res. 53 (2019) 351–365
https://doi.org/10.1051/ro/2018062

RAIRO Operations Research
www.rairo-ro.org

TWO-MACHINE FLOW SHOP WITH SYNCHRONIZED PERIODIC MAINTENANCE

Issam Krimi[2,1,3,*], Rachid Benmansour[1], Saïd Hanafi[1] and Nizar Elhachemi[2]

Abstract. In the literature, some works deal with the two-machine flow shop scheduling problem under availability constraints. Most of them consider those constraints only for one machine at a time and also with limited unavailability periods. In this work, we were interested by the unlimited periodic and synchronized maintenance applied on both machines. The problem is NP-hard. We proposed a mixed integer programming model and a variable neighborhood search for solving large instances in order to minimize the makespan. Computational experiments show the efficiency of the proposed methods.

Mathematics Subject Classification. 90B35, 90C11.

Received May 30, 2017. Accepted July 28, 2018.

1. Introduction

Flow shop scheduling problem is one of the most studied scheduling problems in the last decades [12, 25]. In the flow shop scheduling problem, often we assume that the machines are always available and ready to execute the job at anytime. However, in real industry settings, the machines are not ready to perform the job at any time specially in the actual industry where the production is highly capital intensive due to the massive demand. As consequence, more frequent maintenance avoids high breakdown levels that influence the machine utilization. In this case, the more appropriate strategy is a periodic and synchronized maintenance. Wherefore, we study a two-machine flow shop scheduling problem under synchronized and periodic maintenance activities. The word synchronized is used to specify that the maintenance starts at the same time on both machines.

Formally, we are given $N = \{1, 2, \dots, n\}$ a set of n jobs to be performed on a set of two machines $M = \{1, 2\}$. Let p_{jk} be the processing time of job $j \in N$ on machine $k \in M$. Each machine can perform only one job at a time and each job should be performed on the first machine then on the second one. In our work, we assume a non-resumable model (nr) and also the machines are not always available. In this paper, the unavailability constraints are presented by a periodic maintenance on both machines, where maintenance tasks are scheduled independently of the production schedule. The considered problem consists on splitting the scheduling horizon into batches. Each one of them is defined as the period between two consecutive maintenance tasks. We denote by T the length of each batch and d the duration of a maintenance task. The aim is to find a sequence of jobs

Keywords. Flow shop scheduling, preventive maintenance, mixed integer programming model, variable neighborhood search.

[1] LAMIH UMR CNRS 8201, Polytechnic University of Hauts de France, France.
[2] Complex Systems Engineering & Human Systems, Mohammed VI Polytechnic University, Morocco.
[3] Mohammed V University in Rabat, Mohammadia School of Engineers, Morocco.
[*]Corresponding author: issam.krimi@etu.univ-valenciennes.fr; issam.krimi1@gmail.com

that minimize the makespan. We denote by L the set of batches, with $|L| \leq n+1$. Using the notation of Graham *et al.* [15], we denote this problem as $\mathrm{F2/nr-pm}/C_{\max}$, which is NP-hard, since it can be reduced to the single machine scheduling problem with periodic maintenance, which was proved NP-hard [34], if all the processing times on the first machine are equal to zero.

The contributions of this paper are two-fold: (i) a new mixed integer programming (MIP) formulation based on assignment variables is proposed with a lower bound on the optimal value; (ii) a variable neighborhood search metaheuristic is developed to generate a strong upper bound.

The organization of this paper is as follows. First, we summarize the related works in Section 2. Then, in Section 3, we present the MIP formulation based on assignment variables and a lower bound on the optimal solution. In Section 4, we describe a variable neighborhood search metaheuristic used to solve large-sized instances. In Section 5, we discuss the computational experimentation of the proposed methods followed by a conclusion and our work perspectives in the last section.

2. State of the art

In the last decade, scheduling problems considering machine availability (in single machine, parallel machines [29, 46, 47], flow shop [1, 13], job shop [3], open shop [33], flexible flow shop [40] problems) have attracted researchers' attention due to its realistic characteristic. The most studied case is the single machine scheduling problem. One can distinguish two aspects of the unavailability constraint. Either dealing with one unavailability interval or with periodic ones. For the first case, Kacem *et al.* [27] studied a single machine scheduling problem with one availability constraint to minimize the weighted sum of completion times. The authors proposed three exact methods: a branch-and-bound (B&B) method based on new properties and several lower bounds, a MIP model and a dynamic programming method. They found that the B&B method and the dynamic programming are complementary. The proposed methods are able to solve problems with 3000 jobs within a reasonable period of time. Kacem and Chu [26] improved the B&B algorithm using more tight lower bounds. The new algorithm is able to solve instances up to 6000 jobs. For the second case, single machine scheduling problem under periodic maintenance has been studied by several authors. Benmansour *et al.* [2] considered a single machine scheduling problem against a common and restrictive due date. First, they proposed a MIP model without availability constraints to minimize the weighted sum of maximum earliness and maximum tardiness costs and they showed that the proposed model can be reduced to a polynomially solvable one. Then, for the problem where the machine undergoes a periodic maintenance, a second MIP model was proposed to solve small instances to optimality. Kacem *et al.* [28] studied the single machine scheduling problem under periodic unavailability intervals. They considered the maximization of the weighted number of early jobs as the objective function. The authors distinguished two cases: resumable and non-resumable jobs. For the first case, they showed that although the number of the non-availability intervals can be variable and also a subset of jobs could have deadlines, a fully polynomial time approximation scheme (FPTAS) can be proposed. For the second case, when the number of non-availability intervals is constant and all due dates are arbitrary, they showed that the problem admits a polynomial time approximation scheme (PTAS). Ji *et al.* [24] considered the same problem but for minimizing the makespan considering only non-preemptive jobs. The worst-case ratio of the classical Longest Processing Time (LPT) algorithm was shown to be equal to 2. The authors proved that LPT algorithm is the best possible heuristic for the reason that there is no polynomial time approximation algorithm with a worst-case ratio less than 2 unless $P = \mathrm{NP}$.

