RAIRO-Oper. Res. 53 (2019) 289–302
https://doi.org/10.1051/ro/2018055

RAIRO Operations Research
www.rairo-ro.org

A VARIABLE NEIGHBORHOOD SEARCH ALGORITHM FOR SOLVING THE SINGLE MACHINE SCHEDULING PROBLEM WITH PERIODIC MAINTENANCE

Hanane Krim[1,*], Rachid Benmansour[1,2], David Duvivier[1] and Abdelhakim Artiba[1]

Abstract. In this paper we propose to solve a single machine scheduling problem which has to undergo a periodic preventive maintenance. The objective is to minimize the weighted sum of the completion times. This criterion is defined as one of the most important objectives in practice but has not been studied so far for the considered problem. As the problem is proven to be NP-hard, and a mathematical model is proposed in the literature, we propose to use General Variable Neighborhood Search algorithm to solve this problem in order to obtain near optimal solutions for the large-sized instances in a small amount of computational time.

Mathematics Subject Classification. 90–XX, 68–XX.

Received April 30, 2017. Accepted June 27, 2018.

1. Introduction

For manufacturing companies, the largest part of the investment is dedicated to production systems which must be in operating conditions as often as possible. Unfortunately, these systems are subject to deterioration and failure and therefore have to undergo a preventive maintenance to avoid the consequences of failures. Therefore, these systems may be unavailable during the scheduling horizon due to failures, changeovers, or maintenance. This is why scheduling problems with unavailability constraints are among the most important problems encountered in industrial environments. For that reason, this domain of research has received increasing attention in the last decade. Applications and examples which reflect the growing importance of maintenance are provided by Ben-Daya *et al.* in their book [5] and some maintenance strategies and reliability optimization problems are proposed by Artiba *et al.* in [2].

In this work we propose to study the single machine scheduling problem in order to minimize weighted sum completion times where the machine has to undergo a periodic preventive maintenance to avoid failures. Preventive maintenance is a scheduled program of regular inspections, adjustments, lubrication, or replacement of worn or failing parts in order to maintain an asset's function, and efficiency [9]. We only consider herein the non-preemptive case. As an illustration of this problem, let us consider a critical machine where the cost

Keywords. Single machine scheduling, periodic preventive maintenance, metaheuristic, variable neighborhood search.

[1] University of Valenciennes and Hainaut Cambrésis – LAMIH, UMR CNRS 8201, France.

[2] INSEA, Laboratoire SI2M Rabat, Maroc.

[*] Corresponding author: Hanane.Krim@univ-valenciennes.fr

of failure is very high and preventive maintenance, with a much lower cost, is applied to minimize the risk of failure. Also, it can be considered that preventive maintenance is carried out to meet a regulatory requirement or that the machine must undergo, at regular intervals and for practical reasons, a cleaning, cooling or a tool change. On that basis, we assume that the risk of a major breakdown is greatly reduced when periodic preventive maintenance is applied. Consequently, the random minor failures that may occur on the machine have no impact on the execution of the jobs. In addition, they have such negligible duration that we do not take them into account later. For this reason, as part of our study we consider that the risk of failures is non-existent or at least negligible thanks to an efficient preventive maintenance strategy.

Based on the three field notation $\alpha|\beta|\gamma$ known as the Graham triplet [13], this problem is denoted as $1/\text{pm}/\sum_{i=1}^{n} w_i c_i$. The first field means that there exists only one machine. The second one specifies that the periodic maintenance (pm) aspect is considered. Finally, the last field contains the problem's objective function where c_i is the completion time of a job J_i and w_i is its weight. Many works dealing with periodic maintenance on a single machine have been published but to the best of our knowledge no one considered such a problem with the objective of minimizing the sum of the weighted completion times. This criterion is defined as one of the most important objectives in practice as it is stated by Ashour in his book [3]. Indeed, it can be considered an auxiliary function measuring the total work-in-process inventory cost in a manufacturing system. The weights can also quantify the priority/urgency of a job based on a possible shortage in a downstream level.

The studied problem is NP-hard since Chen *et al.* [25] proved that the special case of minimizing the sum of completion times with periodic maintenance is NP-hard. We provide hereafter a brief overview of previous related works.

Lee and Limane [22] investigated a non-premptive single machine scheduling problem to minimize the sum of job flow times subject to scheduled maintenance with preemption of the jobs not allowed. They proved the NP-hardness of the problem and showed that the Shortest Processing Time (SPT) rule has a worst case error bound of 2/7. Liao and Chen [25] proposed to minimize the maximum tardiness. They developed a heuristic and a branch-and-bound algorithm to solve the problem. Lee *et al.* [23] developed an efficient heuristic to minimize the number of tardy jobs. Ji *et al.* [19] considered minimizing the makespan. They showed that the worst case ratio of the LPT rule is 2, and proved that there is no polynomial time approximation algorithm with a worst-case ratio of less than 2 unless $P = \text{NP}$. Chen *et al.* [7] minimize the total flow time. They proved some theorems and developed a branch and bound algorithm to find an optimal schedule. Low *et al.* [26] presented a particle swarm optimization algorithm and Ebrahimy *et al.* [11] developed a dynamic genetic algorithm to solve that problem with the aim of minimizing the makespan. Wen-Jinn *et al.* [8] proposed a heuristic to solve the single-machine scheduling problem with periodic maintenance for the preemptive case. They minimized the total flow time and the maximum tardiness simultaneously. Benmansour *et al.* [6] investigated the single machine scheduling problem with periodic maintenance in order to minimize the weighted sum of maximum earliness and maximum tardiness costs. They proved the NP-hardness of this problem and proposed an efficient heuristic to solve it. Roux *et al.* [31] proposed an hybridization of an optimization algorithm named Nelder–Mead and a simulation multimodel which is able to integrate production aspects and maintenance strategies. Su *et al.* [33] considered four single machine scheduling problems with a variable machine maintenance. The objectives are mean lateness, maximum tardiness, total flow time and mean tardiness, respectively. They proposed a polynomial-time exact algorithm for the four criteria considered. In [10], Wei-Wei *et al.* proposed to minimize the makespan with a flexible maintenance and release date constraints. They proved that the problem can be solved in polynomial time with Earliest Release Date (ERD) rule in the resumable case. For the non-resumable case, they proposed a mixed linear integer programming (MILP) model, a heuristic named ERD-LPT and a branch-and-bound algorithm to solve the problem. Recently a scheduling problem with multiple unavailability constraints was proposed by Yazdani *et al.* [35]. The considered criteria are the sum of maximum earliness and tardiness of jobs. They proposed a mathematical model and a variable neighborhood search algorithm to solve the problem. For problems with setup-times, Angel-Bello *et al.* [1] proposed a mixed integer programming model for solving the single machine scheduling problem with availability constraints and sequence-dependent setup costs. Later, Luo *et al.* [27] investigated a single-machine scheduling problem with workload-dependent maintenance duration with the aim of minimizing the total weighted completion time. They proposed a $(2+\epsilon)$-approximation algorithm and a fully polynomial time approximation scheme to solve the problem.

