CIRRELT

CIRRELT-2020-02

Tabu Search for a Parallel-Machine Scheduling Problem with Job Rejection, Inventory Penalties and Periodic Maintenance

Hanane Krim
Nicolas Zufferey
Jean-Yves Potvin
Rachid Benmansour
David Duvivier

January 2020

Tabu Search for a Parallel-Machine Scheduling Problem with Job Rejection, Inventory Penalties and Periodic Maintenance

Hanane Krim[1], Nicolas Zufferey[2,3], Jean-Yves Potvin[3,4], Rachid Benmansour[1,5], David Duvivier[1]

1. Laboratoire d'Automatique, de Mécanique et d'Informatique industrielles et Humaines (LAMIH) UMR CNRS 8201, Université Polytechnique Hauts-de-France, 59313 Valenciennes Cedex 9, France
2. Geneva School of Economics and Management,Université de Genève, 1211 Genève 4, Suisse
3. Interuniversity Research Centre on Enterprise Networks, Logistics and Transportation (CIRRELT)
4. Département d'informatique et de recherche opérationnelle, Université de Montréal, C.P. 6128, succ. Centre-Ville, Montréal, Québec, Canada H3C 3J7
5. Institut National de Statistique et d'Économie Appliquée (INSEA), Laboratoire SI2M, Rabat, Maroc

Abstract. We consider a bicriteria scheduling problem on two parallel, non identical machines with a periodic preventive maintenance policy. The two objectives considered involve minimization of job rejection costs and weighted sum of completion times. They are handled through a lexicographic approach, due to a natural hierarchy among the two objectives in the applications considered. The contributions of this paper are first to develop a mixed integer linear program model for the problem and, second, to introduce two new metaheuristics based on tabu search. Computational results on test instances of different sizes are reported to empirically demonstrate the effectiveness of the proposed metaheuristics.

Keywords. Parallel machine scheduling, job rejection, periodic maintenance, lexicographic optimization, tabu search.

Acknowledgements. This research work has been carried out under the ELSAT 2020 project supported 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 present research work has also been carried out in the context of the LIA (International Associate Laboratory) ROI-TML (Operational Research and Computer Science in Transportation, Mobility and Logistics) between LAMIH UMR 8201 (France) and CIRRELT (Canada). The authors gratefully acknowledge the support of these laboratories and related institutions.

---

* Corresponding author: Jean-Yves.Potvin@cirrelt.ca

Dépôt légal – Bibliothèque et Archives nationales du Québec
Bibliothèque et Archives Canada, 2020

1 Introduction

Scheduling problems have been extensively studied in the literature under the assumption that all jobs have to be processed. However, in many practical cases, one may wish or may be forced to postpone the processing of some jobs, although at some cost. Accordingly, a decision has to be made about jobs that will be accepted and those that will be rejected to produce a good schedule. Nowadays, this situation is observed in several companies with a weekly planning (e.g., pharmaceutical products, luxury watches, fast moving consumer goods). Typically, rejected jobs will get a larger weight or priority the next week. At the same time, the parallel-machine scheduling problem has been extensively studied due to its practical applications in various manufacturing systems such as printed circuit board manufacturing, group technology cells, injection molding processes, etc. However, few studies have been done in the context of parallel-machine scheduling with job rejection.

Maintenance is another aspect closely connected to production scheduling in real manufacturing settings. One of the most common assumptions in the scheduling literature is that the machines or processors are always available but, in practice, they may have to be stopped due to failures or preventive maintenance. In particular, the importance of preventive maintenance (PM) has been gradually recognized by decision makers as a mean to avoid machine failures. Preventive maintenance is performed when the machines are idle and, consequently, represents a source of machine unavailability. Tradeoffs to be found between preventive maintenance and production activities have led researchers to investigate different ways of jointly scheduling both activities. Production is expected to be more efficient and revenues to increase when preventive maintenance is well managed.