As for scheduling problems with more than one machine, fifty years ago, Johnson [25] published his seminal paper on flow shop scheduling. He studied the two-machine flow shop scheduling problem to minimize makespan. A polynomial time algorithm in $O(n \log(n))$ was proposed to solve this problem. Lee [35] proved that the two-machine scheduling problem under availability constraint, on at least one machine, is NP-hard. He proposed a pseudo-polynomial dynamic programming algorithm and analyzed the error bound. In his work, he developed two heuristics with worst-case bounds of $1/2$ for the availability constraint on the first machine and $1/3$ for the second machine. In 1999, Lee [36] studied the same problem but under new assumptions. If a job is not finished completely before the next unavailability period, it must be partially restarted. This assumption is named semi-resumable. The author mentioned also two major cases: on the one hand, resumable case when, after the unavailability period, the execution of a job continues without any penalty. On the other hand, non-resumable case when a job should be restated completely after the unavailability period. A pseudo-polynomial dynamic programming algorithm and a heuristic were proposed to solve the problem. Cheng and Wang [8] studied the problem with unavailability constraint on the first machine and proved that the error bound analyzed by Lee [35] was tight. For that reason, the authors developed a heuristic with worst-case bounded by 1/3. Hadda *et al.* [17] assumed that availability constraint is applied on the first machine and the jobs are non-resumable. A new procedure to improve any arbitrary solution was developed in this paper and the results of their method were bounded by 2 times the optimal makespan. The authors presented also a heuristic with a worst-case error of 3/2. For the same problem, but with several unavailability periods, Hadda [16] proposed a polynomial-time approximation scheme. Ng and Kovalyov [41] studied a deterministic two-machine flow shop scheduling problem such that one of the machines is not available during a period of time. They proposed a fully polynomial-time approximation scheme (FPTAS) with $O(n^5/\epsilon^4)$ time complexity. Breit considered the unavailability period on the first [6] and on the second machine [5]. For the first case, he studied the resumable scenario. The author proposed an approximation algorithm to minimize the makespan with a worst-case error bound of 5/4. For the second case, he developed a polynomial-time approximation scheme for this problem. Hnaien *et al.* [23] dealt with the availability constraint applied on the first machine. They presented two MIP models for the problem and several lower bounds used in the developed B&B procedure. The results showed that the MIP models can solve small instances of at most 20 jobs but the B&B was able to solve optimally until 100 jobs. The authors noted that the starting time and the length of unavailability period can be a significant factor to deal with. Kubzin *et al.* [31] studied the two-machine flow shop scheduling problem with several or one unavailability period on the first machine. They distinguished the resumable and the semi-resumable scenario. Considering resumable jobs and several maintenance periods, they presented a fast (3/2)-approximation algorithm. For semi-resumable jobs with one maintenance period, a polynomial-time approximation scheme was developed. Hadda [18] investigated a special case of the two-machine flow shop-scheduling problem considering resumable jobs with several maintenance periods on the second machine. He developed a (4/3)-approximation algorithm. The author showed that the proposed algorithm dominates the (3/2)-approximation algorithm presented by Kubzin *et al.* [31]. In addition, Hadda [19] extended his work by considering a two-machine job-shop scheduling problem limited machine availability on both machines. He proposed a number of characterizations for the optimal solution of a particular case of the two-machine job shop problem with several availability constraints. Some existing polynomial-time approximation schemes were adapted for two particular configurations of the problem. Finally, the author presented a polynomially solvable case. Another type of unavailability constraint was studied by Liao and Tsai [37]. In their problem, they considered that the period of maintenance depends on the number of finished jobs. The authors developed several heuristics. They also presented a heuristic with complexity time of $O(n^2)$. In the same work, they presented a B&B algorithm. Błażewicz *et al.* [4] studied the same problem but considering limited unavailability periods. They proposed constructive and local search based heuristic algorithms. These methods are tested on several instances with size up to 100 jobs and 10 unavailability intervals. The results showed that the algorithms perform well. Considering the same assumptions, Kubiak *et al.* [30] showed that the problem is NP-hard in the strong sense and proposed several proprieties for the optimal schedules and tested a B&B algorithm for the problem. Yang *et al.* [48] considered also the same maintenance policy and developed a heuristic algorithm. Some polynomially solvable cases were also provided. Wang and Cheng [45] studied the two-machine flow shop scheduling problem where setup times could be anticipated, an unavailability period is considered only one of the machines and the jobs are resumable. They proposed two heuristics with worst-case errors less than 2/3. The same authors in [7] considered two successive availability constraints. They presented some proprieties for semi-resumable case and proved that it is sufficient to consider permutation schedules. The authors proposed a heuristic with a worst-case error bound of 2/3 for the non-resumable case. Another type of flow shop scheduling problem that have been considered under periodic maintenance is no-wait scheduling. Espinouse *et al.* [10] considered the maximum completion time ($C_{\max}$) as the objective function. The problem was proved to be NP-hard even with one unavailability period on one single machine, and NP-hard in the strong sense for several unavailability periods. The authors proposed some heuristic algorithms with error bounding analysis. Kubzin and Strusevich [32] studied the two-machine flow shop scheduling problem in no-wait process such that one of the machines undergo a planned maintenance period with a variable length. They presented a polynomial-time approximation scheme to minimize makespan. Ben chihaoui *et al.* [9] considered also the no-wait process where each machine is subject to one unavailability constraint and the jobs have different release dates. They studied the non-resumable schedule that minimize the makespan. Several lower and upper bounds were used in the proposed B&B algorithm. These method was shown effective using large set of instances. Gara-Ali and Espinouse [11] considered a deteriorating preventive maintenance on the second machine. They showed that finding a better timing for starting the maintenance becomes unavoidable especially in the just-in-time environment. The problem is NP-hard and the researchers proposed a B&B method that can solve optimally until 100 jobs. Interested readers are referred to the works of Saidy *et al.* [42], Schmidt [43] and Ma and Zuo [38] who presented surveys on scheduling problems under availability constraints.

Most of works deal with availability constraint on one machine at a time and limited number of unavailability periods. Hence, we study a two-machine flow shop scheduling problem with periodic and synchronized unavailability constraints on both machines.

3. Mixed integer programming formulation

In this section, we present a mixed integer programming (MIP) formulation based on assignment variables. This MIP formulation uses the following decision variables: the continuous variables $C_{\max}$ and C_{jk} represent the makespan and the completion time of job j on the machine k respectively and the binary variables:

$$x_{ijk} = \begin{cases} 1 \text{ if job } i \text{ precedes job } j \text{ on machine } k, \\ 0 \text{ otherwise.} \end{cases}$$

$$y_{jlk} = \begin{cases} 1 \text{ if job } j \text{ belongs to bach } l \text{ on machine } k, \\ 0 \text{ otherwise.} \end{cases}$$