The remainder of the paper is organized as follows: The problem is described in Section 2. Three lower bounds based on job splitting are presented in Section 3. In Section 4 we describe the proposed adaptation of General Variable Neighborhood Search (GVNS) metaheuristic to our scheduling problem. Computational experiments and the calibration tests of GVNS parameters are given in Section 5. Section 6 concludes the study with a discussion and future extensions.

2. Problem description

The addressed problem can be stated as follows: Let $N = \{1, 2, \ldots, n\}$ be a set of n jobs to be processed non-preemptively on a single machine. The machine has to undergo a periodic preventive maintenance during the scheduling horizon. We assume that the machine is new at the beginning of the schedule ($t = 0$) and each preventive maintenance action renews the machine. The period and the duration of maintenance are known in advance and the constraints relating to maintenance are defined as follows: The machine, typically a critical machine of the shop, has to undergo a preventive maintenance of duration δ each time point $kT - \delta$ ($k \in \mathbb{N}^*$). T is defined as the interval between two consecutive maintenance activities. The periodic maintenance induces several unavailability periods. The studied problem $1/\mathrm{pm}/\sum_{i=1}^{n} w_i c_i$ can be formulated as a Mixed Integer Linear Programming model.

Let $\pi = \{\pi_1, \pi_2, \ldots, \pi_k, \ldots, \pi_n\}$ be a permutation of N where π_k indicates the job which is processed in the kth position.

Due to the complexity of the considered problem, only small size instances can be solved optimally by exact methods such as branch and bound or dynamic programming. This is why we propose a metaheuristic to solve large instances of the problem.

3. Lower bound based on job splitting

Job splitting is proposed as a general technique to obtain lower bounds. This concept was introduced by Posner [30] and used later by Belouadah *et al.* [4]. They proved that if a job J_i is split into two jobs J_j and J_k with processing times $p_i = p_j + p_k$ and weights $w_i = w_j + w_k$, it decreases the total weighted completion time by $p_k w_j$. This method was used by Kacem *et al.* in [20] to propose lower bounds for a single machine scheduling problem with only one unavailability period of the machine during the scheduling horizon.

Let P denote the original problem with no split jobs and let P_1 be the corresponding problem in which each job J_i is split into n_i pieces. Each piece $o_{k,i}$ has a processing time $p_{k,i}$ and a weight $w_{k,i}$ ($i \in N, k \in \{1, \ldots, n_i\}$) such that $p_i = \sum_{k=1}^{n_i} p_{k,i}$ and $w_i = \sum_{k=1}^{n_i} w_{k,i}$, the pieces $o_{k,i}$ are constrained to be scheduled contiguously. Let σ^* be an optimal schedule for P and σ_1^* the corresponding optimal schedule for P_1. Belouadah *et al.* [4] proved the following relation:

$$\sum_{i \in N} w_i c_i(\sigma^*) - \sum_{i \in N} w_i c_i(\sigma_1^*) = \mathrm{CBRK} \tag{3.1}$$

and

$$\mathrm{CBRK} = \sum_{i \in N} \left(\sum_{k=1}^{n_i - 1} w_{k,i} \sum_{h=k+1}^{n_i} p_{h,i} \right). \tag{3.2}$$

As suggested by Belouadah *et al.* [4], CBRK may be thought as the cost of breaking all the jobs J_i into n_i pieces.

By relaxing the contiguity constraint in problem P_1, we obtain a relaxed problem denoted as P_2. Let σ_2^* be an optimal schedule for problem P_2. Belouadah *et al.* [4] proposed the following relation:

$$\sum_{i \in N} w_i c_i(\sigma^*) \geq \sum_{i \in N} w_i c_i(\sigma_2^*) + \mathrm{CBRK}. \tag{3.3}$$

To get the lower bounds, the jobs should be split in such a way that the resulting problem P_2 corresponds to a special case for which an optimal schedule σ_2^* is known. This is particularly the case when the processing times are constant. In this context, let us consider three special splits:

First split: Each job J_i $(i \in N)$ is split into two pieces such that $p_{1,i} = p_{2,i} = p/2$ and $w_{1,i} = w_i$, $w_{2,i} = 0$ with $p = \min\limits_{1 \le i \le n} p_i$. Let $\sigma_2^{'*}$ denote the optimal solution of this problem. Hence, we have:

$$\text{LB1} = \sum_{i \in N} w_i c_i(\sigma_2^{'*}) + \text{CBRK}. \tag{3.4}$$

Second split: The second lower bound (LB2) is obtained by considering the following split: Each job J_i $(i \in N)$ is split into two pieces such that $p_{1,i} = p_{2,i} = p/2$ and $w_{1,i} = w_{2,i} = w/2$ with: $p = \min\limits_{1 \le i \le n} p_i$ and $w = \min\limits_{1 \le i \le n} w_i$. Let denote $\sigma_2^{''*}$ the optimal solution of this problem. Hence, we have:

$$\text{LB2} = \sum_{i \in N} w_i c_i(\sigma_2^{''*}) + \text{CBRK}. \tag{3.5}$$

Third split: P_2 is efficiently solvable when all processing times are set to one unit time $(p_{k,i} = 1)$, which means that each job J_i $(i \in N)$ is split into p_i pieces. The weights are set as follows:

If $w_i \ge p_i$, $w_{k,i} = 1$, for $k \in \{1, \ldots, p_i\}$,
If $w_i \le p_i$, for $w_{k,i} = 1$, $k \in \{1, \ldots, w_i\}$ and $w_{k,i} = 0$, for $k \in \{w_i + 1, \ldots, p_i\}$.
Let $\sigma_2^{'''*}$ denote the optimal solution of this problem. Hence, we have:

$$\text{LB3} = \sum_{i \in N} w_i c_i(\sigma_2^{'''*}) + \text{CBRK}. \tag{3.6}$$

4. General Variable Neighborhood search

To deal with large-sized instances, we need to develop heuristics which are able to find very good solutions in a small or at least reasonable amount of time. With this aim in mind, in this section a General Variable Neighborhood Search is proposed.