Let B be a big positive value, the proposed MIP formulation is the following:

$$\begin{aligned}
\min \quad & C_{\max} && && (3.1)\\
\text{s.t.} \quad & C_{\max} \geq C_{j2}, && \forall j \in N && (3.2)\\
& C_{j2} \geq C_{j1} + p_{j2}, && \forall j \in N && (3.3)\\
& C_{ik} \geq p_{ik} + C_{jk} - Bx_{ijk}, && \forall i, j \in N, i < j, k \in M && (3.4)\\
& C_{jk} \geq p_{jk} + C_{ik} - B(1 - x_{ijk}), && \forall i, j \in N, i < j, k \in M && (3.5)\\
& \sum_{l \in L} y_{jlk} = 1, && \forall j \in N, k \in M && (3.6)\\
& \sum_{j \in N} p_{jk} y_{jlk} \leq T, && \forall l \in L, k \in M && (3.7)\\
& C_{jk} - p_{jk} \geq (l-1)(T+d) y_{jlk}, && \forall j \in N, l \in L, k \in M && (3.8)\\
& C_{jk} \leq (lT + (l-1)d) y_{jlk} + B(1 - y_{jlk}), && \forall j \in N, l \in L, k \in M && (3.9)\\
& C_{jk} \geq 0, && \forall j \in N, k \in M && (3.10)\\
& x_{ijk}, y_{jlk} \in \{0, 1\}, && \forall i, j \in N, k \in M, l \in L. && (3.11)
\end{aligned}$$

The objective function (3.1) minimizes makespan $C_{\max} = \max\{C_{j2} : j \in N\}$ *i.e.* the completion time of all jobs on machine 2, which is assured by constraints (3.2). Constraints (3.3) guarantee that the completion time

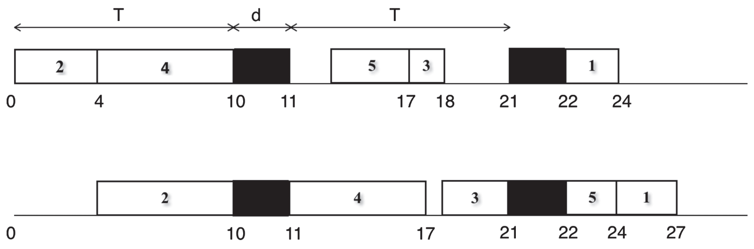

Figure 1. Gantt chart of an optimal schedule.

of a job on machine 2 is not earlier than its completion on machine 1 plus its processing time on machine 2. Constraints (3.4) and (3.5) avoid the overlap between two jobs. Constraints (3.6) state that a job is assigned to only one batch. Constraints (3.7) guarantee that the length of each batch does not exceed its duration T. To avoid that a job starts (or completes) during the maintenance operation, we add constraints (3.8) and (3.9). Constraints (3.10) and (3.11) define the domain of decision variables. The proposed model contains $4n^2 + 4n$ binary variables, $2n + 1$ continuous variables and $6n^2 + 4n$ constraints. Note that batches are not necessarily the same on both machines as showed in Figure 1 where job 4 is assigned to batch 1 on machine 1 and to batch 2 on machine 2.

To illustrate the problem, we provide an example (Fig. 1) of an instance with $n = 5$, $T = 10$, $d = 1$ and $p_{.1} = (2, 4, 1, 6, 4)$, $p_{.2} = (3, 6, 3, 6, 2)$. The optimal makespan is 27.

Lemma 3.1. *There exists an optimal schedule for* $\mathrm{F2/nr-pm}/C_{\max}$ *such that no idle-time exists between two adjacent jobs on each machine.*

Proof. Suppose that a job i precedes job j on machine 2 with an idle-time between them due to the completion of job j on machine 1. Deleting the idle-time by shifting forward job i will not change the makespan. □

Lemma 3.2. *In the problem* $\mathrm{F2/nr-pm}/C_{\max}$*, there is an optimal sequence with a permutation schedule.*

Proof. Let's consider an optimal schedule for the problem such that a job i precedes immediately job j on machine 1 (the jobs are in the same batch). One can distinguish two cases:

Case 1. Job j precedes immediately job i on machine 2. Hence, interchanging i and j on both machines will not increase the makespan.

Case 2. Job j precedes job i on machine 2 (the jobs are not in the same batch). Hence, interchanging i and j on machine 1 will not increase the makespan.

Let consider a job i scheduled in a given batch and a job j scheduled on the next one on machine 1. Two cases can be studied:

Case 1. Job j precedes immediately job i on machine 2. Hence, interchanging i and j on machine 2 will not increase the makespan.

Case 2. Job j precedes job i on machine 2 (i and j are not in the same batch). Hence, interchanging all the jobs which are within the batches containing i and j on machine 1 will not increase the makespan. □

Thus, according to Lemma 3.2, to solve the problem, it is sufficient to consider permutation schedules.

Several lower bounds can be given by Linear programming (LP) relaxation and Lagrangian relaxation (LR). However, these methods are expensive. Thus, we propose the following trivial lower bounds on the optimal solution value $C^*_{\max}$ for the problem.

For any arbitrary sequence of jobs, the makespan is greater than the total completion time on the first machine (id. on machine 2) and also the shortest processing time on the second one (id. on machine 1). The completion time on the first one (id. on the second one) is the sum of the processing times and maintenance tasks durations.

Proposition 3.3. *Let $\overline{p}_k = \sum_{j\in N} p_{jk}$, the sum of the processing times on machine k.*

$$\text{LB1} = \overline{p}_1 + d\left\lfloor \frac{\overline{p}_1}{T} \right\rfloor + \min\{p_{j2} : j \in N\}. \tag{3.12}$$

$$\text{LB2} = \overline{p}_2 + d\left\lfloor \frac{\overline{p}_2}{T} \right\rfloor + \min\{p_{j1} : j \in N\}. \tag{3.13}$$

$$\text{LB} = \max\{\text{LB1}, \text{LB2}\} \tag{3.14}$$

4. Variable neighborhood search

Based on the idea of changing systematically the neighborhood structures to escape from local optimums, variable neighborhood search (VNS) [39] was presented as a single solution metaheuristic. The use of different neighborhood structures is explained [22] by the following observations: (i) it is not necessary to find the same local optimum for all the neighborhood structures. (ii) a global optimum is a local one for all the neighborhood structures. (iii) despite the diversity of the neighborhood structures, for many problems the local optimums are relatively close. Basic VNS and its variants have been applied to many optimization problems [21] as scheduling [44], routing, etc.

To implement the VNS metaheuristic, we need to define the following parameters: an initial solution, a perturbation scheme used to escape from the local optimum, a set of neighborhood structures, a local search and an evaluation scheme.

4.1. Solution representation and initial solution

We presented the solution as a sequence of jobs $\Pi = (\pi_1, \pi_2 ..., \pi_n)$ called permutation-based representation. We used a greedy procedure to construct the initial solution using a priority list of jobs obtained by applying different sorting rules on the first machine (line 1 of Algorithm 1): the shortest processing time rule (SPT), which consists in ordering the jobs by their duration beginning by the shorter, the longest processing time rule (LPT), which consists in ordering the jobs by their duration beginning by the longer and also Johnson's rule (JR); In an optimal solution, job i precedes job j if:

$$\min\{p_{i1}, p_{j2}\} \leq \min\{p_{j1}, p_{i2}\}$$

where, p_{i1}, p_{j1} represent the processing times of job i and j on machine 1 and p_{i2}, p_{j2} represent the processing times of job i and j on machine 2. If the operation is at the first machine, it will be scheduled near to the beginning of the sequence. Otherwise, it will be scheduled near to the end of the sequence. This procedure is repeated until all the jobs are scheduled. Our approach decomposes the overall problem to exploit each machine. On the first one, Algorithm 1 creates the first batch for both machines and it schedules the first job in the list on the first batch (lines 2–8 of Algorithm 1). Then, it goes through the list to schedule the next ones. The procedure verifies the length T of the batch, if we can add a job without overloading the batch, Algorithm 1 schedules it in the current batch just after the previous job. Otherwise, it creates a new batch and the job begins exactly after the maintenance period. The scheduling procedure repeats these steps (lines 9–13 of Algorithm 1) until all the jobs are scheduled on the first machine. Then, the scheduling procedure deals with the second machine. For each job on the priority list, its operation on machine 2 should begin after its completion on machine 1. Algorithm 1 satisfies the execution of the job exactly after its completion on machine 1 without overlapping with the next maintenance period on machine 2. If this is the case, we create the next batch and the job is scheduled on it and its starting time will be the completion time of the previous maintenance period. Otherwise, the job begins on machine 2 just after its completion on machine 1. The scheduling procedure repeats these steps (lines 14–24 of Algorithm 1) until all the jobs are scheduled.

Algorithm 1. The scheduling procedure.

```
1  Input $\pi = (\pi_1, \pi_2, ..., \pi_n)$ the job list given by a priority rule.
2  Initialization of the first batch index $l_k = 1$ on each machine $k \in M$.
3  Set $C_{\pi_1,1} = p_{\pi_1,1}$;
4  if $C_{\pi_1,1} + p_{\pi_1,2} \leq T$ then
5      $C_{\pi_1,2} = C_{\pi_1,1} + p_{\pi_1,2}$;
   end
6  else
7      $l_2 = 2$;
8      $C_{\pi_1,2} = T + d + p_{\pi_1,2}$;
   end
   for $i = 2$ to $n$ do
       /* Scheduling on the first machine */
9      if $C_{\pi_{i-1},1} + p_{\pi_i,1} \leq l_1(T + d) - d$ then
10         $C_{\pi_i,1} = C_{\pi_{i-1},1} + p_{\pi_i,1}$;
       end
11     else
12         create a new batch: $l_1 = l_1 + 1$;
13         $C_{\pi_i,1} = (l_1 - 1)(T + d) + p_{\pi_i,1}$;
       end
       /* Scheduling on the second machine */
14     $k = 2$;
15     $l_2 = l_1$;
16     $j = i - 1$;
17     if $C_{\pi_i,1} \geq C_{\pi_{i-1},2}$ then
18         $k = 1$;
19         $j = i$;
       end
20     if $C_{\pi_j,k} + p_{\pi_i,2} \leq l_2(T + d) - d$ then
21         $C_{\pi_i,2} = C_{\pi_j,k} + p_{\pi_i,2}$;
       end
22     else
23         create a new batch: $l_2 = l_2 + 1$;
24         $C_{\pi_i,2} = (l_2 - 1)(T + d) + p_{\pi_i,2}$;
       end
   end
25 Return $C_{\pi_n,2}$
```

4.2. Neighborhood structures

The complexity and relative error of a solution depend essentially on the systems of neighborhoods used on local search methods. For each problem, several neighborhood structures with different cardinalities can be suggested. Consequently, a set of local optimums are obtained. For scheduling problems based on permutation solutions, three neighborhoods of different cardinalities can be used [14]: transpose (1-Opt), insert (or-Opt) and interchange neighborhood. For the sake of simplicity a neighborhood structure is referred by the name of its corresponding operator. Based on the experimentations of different possible combinations, we choose these structures: *Interchange operator*, *Reverse operator* and *Insert operator*.

- Interchange operator $\mathcal{N}_1$: this operator selects a pair of jobs and exchanges their positions. The operator repeats this process for all the jobs until all the neighborhoods have been searched. We assume a solution of

the problem $\Pi_a = (\pi_1^a, \pi_2^a, ..., \pi_n^a)$ and one of its neighbors $\Pi_b = (\pi_1^b, \pi_2^b, ..., \pi_n^b)$. We fix two different indices $s \neq t$. We exchange the positions: $\pi_s^b = \pi_t^a$, $\pi_t^b = \pi_s^a$ and $\forall c, c \neq (t, s)$ $\pi_c^b = \pi_c^a$.
- Reverse operator $\mathcal{N}_3$: this operator selects two positions and reverses job's order between them. The operator repeats this process for all the jobs until all the neighborhoods have been searched. To more explain the structure, we assume a solution $\Pi_a = (\pi_1^a, \pi_2^a, ..., \pi_n^a)$ and one of its neighbors $\Pi_b = (\pi_1^b, \pi_2^b, ..., \pi_n^b)$. We choose two position $s < t$. $\forall c$, $c \leq s \wedge c \geq t$ $\pi_c^b = \pi_c^a$. Otherwise, $\pi_{s+1}^b = \pi_{t-1}^a$ where $1 \leq s \leq n \wedge 1 \leq t - s < n - 1$.
- Insert operator $\mathcal{N}_4$: this operator removes a job and inserts it in another position. The operator repeats this process for all the jobs until all the neighborhoods have been searched. We assume a solution $\Pi_a = (\pi_1^a, \pi_2^a, ..., \pi_n^a)$ and one of its neighbors $\Pi_b = (\pi_1^b, \pi_2^b, ..., \pi_n^b)$. For $i < j$, $\pi_i^b = \pi_j^a, \pi_{i+1}^b = \pi_i^a, ..., \pi_j^b = \pi_{j-1}^a$. In the case of $i > j$, $\pi_j^b = \pi_{j+1}^a, ..., \pi_{i-1}^b = \pi_i^a, \pi_i^b = \pi_j^a$.

For the evaluation scheme, the makespan is calculated using the scheduling procedure (Algorithm 1) described above and the job list generated after each move.

4.3. Local search

In the literature, the difference between best and the first improvement local search [20] is the way of selecting the next neighbors. In the first one, the entire neighborhood of the current solution is explored and the best solution found so far is returned, whereas in first improvement, the first best neighbor is selected.

In our VNS, we choose the best improvement local search for each neighborhood structure. The local search LS($\mathcal{N}_k$) (Algorithm 2) starts with an initial permutation and tries to repeatedly improve it by moving to a better neighbor considering the current neighborhood structure $\mathcal{N}_k$.