Variable neighborhood search (VNS) is a flexible framework for building heuristics, proposed by Mladenovic and Hensen in 1997 [29]. It uses the idea of changing the neighborhood systematically in order to avoid being trapped in a local optimum during the search.

The basic VNS combines two search approaches: A stochastic approach called "shaking step" that is defined as a perturbation of the incumbent solution by jumping to the kth neighborhood and a deterministic approach in which a local search is applied. When we replace the local search by the Variable Neighborhood Descent (VND), it can be seen as a generalization of a local search since it explores several neighborhood structures at once instead of one. The resulting algorithm is called a General VNS (GVNS). VND is a deterministic heuristic, it starts from a feasible solution as the current one and then carries out a series of neighborhood searches through operators. A best solution in each neighborhood is determined. This heuristic has been successfully applied for solving different combinatorial optimization problems [15, 16, 28].

GVNS has received much attention and showed good performances compared to other VNS variants. It has been applied to many optimization problems [18, 24, 34].

The proposed GVNS is described in the following subsections.

4.1. Neighborhood structures

Five neighborhoods are proposed in the following paragraphs:

Reverse_two_consecutive ($\mathscr{N}_1(\pi)$): The neighborhood structure consists of all solutions obtained from solution π by swapping two consecutive jobs of π. The complexity of this neighborhood structure is $\Theta(n)$.

Swap ($\mathscr{N}_2(\pi)$): The neighborhood structure consists of all solutions obtained from solution π by swapping all pairs of jobs. The complexity of this neighborhood structure is $\Theta(n^2)$.

Insertion ($\mathscr{N}_3(\pi)$): The neighborhood structure consists of all solutions obtained from solution π by inserting each job of π at the position k ($1 \leq k \leq n$). The complexity of this neighborhood structure is $\Theta(n)$.

Restart ($\mathscr{N}_4(\pi)$): The neighborhood structure consists of all solutions obtained from solution π in such a way as to start from each job π_i and finish with the job π_{i-1} ($i \geq 2$). The complexity of this neighborhood structure is $\Theta(n)$

2-opt ($\mathscr{N}_5(\pi)$): The 2-opt operator is the most classical one when it comes to solve the traveling salesman problem [17]. It removes two edges from the circuit and reconnects the two paths created. In our implementation, the 2-opt operator selects two jobs π_i and π_j from the current sequence, then constructs a new sequence which deletes the connection between π_i and its successor π_{i+1} and the connection between π_j with its successor π_{j+1}, and then connects π_i with π_j and π_{i+1} with π_{j+1}. Furthermore, the partial sequence between π_{i+1} and π_{j+1} is reversed. The complexity of this neighborhood structure is $\Theta(n^2)$.

These neighborhood structures have been widely used in literature to solve different combinatorial optimization problems specially for those which are represented by permutations [14].

4.2. Initial solution

Definition: We first recall the definition of the WSPT rule (weighted shortest processing time). WSPT is the heuristic that schedules jobs in increasing order according to the ratio p_i/w_i. It was introduced by Smith in 1956 [32]. Smith also proved that the WSPT rule solves optimally the scheduling problem $1/\sum_{i=1}^{n} w_i c_i$.

Since GVNS is a trajectory-based metaheuristic, we need to start from a given solution. Due to the impact of the WSPT on the quality of the solution, we designed the initial solution taking into account this rule as a first step, before applying a local search based on swapping two jobs. The choice of this neighborhood structure is motivated by preliminary tests presented in Section 5.

Algorithm 1. Initial Solution.

```
**Data:** $Max\_Iter$
**Result:** $\pi_{best}$
$\pi \leftarrow$ rank the jobs according to the WSPT rule
$\pi_{best} \leftarrow \pi$
$i \leftarrow 1$
**while** $i \leq Max\_Iter$ **do**
    $\pi'' \leftarrow argmin_{y \in \mathscr{N}_2(\pi)} f(y)$
    **if** $\pi''$ *is better than* $\pi_{best}$ **then**
        $\pi_{best} \leftarrow \pi''$
    **end**
    $i++$
    $\pi \leftarrow \pi''$
**end**
```

The proposed local search is defined as follows: In each iteration i, we search for the best solution in the neighborhood $\mathscr{N}_2(\pi)$ of the best solution found in iteration $i-1$. Max_Iter is the maximum number of iterations. $\text{argmin}_{y \in \mathscr{N}_2(\pi)} f(y)$ is the value of $y \in \mathscr{N}_2(\pi)$ for which $f(y)$ attains its minimum.

4.3. Shaking procedure

Our shaking procedure consists of randomly generating a neighboring solution π' from the current solution π using the 2-opt operator. Some random jumps in the search space may be used to escape from a local optimum due to the path reversion, and contribute to diversify the search. Preliminary tests show us that, for our problem, the shaking procedure with more operators reduces the quality of the results. In order to tune the strength of the diversification induced by the 2-opt operator, the shaking procedure is repeated $L_{\max}$ times. This parameter is fixed on the basis of primarily tests presented in Section 5.

4.4. VND procedure

A complete local search is designed as a VND for each solution returned by the shaking procedure. In this work, we fixed the number of neighborhoods to three ($k_{\max} = 3$), as it is advised in the relating literature [12]. To efficiently explore possible solutions it is important to establish the "best" order of the three neighborhood structures defined above. Due to this, we performed a preliminary test. Six different orderings ($a_{i,j}$) of the three neighborhood structures are listed in Table 1.