Algorithm 2. LS($\Pi, \mathcal{N}_k$).

```
1 $\Pi$ is the current permutation
2 $\mathcal{N}_k$: the neighborhood structure
  while the time limit is not reached do
3     Generate $\Pi' \in \mathcal{N}_k(\Pi)$, such that
      $C_{\max}(\Pi') = \min_{x \in \mathcal{N}_k(\Pi)} C_{\max}(x)$;
4     if $C_{\max}(\Pi') \leq C_{\max}(\Pi)$ then
5         $\Pi \leftarrow \Pi'$;
      end
  end
```

The perturbation procedure is a random shake within the job list. It depends on the neighborhood structure. For example, the shake procedure for the interchange operator consists in selecting randomly two different positions on the job list and permute them.

4.4. VNS algorithm

We propose the VNS algorithm summarized in Algorithm 3. It starts with an initial solution generated by the scheduling procedure (Algorithm 1). We generate a random neighbor of this solution with respect to the correspondent neighborhood structure $\mathcal{N}_k$. Then, we apply a best improvement local search (Algorithm 2). If no improvement exists, the neighborhood structure will be changed. The algorithm stops when the number of maximum iterations $MaxIter$ is reached.

Algorithm 3. Variable neighborhood search.

```
$k_{\max}$: the number of the neighborhood structures
$MaxIter$: the maximum number of iterations
$CPU_{\max}$: the execution time limit
$\Pi \leftarrow$ Initial_Solution();
iter=1;
while $iter \leq MaxIter \vee CPU \leq CPU_{\max}$ do
    $k \leftarrow 1$;
    while $k \leq k_{\max}$ do
        $\Pi' \leftarrow$ Shake$(\Pi, k)$;
        $\Pi'' \leftarrow$ LS$(\Pi', \mathcal{N}_k)$;
        if $C_{\max}(\Pi'') < C_{\max}(\Pi)$ then
            $\Pi \leftarrow \Pi''$;
            $k \leftarrow 1$;
        end
        else
            $k \leftarrow k + 1$;
        end
    end
    $iter \leftarrow iter + 1$ ;
end
Return $\Pi$
```

5. Computational experiments and results

In this section, we resume our computational experiments of the MIP formulation, the lower bound and also the VNS algorithm. We used CPLEX 12.5 solver for MIP model and C++ language to code the VNS algorithm. The tests were performed on a PC i5 with 2.7 GHz and 8 GB of RAM PC.

5.1. Data generation

Since no similar work in the literature has studied the considered problem. We were inspired by Taillard's code found in OR-library and we adapted it to uniformly generate a large number of instances under three assumptions:

- **Case 1:** $p_{j1}, p_{j2} \in$ U[1,20]
- **Case 2:** $p_{j1} \in$U[2,20] and $p_{j2} \in$U[1,10]
- **Case 3:** $p_{j2} \in$U[2,20] and $p_{j1} \in$U[1,10].

We generated 10 instances for every combination of the following parameters:

- The number of jobs n in $\{10, 15, 20, 25, 40, 50, 60, 70, 80, 90, 100, 500, 1000\}$
- The length of the availability period is equal to $\max\{p_{j1}, p_{j2} : j \in N\}$.

For the sake of simplicity, we chose a unique parameter for the length of the maintenance period ($d = 1$).

5.2. Computational results for the proposed MIP model and the proposed lower bound LB

Table 1 summarizes the results provided by the proposed MIP model. It can optimally solve problems with size up to 25 jobs (see Tab. 2) for the three cases. We use the following metrics: the number of jobs n, the average time (Avg_time) required to find the optimal solution and the average optimal solution (Avg_opt) given by CPLEX. We remark that the instances of case 1 are more difficult to solve in a reasonable amount of time

Table 1. Results of the proposed MIP model.

	Case 1		Case 2		Case 3	
n	Avg_opt	Avg_time (s)	Avg_opt	Avg_time (s)	Avg_opt	Avg_time (s)
10	140.02	3.33	135	2.83	127.8	2.93
15	181.3	115.92	188.1	41.8	171.8	27.11
20	253.8	559.38	248.6	396.67	248.8	73.38
25	318.5	639.27	284	474.99	309	452.3

Table 2. Limits of the proposed MIP model.

	Case 1	Case 2	Case 3
n	Avg_gap (%)	Avg_gap (%)	Avg_gap (%)
30	27.8	18.3	25.3
35	44.2	42.3	49.2
40	58.4	47.3	45.2
50	69.3	64.5	62.1

Table 3. Performance of the proposed lower bound.

	Case 1			Case 2			Case 3		
n	Avg_opt	Avg_LB	Avg_gap(%)	Avg_opt	Avg_LB	Avg_gap(%)	Avg_opt	Avg_LB	Avg_gap(%)
10	140.02	123.6	13.3	135	120.3	12.21	127.8	117.5	8.76
15	181.3	168.7	7.46	188.1	173.9	8.16	171.8	163.2	5.26
20	253.8	230.3	10.2	248.6	232.2	7.06	248.8	232.4	7.05
25	318.5	295.8	7.67	284	264.9	7.21	309	292.6	5.6

comparing with the other cases. However, results of case 2 show that the average time varies significantly with the number of jobs. In all cases, the average time is less than 12 min.

Table 2 shows the limit of our model by highlighting the average gap given by CPLEX. We remark, for all cases, that instances with more than 25 jobs are too difficult to solve within a reasonable time. Note that the execution time limit was fixed to 3600 s. The model has great difficulties finding a feasible solution for problems up to 50 jobs. One can state that instances of case 1, where the jobs may all have significant long processing times, are more difficult to solve with our model.

Table 3 compares the average makespan calculated by the proposed lower bound with the average optimal value of given by CPLEX. For the three cases, the average lower bound is very close to the average optimal makespan (Fig. 2). In addition, the linear relaxation generate, more or less, same lower bound quality comparing with the proposed one. However, in terms of the execution time, one can prefer the proposed LB especially for large instances.

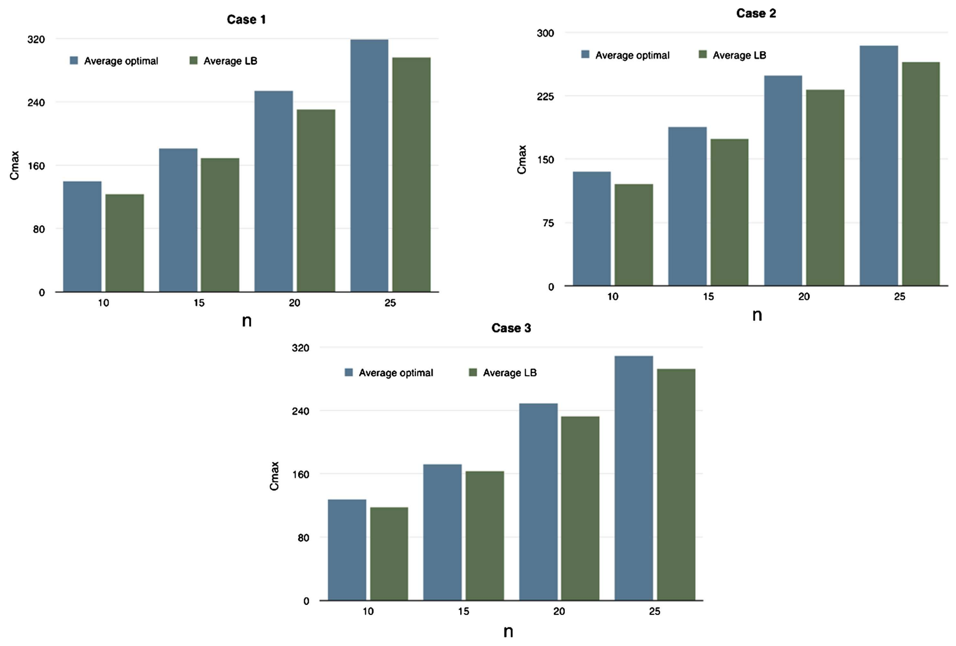

Figure 2. Performance of the proposed lower bound.

5.3. Impact of the availability period T on the proposed MIP model

In order to evaluate the impact of the availability period on the MIP model performance, we consider two different policies to determine T:

$$T_1 = \max\{p_{j1}, p_{j2} : j \in N\} \tag{5.1}$$

$$T_2 = 2 \times \max\{p_{j1}, p_{j2} : j \in N\} \tag{5.2}$$

According to Figure 3, the length of the availability period affects significantly the CPU time. The MIP model requires more time, for each case, to optimally solve small and medium instances when the length T increases.

5.4. Computational results for the proposed VNS algorithm

In this section, we intend to evaluate the VNS algorithm. We set the stopping criteria parameters as follows: the time limit $CPU_{\max}$ is equal to 600 s and the maximum iterations $MaxIter$ is equal to 1000. The results of the experiments for small and medium-seized problems are reported in Table 4 and large size instances in Table 5. Both tables summarize the average results for each n. As expected, the proposed VNS is very efficient for instances with $n \leq 25$ and it provides solutions with small deviation for $n \geq 40$ within a reduced computational time. Comparing with best solutions obtained by CPLEX and also the computed lower bound, our proposed metaheuristic provides convincing results in general. The deviation is calculated as follows:

For $n > 25$: dev $= \frac{\text{Best\_VNS} - \text{LB}}{\text{LB}} \times 100$. Otherwise: dev $= \frac{\text{Best\_VNS} - \text{CPX\_Opt}}{\text{CPX\_Opt}} \times 100$.
Where Best_VNS is the best makespan obtained after 10 runs of the VNS algorithm, CPX_Opt is the optimal makespan given by CPLEX and LB is the lower bound.

We summarize, in Figures 4 and 5, the performance of the VNS algorithm. These figures show the deviation between the average optimal makespan, or the average lower bound, and VNS solution. Instances of case 1

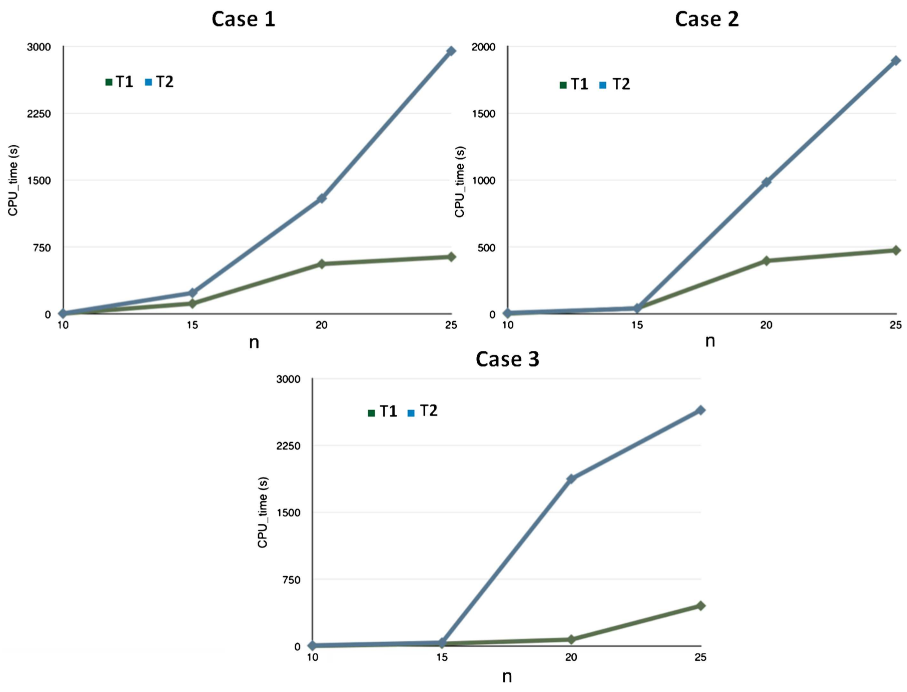

FIGURE 3. Impact of the availability period T on the proposed MIP model.

TABLE 4. Results of the proposed VNS for small and medium size instances.

	Case 1		Case 2		Case 3	
n	Avg_dev	Avg_time (s)	Avg_dev	Avg_time (s)	Avg_dev	Avg_time(s)
10	0.021	0.15	0	0.11	0.01	0.13
15	0.012	0.32	0.011	0.31	0.012	0.34
20	0.033	0.59	0.025	0.69	0.022	0.62
25	0.042	1.23	0.062	1.31	0.021	0.65

TABLE 5. Results of the proposed VNS for large size instances.

	Case 1		Case 2		Case 3	
n	Avg_dev	Avg_time (s)	Avg_dev	Avg_time (s)	Avg_dev	Avg_time(s)
40	0.119	3.42	0.079	3.77	0.063	3.68
50	0.117	9.16	0.095	6.56	0.062	7.06
60	0.118	11.23	0.104	11.06	0.05	11.49
70	0.104	17.85	0.085	17.35	0.052	18.18
80	0.098	25.6	0.08	24.7	0.047	27.65
90	0.092	30.88	0.075	36.3	0.043	39.32
100	0.094	50.82	0.073	51.52	0.048	52.36
500	0.112	172.47	0.111	74.98	0.141	245.56
1000	0.101	600	0.093	600	0.203	600

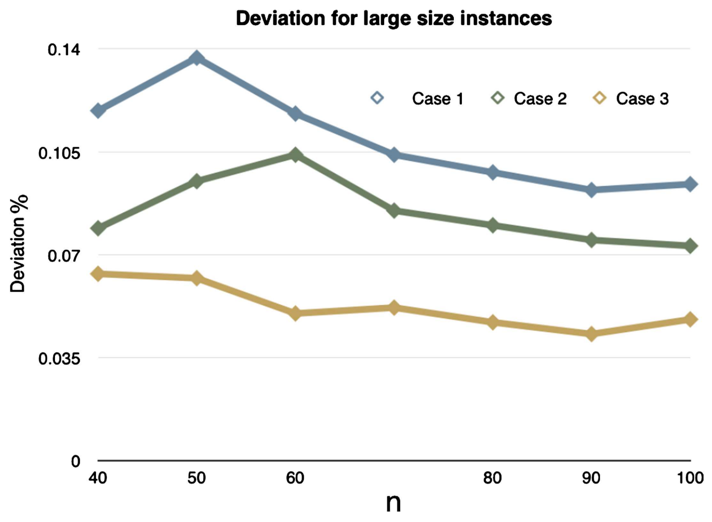

FIGURE 4. Performance of the proposed VNS for large instances.