Table 1. Neighborhood structure test cases.

$a_{1,1}$	$a_{1,2}$	$a_{1,3}$
Swap	Swap	Insertion
Insertion	2-Opt	Swap
2-Opt	Insertion	2-Opt
$a_{2,1}$	$a_{2,2}$	$a_{2,3}$
Insertion	2-Opt	2-Opt
2-Opt	Insertion	Swap
Swap	Swap	Insertion

The proposed VND operates a changing neighborhood strategy, which consists to explore a complete neighborhood using the same operator as long as there is an improvement, otherwise another operator is used. The pseudo code of VND and GVNS proposed in this work are presented as Algorithms 2 and 3.

5. Computational results

Since there is no publicly available benchmark instances for the problem under study in the related literature, we decided to carry out experiments with randomly generated instances. The processing times p_i and the weights w_i are respectively uniformly distributed over $[20, 100]$ and $[1, n]$. The period of maintenance is generated as follows:

$$T = \max\left(\max_{1\le i\le n}(p_i), 4 * \left(\sum_{i=1}^{n} p_i/n\right)\right). \tag{5.1}$$

To ensure feasibility of the generated instances, T is generated in a way that makes it greater than the largest processing time p_i. Also, to avoid solutions with only one job per batch, we choose T greater than 4 times the average of the processing times. To get instances with different levels of tightness regarding the duration of maintenance δ, this latter is set to $\lfloor \frac{2\cdot T}{100} + 0.5\rfloor$ which is denoted by $\delta = 2\%T$.

Both the GVNS and the lower bounds are coded in JAVA language. All the experiments described in this section have been carried out on a i7 Intel Core at 2.50 GHz and Linux with 16 GB RAM. The experiments are

Algorithm 2. GVNS.

```
Data: $L_{max}$, $k_{max}$, $Max\_Iter$
Result: $\pi_{best}$
$l \leftarrow 1$
$\pi_{best} \leftarrow Initial\_Solution(Max\_Iter)$
while ($l \leq L_{max}$) do
    improve ← true
    $\pi' \leftarrow$ shaking($\pi_{best}$)
    if $\pi'$ is better than $\pi_{best}$ then
        $\pi_{best} \leftarrow \pi'$
    end
    while improve do
        $\pi'' \leftarrow$ VND($\pi', k_{max}$)
        if $\pi''$ is better than $\pi_{best}$ then
            $\pi' \leftarrow \pi''$
            $\pi_{best} \leftarrow \pi''$
        end
        else
            improve ←false
        end
    end
    $l$++
end
```

Algorithm 3. VND.

```
Data: $\pi', k_{max}$
Result: $\pi''$
$k \leftarrow 1$
$\pi'' \leftarrow \pi'$
while $k \leq k_{max}$ do
    $\pi' \leftarrow argmin_{y \in \mathscr{N}_k(\pi'')} f(y)$
    if $\pi'$ is better than $\pi''$ then
        $\pi'' \leftarrow \pi'$
    end
    else
        $k++$
    end
end
```

conduced using 54 different problem sizes, namely $n = 2, 3, 4, \ldots, 50, 60, 70, 80, 90$ and 100. Fifteen instances are generated for each instance-size n.

Computational results were conduced using MILP presented in [21]. In order to analyze the performances of GVNS and lower bounds, we include in this paper the obtained results from $n = 2$ to 20 for the MILP using CPLEX solver. Due to the NP-hardness of the considered problem, it takes at least 20 min to obtain an optimal solution for each instance with size greater than 14.

Note that, in all the tables presented in this section and for each instance size, we report the average value of fifteen instances. All the tests (except for the MILP) presented in this work are done for instances of all sizes $n = 2, 3, 4, \ldots, 50, 60, 70, 80, 90$ and 100. However, to save space, we don't present each time the results of all instance sizes.

5.1. Testing local search procedures

5.1.1. *Local searches in the proposed initial solution*

In this section, we propose to compare local searches based on the predefined five neighborhood structures (*i.e.* $\mathscr{N}_1$, $\mathscr{N}_2$, $\mathscr{N}_3$, $\mathscr{N}_4$, $\mathscr{N}_5$) in order to determine the best one to use within the Algorithm 1 proposed to generate the initial solution. Each local search is executed with $L_{\max}$ set to 30. For each operator, we start each time from the same solution which is returned by the WSPT rule. The summarized results are reported in Table 2. MinV gives the minimum value among the five results ($\text{Value}_{\mathscr{N}_i}$, $i \in \{1, \ldots, 5\}$) returned by each tested local search for each instance of size n.

$$\text{MinV} = \min\{\text{Value}_{\mathscr{N}_1}, \text{Value}_{\mathscr{N}_2}, \text{Value}_{\mathscr{N}_3}, \text{Value}_{\mathscr{N}_4}, \text{Value}_{\mathscr{N}_5}\}. \tag{5.2}$$

The gap reported in columns 2, 3, 4 and 5 is calculated as follows:

$$100 * ((\text{Value}_{\mathscr{N}_i} - \text{MinV})/\text{Value}_{\mathscr{N}_i}). \tag{5.3}$$

While comparing the results in Table 2, we observe that local search based on $\mathscr{N}_2$ (*Swap* operator), significantly outperforms all the other local searches as it obtains the minimum value at almost every instance. Otherwise, local search based on $\mathscr{N}_4$ (*Restart* operator) provides the worst values. To summarize, the Neighborhoods are ranked as follows in increasing order with respect to their efficiency: $\mathscr{N}_4$, $\mathscr{N}_1$, $\mathscr{N}_3$, $\mathscr{N}_5$ and $\mathscr{N}_2$. This result justifies our choice to use the *Swap* operator in the local search proposed within the initial solution algorithm.