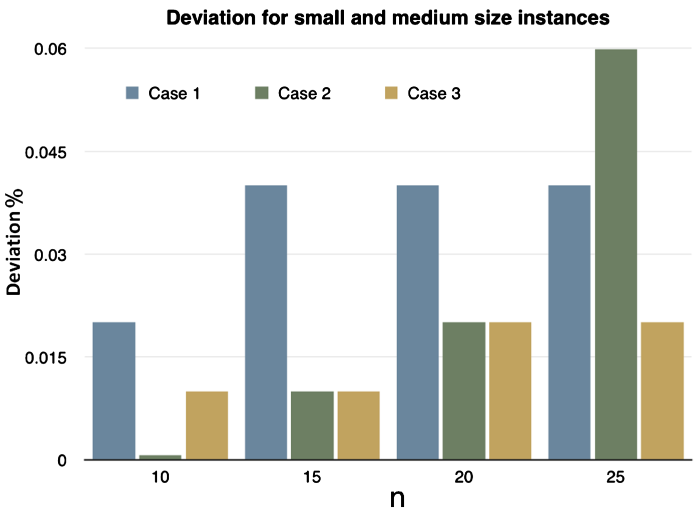

FIGURE 5. Performance of the proposed VNS for small and medium instances.

are the most difficult to solve compared with the other cases except some particular instances. However, the deviation is still very small for instances less than 25 jobs and accepted for ones with more than 30 jobs.

The proposed MIP was shown very efficient for small and medium size instances. In all cases, the model can solve, optimally, problems with less than 30 jobs within reasonable time up to 650 s. For large instances, the gap obtained by CPLEX is satisfactory (less than 13%) and that was noticed in the three cases. The linear relaxation of the model generates acceptable lower bounds when compared with MIP results of instances up to three. However, it turned out that the lower bound is much more efficient in term of deviation and execution time. The use of such lower bound is necessary to validate our metaheuristic. In real situations, the problem can be more complicated considering a significant number of jobs to execute. Thus, we proposed a VNS metaheuristic to get a trade-off between the solution quality and time consumption. Computational results show that the proposed metaheuristic is efficient and effective. In one hand, for small and medium-size instances, the deviation between the solution obtained by the VNS algorithm and the optimal makespan is less than 0.07% and that within an execution time up to 2 s. In the other hand, these solutions are satisfactory especially if we note that the comparison was based on the proposed lower bound. The time execution was fixed to 600 s for large instances but most of runs did not reach 300 s except instances with 1000 jobs.

6. Conclusion and perspectives

In this paper, we presented a MIP model for the two-machine flow shop scheduling problem with synchronized and periodic maintenance in order to minimize the makespan. A lower bound was proposed. To deal with large-scale instances, we developed a VNS algorithm. Our computational studies have shown that the models can not solve optimally instances up to 25 jobs. Alternatively, the use of the VNS algorithm was very effective in solving medium and large size instances. The length of the availability periods has an impact on the difficulty to solve the problem. As future work, we will extend the problem by considering the flexible policy for preventive maintenance. In addition, it is very important to study the problem from a theoretical perspective. Thus, we will investigate a number of theoretical results, that will ameliorate the general contribution.

References

[1] R. Aggoune and M.C. Portmann, Flow shop scheduling problem with limited machine availability: a heuristic approach. *Int. J. Prod. Econ.* **99** (2006) 4–15.

[2] R. Benmansour, H. Allaoui, A. Artiba and S. Hanafi, Minimizing the weighted sum of maximum earliness and maximum tardiness costs on a single machine with periodic preventive maintenance. *Comput. Oper. Res.* **47** (2014) 106–113.

[3] M. Benttaleb, F. Hnaien and F. Yalaoui, Two-machine job shop problem for makespan minimization under availability constraint. *IFAC-PapersOnLine* **49** (2016) 132–137.

[4] J. Błażewicz, J. Breit, P. Formanowicz, W. Kubiak and G. Schmidt, Heuristic algorithms for the two-machine flowshop with limited machine availability. *Omega* **29** (2001) 599–608.

[5] J. Breit, An improved approximation algorithm for two-machine flow shop scheduling with an availability constraint. *Inf. Process. Lett.* **90** (2004) 273–278.

[6] J. Breit, A polynomial-time approximation scheme for the two-machine flow shop scheduling problem with an availability constraint. *Comput. Oper. Res.* **33** (2006) 2143–2153.

[7] T.C.E. Cheng and G. Wang, Two-machine flowshop scheduling with consecutive availability constraints. *Inf. Process. Lett.* **71** (1999) 49–54.

[8] T.C.E. Cheng and G. Wang, An improved heuristic for two-machine flowshop scheduling with an availability constraint. *Oper. Res. Lett.* **26** (2000) 223–229.

[9] F. Ben Chihaoui, I. Kacem, A.B. Hadj-Alouane, N. Dridi and N. Rezg, No-wait scheduling of a two-machine flow-shop to minimise the makespan under non-availability constraints and different release dates. *Int. J. Prod. Res.* **49** (2011) 6273–6286.

[10] M.L. Espinouse, P. Formanowicz and B. Penz, Minimizing the makespan in the two-machine no-wait flow-shop with limited machine availability. *Comput. Ind. Eng.* **37** (1999) 497–500.

[11] A. Gara-Ali and M.L. Espinouse, A two-machine flowshop with a deteriorating maintenance activity on the second machine. In: Industrial Engineering and Systems Management (IESM), 2015 International Conference on. IEEE (2015) 481–488.

[12] M.R. Garey, D. S. Johnson and R. Sethi, The complexity of flowshop and jobshop scheduling. *Math. Oper. Res.* **1** (1976) 117–129.

[13] H. Gholizadeh, R. Tavakkoli-Moghaddam and B. Tootooni, Minimizing the makespan in a flow shop scheduling problem with sequence-dependent setup times and periodic maintenance by a hybrid algorithm. In: 2012 International Conference on Industrial Engineering and Operations Management (2012) 806–814.

[14] F.W. Glover and G.A. Kochenberger, Handbook of Metaheuristics, Vol. 57. Springer Science & Business Media, Berlin (2006).

[15] R.L. Graham, E.L. Lawler, J.K. Lenstra and A.R. Kan, Optimization and approximation in deterministic sequencing and scheduling: a survey. *Ann. Discrete Math.* **5** (1979) 287–326.

[16] H. Hadda, A polynomial-time approximation scheme for the two-machine flow shop problem with several availability constraints. *Optim. Lett.* **6** (2012) 559–569.

[17] H. Hadda, N. Dridi and S. Hajri-Gabouj, An improved heuristic for two-machine flowshop scheduling with an availability constraint and nonresumable jobs. *4OR-Q J. Oper. Res.* **8** (2010) 87–99.

[18] H. Hadda, A (4/3)- approximation algorithm for a special case of the two machine flow shop problem with several availability constraints. *Optim. Lett.* **3** (2009) 583–592.

[19] H. Hadda, Approximation results for the two-machine job shop under limited machine availability. *OPSEARCH* **54** (2017) 651–662.

[20] P. Hansen and N. Mladenović, First vs. best improvement: an empirical study. *Discrete Appl. Math.* **154** (2006) 802–817.

[21] P. Hansen, N. Mladenović and J.A. Moreno Pérez, Variable neighbourhood search: methods and applications. *Ann. Oper. Res.* **175** (2010) 367–407.

[22] P. Hansen, N. Mladenović, R. Todosijević and S. Hanafi, Variable neighborhood search: basics and variants. *EURO J. Comput. Optim.* **5** (2016) 423–454.

[23] F. Hnaien, F. Yalaoui and A. Mhadhbi, Makespan minimization on a two-machine flowshop with an availability constraint on the first machine. *Int. J. Prod. Econ.* **164** (2015) 95–104.

[24] M. Ji, Y. He and T.C.E. Cheng, Single-machine scheduling with periodic maintenance to minimize makespan. *Comput. Oper. Res.* **34** (2007) 1764–1770.

[25] S.M. Johnson, Optimal two-and three-stage production schedules with sertup times included. *Nav. Res. Logist. Quart.* **1** (1954) 61–68.

[26] I. Kacem and C. Chu, Efficient branch-and-bound algorithm for minimizing the weighted sum of completion times on a single machine with one availability constraint. *Int. J. Prod. Econ.* **112** (2008) 138–150.

[27] I. Kacem, C. Chu and A. Souissi, Single-machine scheduling with an availability constraint to minimize the weighted sum of the completion times. *Comput. Oper. Res.* **35** (2008) 827–844.

[28] I. Kacem, H. Kellerer and Y. Lanuel, Approximation algorithms for maximizing the weighted number of early jobs on a single machine with non-availability intervals. *J. Comb. Optim.* **30** (2015) 403–412.

[29] I. Kacem, M. Sahnoune and G. Schmidt, Strongly fully polynomial time approximation scheme for the weighted completion time minimization problem on two-parallel capacitated machines. *RAIRO: OR* **51** (2017) 1177–1188.

[30] W. Kubiak, J. Błażewicz, P. Formanowicz, J. Breit and G. Schmidt, Two-machine flow shops with limited machine availability. *Eur. J. Oper. Res.* **136** (2002) 528–540.

[31] M.A. Kubzin, C.N. Potts and V.A. Strusevich, Approximation results for flow shop scheduling problems with machine availability constraints. *Comput. Oper. Res.* **36** (2009) 379–390.

[32] M.A. Kubzin and V.A. Strusevich, Two-machine flow shop no-wait scheduling with machine maintenance. *4OR: A Quart. J. Oper. Res.* **3** (2005) 303–313.

[33] M.A. Kubzin and V.A. Strusevich, Planning machine maintenance in two-machine shop scheduling. *Oper. Res.* **54** (2006) 789–800.

[34] C.Y. Lee, Machine scheduling with an availability constraint. *J. Global Optim.* **9** (1996) 395–416.

[35] C.Y. Lee, Minimizing the makespan in the two-machine flowshop scheduling problem with availability constraint. *Oper. Res. Lett.* **20** (1997) 129–139.

[36] C.Y. Lee, Two-machine flowshop scheduling problem with availability constraints. *Eur. J. Oper. Res.* **114** (1999) 420–429.

[37] L.M. Liao and C.H. Tsai, Heuristics algorithms for two-machine flowshop with availability constraints. *Comput. Ind. Eng.* **56** (2009) 306–311.

[38] Y. Ma and C. Zuo, A survey of scheduling with deterministic machine availability constraints. *Comput. Ind. Eng.* **58** (2010) 199–211.

[39] N. Mladenović and P. Hansen, Variable neighborhood search: Principles and applications. *Eur. J. Oper. Res.* **130** (2001) 449–467.

[40] B. Naderi, M. Zandieh and M. Aminnayeri, Incorporating periodic preventive maintenance into flexible flowshop scheduling problems. *Appl. Soft Comput.* **11** (2011) 2094–2101.

[41] C.T. Ng and M.Y. Kovalyov, An FPTAS for scheduling a two-machine flowshop with one unavailability interval. *Nav. Res. Logistics (NRL)* **51** (2004) 307–315.

[42] D. Saidy, H. Reza and M. Taghi Taghavi-Fard, Study of scheduling problems with machine availability constraint. *J. Ind. Syst. Eng.* **1** (2008) 360–383.

[43] G. Schmidt, Scheduling with limited machine availability. *Eur. J. Oper. Res.* **121** (2000) 1–15.

[44] R. Todosijević, R. Benmansour, S. Hanafi and N. Mladenović, Nested general variable neighborhood search for the periodic maintenance problem. *Eur. J. Oper. Res.* **252** (2016) 385–396.

[45] X. Wang and T.C.E. Cheng, Heuristics for two-machine flowshop scheduling with setup times and an availability constraint. *Comput. Oper. Res.* **34** (2007) 152–162.

[46] D. Xu, Z. Cheng, Y. Yin and H. Li, Makespan minimization for two parallel machines scheduling with a periodic availability constraint. *Comput. Oper. Res.* **36** (2009) 1809–1812.

[47] D. Xu and D.L. Yang, Makespan minimization for two parallel machines scheduling with a periodic availability constraint: mathematical programming model, average-case analysis, and anomalies. *Appl. Math. Model.* **37** (2013) 7561–7567.

[48] D.L. Yang, C.J. Hsu and W.H. Kuo, A two-machine flowshop scheduling problem with a separated maintenance constraint. *Comput. Oper. Res.* **35** (2008) 876–883.