5.1.2. *Neighborhoods order within VND*

It is important to establish the "best" order of the neighborhood structures since it affects the efficiency of the GVNS algorithm. To do this, we performed a set of preliminary experiments. The different orderings are listed in Table 1 presented in Section 4. Focusing on results observed in the last paragraph we decided to implement the $\mathscr{N}_5$, $\mathscr{N}_3$, $\mathscr{N}_2$ complete neighborhood structures based respectively on the following operators: 2-*Opt*, *Insertion* and *Swap*. The summarized results are reported in Table 3. To test the performances of each case order a_{ij} presented in Table 1, we compared its results with minV (Eq. (5.4)) for each instance size, namely $n = 10, 15, 20, 25, 30, 35, 40, 45, 50, 60, 70, 80, 90, 100$.

$$\text{minV} = \min\{\text{Value}_{a_{1,1}}, \text{Value}_{a_{1,2}}, \text{Value}_{a_{1,3}}, \text{Value}_{a_{2,1}}, \text{Value}_{a_{2,2}}, \text{Value}_{a_{2,3}}\}. \tag{5.4}$$

The formulation below is used for the comparison:

$$\text{gap}_{a_{i,j}} = 100 * ((\text{Value}_{a_{i,j}} - \text{minV})/\text{Value}_{a_{i,j}}). \tag{5.5}$$

TABLE 2. Comparison of different local search procedures for the generated instances.

n	$\text{gap}_{\mathscr{N}_1}$	$\text{gap}_{\mathscr{N}_2}$	$\text{gap}_{\mathscr{N}_3}$	$\text{gap}_{\mathscr{N}_4}$	$\text{gap}_{\mathscr{N}_5}$
10	1.01	0.00	0.10	3.04	0.06
15	2.37	0.01	0.00	6.69	0.02
20	3.92	0.00	0.71	6.58	0.26
25	4.89	0.00	1.29	8.81	0.28
30	4.78	0.00	1.11	9.12	0.45
35	2.99	0.00	0.57	9.19	0.10
40	5.70	0.00	1.02	9.83	0.40
45	6.04	0.00	0.80	10.03	0.18
50	4.79	0.00	0.62	9.48	0.05

TABLE 3. Neighborhood structures test cases.

n	$\text{gap}_{a_{1,1}}$	$\text{gap}_{a_{1,2}}$	$\text{gap}_{a_{1,3}}$	$\text{gap}_{a_{2,1}}$	$\text{gap}_{a_{2,2}}$	$\text{gap}_{a_{2,3}}$
10	0.16	0.16	0.20	0.00	0.23	0.23
15	0.23	0.19	0.00	0.00	0.06	0.23
20	0.26	0.12	0.08	0.02	0.00	0.06
25	0.00	0.27	0.49	0.34	0.44	0.34
30	0.27	0.37	0.25	0.14	0.00	0.13
35	0.18	0.07	0.00	0.15	0.17	0.04
40	0.02	0.00	0.24	0.02	0.39	0.05
45	0.03	0.00	0.05	0.09	0.08	0.13
50	0.00	0.17	0.17	0.34	0.14	0.25
60	0.04	0.12	0.11	0.10	0.42	0.12
70	0.20	0.20	0.12	0.12	0.38	0.14
80	0.18	0.10	0.23	0.13	0.47	0.09
90	0.04	0.17	0.19	0.13	0.11	0.00
100	0.04	0.17	0.19	0.13	0.11	0.00
Average	0.11	0.15	0.16	0.13	0.21	0.14

The entries $\text{Value}_{a_{i,j}}$ represent the value returned by the proposed VND.

As seen in Table 3, the six tests are very close to each others. It follows that, for our problem, VND is not very sensitive to the exploration order of the neighborhood structures. In spite of this, the results show that the order $a_{1,1}$ (*swap*, *insertion*, 2-*Opt*) is better than the other orders, since it returns the minimum value or a very tight value from the minimum for the large instances. In our proposed GVNS we suggest to use this order of neighborhoods in the VND heuristic.

5.2. Parameters calibration of GVNS metaheuristic

In this subsection, we examined the influence of the parameters Max_Iter in the initial solution and $L_{\max}$ in the GVNS. To do this, we tested different values of Max_Iter and $L_{\max}$. The testing is performed by varying their values, Max_Iter from 1 to 180, and $L_{\max}$ from 5 to 80. Note that when Max_Iter = 0, it represents the value returned by the WSPT rule.

$L_{\max}$ can be understood as the amount of time that the shaking procedure is used within the proposed GVNS. The obtained results are presented in Tables 4 and 5 for fifteen instances of size $n = 100$. The large size of these instances could reveal the performance of the initial solution for a given value of Max_Iter and $L_{\max}$. For each value of Max_Iter and $L_{\max}$ the average solution value found and also the average time spent upon reaching these solutions are presented. As GVNS is a stochastic heuristic, we run this latter 10 times for each instance. Since the coefficient of variation (*i.e.* the relative standard deviation) is smaller than 0.1% we report only the best of these 10 replicates.

From the results presented in Tables 4 and 5, it follows that the proposed initial solution algorithm is very sensitive to the value of Max_Iter parameter. Indeed, it turns out that the initial solution returns the best solution values when the parameter Max_Iter is set to 140, consuming the least amount of time (0.83 s). Same conclusion can be drawn for $L_{\max}$ parameter, as the results returned after $L_{\max} = 40$ seem to be very close. Consequently, for the rest of the testing, Max_Iter is set to 140 and $L_{\max}$ is set to 40.

5.3. Lower bounds performances

In this section, we propose to compare the computational results between the three proposed lower bounds based on job splitting and the linear bound, LB_{RP}, obtained by relaxing the integrity constraints for $x_{i,j}$ in

TABLE 4. Initial solution with different values of Max_Iter.

Max_Iter	Aver.Value	Aver.Time(s)
0	9,858,331.13	0.0186
10	9,451,353	0.077
20	9,349,855	0.139
30	9,304,307.46	0.2
40	9,278,048.93	0.26
50	9,259,602	0.3
60	9,246,078	0.37
80	9,233,504	0.48
100	9,229,189	0.596
120	9,224,482	0.71
140	9,223,864.60	0.83
160	9,223,864.60	0.94
180	9,223,864.60	1.06

TABLE 5. Initial solution with different values of $L_{\max}$.

$L_{\max}$	Aver.Value	Aver.Time(s)
5	9,204,698	0.99
10	9,204,119.26	1.07
20	9,201,641.53	1.23
30	9,200,712.33	1.37
40	9,196,670.20	1.55
60	9,196,668.10	1.81
80	9,196,666.50	2.06

the mixed integer programming model. Note that relaxing the integrity constraints for y_{ij} is not reported here because it appears to be useless (*i.e.* no significant decrease in terms of computational time) consuming similarly relaxing $x_{i,j}$ and $y_{i,j}$ at the same time does not provide a good lower bound quality. Table 6 shows the performances of each lower bound obtained by calculating the Ratio between the lower bound LB$_i$ and the optimal value of the MILP as follows:

$$\text{Ratio}_{\text{LB}_i} = 100 * (\text{Value}_{\text{LB}_i}/\text{Value}_{\text{MILP}}). \tag{5.6}$$

Entries in the last column represent the results of the tightest value returned among the four lower bounds.

$$\text{LB}_{\text{best}} = \max\{\text{LB1}, \text{LB2}, \text{LB3}, \text{LB}_{\text{RP}}\}. \tag{5.7}$$

The calculated ratio clearly shows that LB3 is much tighter than LB1 and LB2. Actually the larger the size of the instances, the greater the gap between the lower bounds (LB1, LB2, LB3) and the optimum value. The lower bound LB$_{\text{RP}}$ seems to be better than the others when the size of instances is greater than twelve ($n > 12$).

5.4. GVNS performances

In this section, experiments are carried out to evaluate the performance of the proposed GVNS. Due to the stochastic aspect, we run the GVNS algorithm 10 times. As we already stated in Section 5.2, the coefficient of variation is smaller than 0.1%. Consequently only report the best of the 10 replicates. The results of comparisons between the MILP and GVNS, and also between GVNS and LB$_{\text{best}}$ are presented in Table 7. The entries in

TABLE 6. Comparison of lower bounds.

n	$\text{Ratio}_{\text{LB1}}$	$\text{Ratio}_{\text{LB2}}$	$\text{Ratio}_{\text{LB3}}$	$\text{Ratio}_{\text{LB}_{\text{RP}}}$	$\text{Ratio}_{\text{LB}^*}$
2	92.06	81.84	99.33	72.53	99.33
3	73.67	75.28	91.04	59.45	91.04
4	54.65	62.14	91.81	72.77	91.81
5	50.50	62.64	80.12	66.33	80.12
6	39.93	54.03	62.45	69.38	69.38
7	36.71	52.84	96.34	69.57	96.34
8	34.96	50.26	88.16	74.03	88.16
9	32.11	48.19	92.76	74.56	92.76
10	29.18	43.97	98.25	74.33	98.25
11	29.50	46.85	86.69	75.27	86.69
12	27.71	44.34	79.16	77.76	79.16
13	27.42	45.63	76.47	77.85	77.85
14	28.41	47.63	69.41	79.36	79.36
15	27.17	45.64	69.32	80.25	80.25
16	24.93	42.80	66.67	81.20	81.20
17	24.30	42.08	60.05	81.52	81.52
18	24.81	43.50	62.94	82.72	82.72
19	22.60	40.13	54.61	83.30	83.30
20	23.38	41.26	48.01	84.27	84.27

columns three and five represent the ratio between the optimal value returned by the MILP ($\text{Value}_{\text{MILP}}$) and the objective function value returned by the GVNS ($\text{Value}_{\text{GVNS}}$) respectively, between the lower bound ($\text{Value}_{\text{LB}_{\text{best}}}$) and ($\text{Value}_{\text{GVNS}}$). It is calculated as follows:

$$\text{Ratio}_{\text{MH}} = 100 * (\text{Value}_{\text{MILP}}/\text{Value}_{\text{GVNS}}). \tag{5.8}$$

respectively,

$$\text{Ratio}_{\text{LH}} = 100 * (\text{Value}_{\text{LB}_{\text{best}}}/\text{Value}_{\text{GVNS}}). \tag{5.9}$$

The three last columns report the time consumed upon reaching the reported values returned respectively by the MILP, the GVNS and the lower bounds. It follows that GVNS is very fast (an average of 4.22 s for instances with size 100). For instances of size $n \leq 20$, the GVNS is very efficient in comparison with the MILP. The ratio Ratio_{LH} becomes smaller for instance sizes ranging from $n = 11$ to $n = 21$. This is certainly due to the lower bounds performances. Indeed, the gap is practically rather large for these instance sizes (see Tab. 6). This shows that GVNS is able to reach high quality solutions in a very small amount of time.

The relaxed problem is easier to solve in the case of small instances than in the case of large instances. On top of this, the computation time in both cases is limited to 20 min, so it is more likely that, for large instances, the relaxed MILP is struggling to improve (*i.e.* to minimize) the solution. Hence the returned value of the relaxed MILP is still quite large and relatively close to the solution returned by GVNS. More precisely, when $n > 14$, the best lower bound LB_{best} is equal to the lower bound LB_{RP} based on the relaxation of the integrity constraints of the MILP model. Indeed, via the relaxed MILP, CPLEX solves optimally the problem within the calculation time limit of 20 min for instances with $n < 50$. When $n \geq 50$, returned solutions are not necessarily optimal and require more than 20 min to get the optimal solutions. Therefore, the value returned by relaxed MILP is closer to the GVNS value than to the optimal solution of the relaxed MILP (LB_{RP}). Empirical experiments for those instances show that the gap returned by CPLEX solver remains 90% (in average for the 15 instances) after 20 min. This explains why the calculated gap (Col. 5 in Tab. 7) is so high for the large instances.

TABLE 7. General Variable Neighborhood Search performances.

n	Av.MILP	Av.GVNS	Ratio_{MH}(%)	$\text{Av.LB}_{\text{best}}$	Ratio_{LH}(%)	$\text{Av.Time}_{\text{MILP}}$(s)	$\text{Av.Time}_{\text{GVNS}}$(s)	$\text{Av.Time}_{\text{LB}_{\text{best}}}$(s)
2	257.73	257.73	100	256.00	99.33	0.01	≈0	≈0
3	574.47	574.47	100	523.00	91.04	0.01	≈0	≈0
4	1,221.00	1,221.00	100	1,121.00	91.81	0.02	≈0	≈0
5	1,899.40	1,899.40	100	1,521.80	80.12	0.03	≈0	≈0
6	3,661.73	3,661.73	100	2,540.53	69.38	0.04	≈0	≈0.01
7	5,248.80	5,248.80	100	5,056.93	96.34	0.07	≈0	≈0.01
8	7,046.73	7,046.73	100	6,212.07	88.16	0.12	≈0	≈0.03
9	9,974.47	9,986.87	99.88	9,252.13	92.64	0.55	≈0	0.05
10	1,1708.60	1,170,8.60	100	1,180,3.73	98.81	1.21	≈0	0.08
11	1,7603.67	1,760,3.67	100	1,526,0.33	86.69	7.80	≈0.02	0.13
12	2,1100.07	2,113,5.33	99.83	1,670,2.47	79.03	57.97	≈0.02	0.23
13	2,3742.47	2,375,9.20	99.93	1,848,4.53	77.80	120.79	≈0.01	0.41
14	3,0907.40	3,092,3.87	99.95	2,452,7.33	79.32	322.86	≈0.02	0.55
15	4,2924.40	4,297,6.26	99.88	3,444,5.20	80.15	754.32	≈0.02	0.63
16	4,6450.27	4,650,2.26	99.89	3,771,9.93	81.11	886.72	≈0.02	0.85
17	5,5079.13	5,515,8.53	99.86	4,489,7.80	81.40	1067.49	≈0.02	1.02
18	6,7864.67	6,794,3.93	99.88	5,613,8.33	82.62	≥1200	≈0.02	1.16
19	7,8707.67	7,879,9.60	99.88	6,556,1.20	83.20	≥1200	0.03	1.33
20	9,0541.67	9,071,7.13	99.81	7,629,6.00	84.10	≥1200	0.03	1.65
21	//	9,510,3.00	//	7,952,5.47	83.62	≥1200	0.04	1.83
22	//	1,197,83.33	//	1,017,04.33	84.91	≥1200	0.04	2.12
23	//	1,497,93.07	//	1,276,08.47	85.19	≥1200	0.05	2.52
24	//	1,477,42.40	//	1,264,70.00	85.60	≥1200	0.05	3.09
25	//	1,853,65.87	//	1,595,33.80	86.06	≥1200	0.06	3.56
26	//	1,896,95.40	//	1,635,48.13	86.22	≥1200	0.07	4.21
27	//	2,184,26.93	//	1,888,32.80	86.45	≥1200	0.08	5.34
28	//	2,421,23.53	//	2,102,93.80	86.85	≥1200	0.08	5.88
29	//	2,695,41.13	//	2,358,38.40	87.50	≥1200	0.09	6.78
30	//	3,009,39.27	//	2,630,57.20	87.41	≥1200	0.1	7.78
31	//	2,957,34.73	//	2,590,63.87	87.60	≥1200	0.12	8.73
32	//	3,551,61.20	//	3,129,55.47	88.12	≥1200	0.13	9.95
33	//	3,943,04.53	//	3,489,51.80	88.50	≥1200	0.15	12.68
34	//	4,147,89.80	//	3,677,58.13	88.66	≥1200	0.18	12.60
35	//	4,312,14.20	//	3,839,17.60	89.03	≥1200	0.2	14.71
36	//	4,875,72.47	//	4,339,79.93	89.01	≥1200	0.22	19.78
37	//	4,929,42.33	//	4,382,08.53	88.90	≥1200	0.24	23.16
38	//	5,681,53.67	//	5,092,73.27	89.64	≥1200	0.26	30.22
39	//	5,903,33.00	//	5,281,01.87	89.46	≥1200	0.27	27.38
40	//	6,609,70.73	//	5,934,30.73	89.78	≥1200	0.29	36.65
41	//	7,130,95.87	//	6,427,51.47	90.14	≥1200	0.3	39.21
42	//	6,961,91.33	//	6,253,14.80	89.82	≥1200	0.32	52.35
43	//	8,129,31.20	//	7,340,15.33	90.29	≥1200	0.35	113.43
44	//	8,452,37.93	//	7,663,81.53	90.67	≥1200	0.37	93.85
45	//	9,413,81.20	//	8,516,57.00	90.47	≥1200	0.39	176.18
46	//	9,653,14.73	//	8,771,82.73	90.87	≥1200	0.42	372.50
47	//	9,948,41.27	//	9,046,61.20	90.94	≥1200	0.44	387.30
48	//	1,099,902.07	//	1,000,075.13	90.92	≥1200	0.48	456.32
49	//	1,173,683.47	//	1,075,626.87	91.65	≥1200	0.5	1,138.48
50	//	1,257,111.40	//	1,171,765.73	93.21	≥1200	0.52	≥1200
60	//	2,003,433.267	//	2,087,964.20	93.21	≥1200	0.65	≥1200
70	//	3,198,406.60	//	3,353,236.80	95.38	≥1200	0.74	≥1200
80	//	4,856,950.60	//	4,868,220.86	99.76	≥1200	1.07	≥1200
90	//	7,173,766.92	//	7,373,766.92	99.86	≥1200	0.94	≥1200
100	//	9,196,666.50	//	9,187,652.35	95.95	≥1200	0.87	≥1200

6. CONCLUSION

In this paper we studied the single machine scheduling problem with periodic maintenance. The objective of our study is to minimize the sum of weighted completion times. We proposed a GVNS algorithm to solve this NP-hard problem. The proposed algorithm was tested and compared with a MILP model for small instance sizes, and with four lower bounds for instance sizes greater than 20, three of them based on job splitting and one based on the relaxation of one integrity constraint. The computational results show that the proposed GVNS is very efficient both in terms of quality of the objective function values and computational time. In future work, it will be interesting to propose more lower bounds and generate more test instances. It would be interesting to compare the performance of GVNS with other metaheuristics for solving the considered problem after generalizing it by adding more realistic constraints such as setup times, release times, failures, etc. Another extension of this research would be to consider the interval between two consecutive maintenance activities as a decision variable of the problem.

Acknowledgements. The present research work has been carried out in the ELSAT 2020 project. The ELSAT 2020 project is co-financed by the European Union with the European Regional Development Fund, the French State and the Hauts de France Region Council. The authors gratefully acknowledge the support of these institutions. The authors would like to thank the anonymous referees for their helpful suggestions which have improved the presentation of this paper.

References

[1] F. Ángel-Bello, A. Álvarez, J. Pacheco and I. Martínez, A single machine scheduling problem with availability constraints and sequence-dependent setup costs. *Appl. Math. Model.* **35** (2011) 2041–2050.

[2] A. Artiba, F. Riane, M.A. Jamali, D. Ait-Kadi and R. Cléroux, Joint optimal periodic and conditional maintenance strategy. *J. Qual. Main. Eng.* **11** (2005) 107–114.

[3] S. Ashour, Sequencing Theory. Vol. 69. Springer Science & Business Media. Springer Berlin, Heidelberg, New York (2012).

[4] H. Belouadah, M.E. Posner and C.N. Potts, Scheduling with release dates on a single machine to minimize total weighted completion time. *Discrete Appl. Math.* **36** (1992) 213–231.

[5] M. Ben-Daya, D. Ait-Kadi, S.O. Duffuaa, J. Knezevic and A. Raouf, Handbook of maintenance management and engineering. Vol. 7. Springer, Springer Dordrecht Heidelberg London New York (2009).

[6] 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.

[7] W.-J. Chen, Minimizing total flow time in the single-machine scheduling problem with periodic maintenance. *J. Oper. Res. Soc.* **57** (2006) 410–415.

[8] W.-J. Chen, An efficient algorithm for scheduling jobs on a machine with periodic maintenance. *Int. J. Adv. Manuf. Technol.* **34** (2007) 1173–1182.

[9] R. Cruzan, Manager's Guide to Preventive Building Maintenance. The Fairmont Press, INC. 711 Indian trail Lilburn, GA 300047 (1970).

[10] W.-W. Cui and Z. Lu, Minimizing the makespan on a single machine with flexible maintenances and jobs release dates. *Comput. Oper. Res.* **80** (2017) 11–22.

[11] A. Ebrahimy Zade and M. Bagher Fakhrzad, A dynamic genetic algorithm for solving a single machine scheduling problem with periodic maintenance. *ISRN Ind. Eng.* **2013** (2013) 11.

[12] F.W. Glover and G.A. Kochenberger, Handbook of Metaheuristics. Vol. 57. Springer Science & Business Media. Kluwer Academic Publishers. New York, Boston, Dordrecht, London, Moscow (2006).

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

[14] P. Guo, W. Chen and Y. Wang, A general variable neighborhood search for single-machine total tardiness scheduling problem with step-deteriorating jobs. Preprint arXiv: 1301.7134 (2013).

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

[16] P. Hansen, N. Mladenović and D. Perez-Britos, Variable neighborhood decomposition search. *J. Heuristics* **7** (2001) 335–350.

[17] K. Helsgaun, *An effective implementation of K-opt moves for the Lin-Kernighan TSP heuristic*. Ph.D. thesis, Roskilde University, Department of Computer Science (2006).

[18] A. Ilić, D. Urošević, J. Brimberg and N. Mladenović, A general variable neighborhood search for solving the uncapacitated single allocation p-hub median problem. *Eur. J. Oper. Res.* **206** (2010) 289–300.

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

[20] 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.

[21] H. Krim, R. Benmansour and D. Duvivier, Minimizing the weighted completion time on a single machine with periodic maintenance. In: ROADEF 2016, Compiégne, France (Februrary 2016).

[22] C.-Y. Lee and S.D. Liman, Single machine flow-time scheduling with scheduled maintenance. *Acta Info.* **29** (1992) 375–382.

[23] J.-Y. Lee and Y.-D. Kim, Minimizing the number of tardy jobs in a single-machine scheduling problem with periodic maintenance. *Comput. Oper. Res.* **39** (2012) 2196–2205.

[24] H. Lei, G. Laporte and B. Guo, A generalized variable neighborhood search heuristic for the capacitated vehicle routing problem with stochastic service times. *TOP* **20** (2012) 99–118.

[25] C.-J. Liao and W.-J. Chen, Single-machine scheduling with periodic maintenance and nonresumable jobs. *Comput. Oper. Res.* **30** (2003) 1335–1347.

[26] C. Low, C.-J. Hsu and C.-T. Su, A modified particle swarm optimization algorithm for a single-machine scheduling problem with periodic maintenance. *Expert Syst. Appl.* **37** (2010) 6429–6434.

[27] W. Luo and F. Liu, On single-machine scheduling with workload-dependent maintenance duration. *Omega* **68** (2017) 119–122.

[28] A. Mjirda, R. Todosijevic, S. Hanafi, P. Hansen and N. Mladenovic, Sequential variable neighborhood descent variants: an empirical study on the traveling salesman problem. *ITOR* **24** (2017) 615–633.

[29] N. Mladenović and P. Hansen, Variable neighborhood search. *Comput. Oper. Res.* **24** (1997) 1097–1100.

[30] M.E. Posner, Minimizing weighted completion times with deadlines. *Oper. Res.* **33** (1985) 562–574.

[31] O. Roux, D. Duvivier, G. Quesnel and E. Ramat, Optimization of preventive maintenance through a combined maintenance-production simulation model. *Int. J. Prod. Econ.* **143** (2013) 3–12.

[32] W.E. Smith, Various optimizers for single-stage production. *Nav. Res. Log. Q.* **3** (1956) 59–66.

[33] L.-H. Su and H.-M. Wang, Minimizing total absolute deviation of job completion times on a single machine with cleaning activities. *Comput. Ind. Eng.* **103** (2017) 242–249.

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

[35] M. Yazdani, A. Aleti, S.M. Khalili and F. Jolai, Optimizing the sum of maximum earliness and tardiness of the job shop scheduling problem. *Comput. Ind. Eng.* **107** (2017) 12–24.