In this regard, we address a scheduling problem (P) with two parallel and non-identical machines (it is formally a *2-Parallel Machines problem with Periodic Maintenance, Job Rejection and Weighted sum of Completion Times*). In this problem, the two machines must undergo periodic preventive maintenance over the scheduling horizon. Solution quality is measured with two criteria. The first one is the total cost of rejected jobs and the second one is the weighted sum of job completion times. In the latter case, the weights can stand for the holding or inventory cost of the corresponding jobs as well as their priority level (importance, urgency). A strategy based on Lexicographic Optimization (LO) is proposed to deal with this multi-objective problem. In LO, the decision maker establishes beforehand a priority order among the optimization objectives, where each higher-level objective is infinitely more important than any lower-level objective. LO is a convenient approach to address multiobjective problems in practice, as reported in (Zykina, 2004; Ehrgott, 2005; Thevenin et al., 2017b; Prats et al., 2010; Solnon et al., 2008; T'kindt and Billaut, 2006).

The remainder of this paper is organized as follows. A literature review dealing with order acceptance and scheduling, job rejection, periodic maintenance and multi-availability constraints is proposed in Section 2. Next, a Mixed Integer Linear Program (MILP) for problem (P) is presented in Section 3. The greedy constructive heuristic and the two metaheuristics based on tabu search are described in Sections 4 and 5, respectively, whereas Section 6 reports computational results. Finally, Section 7 ends the paper with a conclusion and some perspectives for the future.

2 Literature review

Based on the three-field notation $\alpha \mid \beta \mid \gamma$ known as the Graham triplet (Graham et al., 1979), our problem (P) can be denoted as $P2 \mid pm \mid \sum_{j=1}^{n} u_j, \sum_{j=1}^{n} w_j C_j$. The first field ($\alpha$) means that there are two parallel machines. The second field (β) indicates that a periodic preventive maintenance (pm) must be performed on each machine. Finally, the last field (γ) represents the objective functions (see the notation used in Section 3). To the best of our knowledge, problem (P) has never been studied in the literature. Nonetheless, Subsections 2.1 to 2.4 will review works that are related to this problem. Subsections 2.1 and 2.2 are dedicated to the order acceptance and scheduling literature and to the scheduling problem with job rejection literature, respectively. In both cases, the same problematic issue is addressed, namely job scheduling when the production capacity does not allow all jobs to be scheduled. This situation leads to the rejection (resp. acceptance) of some of them, which is penalized (resp. rewarded) in the objective function. Subsections 2.3 and 2.4 focus on the maintenance and on the lexicographic optimization aspects in the context of job scheduling.

2.1 Order acceptance and scheduling

A taxonomy and a general review on order acceptance and scheduling (OAS) can be found in (Slotnick, 2011). This problem is to jointly decide about job acceptance and the scheduling of accepted jobs. Different problem characteristics and problem-solving methodologies, starting from this basic scheme, have been proposed in the literature. In the following, papers dealing with a single machine and different objective functions are reviewed, followed by a discussion on problems with two or more machines.

2.1.1 Single machine

Oğuz et al. (2010) consider the single-machine scheduling problem where job acceptance depends on the release date, due date, deadline, processing time, sequence-dependent setup time and revenue. The main objective is the maximization of the total revenue. The authors propose a MILP and also develop three heuristic algorithms to solve their problem. Based on the same objective function, Bahriye et al. (2012) propose a tabu search to solve a problem that considers sequence-dependent setup times and tardiness penalties. Nobibon and Leus (2011) generalize two existing problems defined in a single-machine environment, that is, the order acceptance and scheduling problem with weighted-tardiness penalties reported in (Slotnick and Morton, 2007) and the total weighted tardiness scheduling problem reported in (Potts and Van Wassenhove, 1985). The generalized problem reduces to the latter when the pool of firm planned orders is empty and all jobs can potentially be rejected. To solve their generalized problem, the authors propose a MILP and two exact branch-and-bound algorithms. They report solving instances with up to 50 jobs in less than two hours. In (Thevenin et al., 2016), the authors address a production scheduling problem in a single-machine environment with earliness and tardiness penalties, sequence-dependent setup times and costs. The objective function includes setup costs, job rejection penalties and weighted tardiness penalties. The authors propose various methods to solve this problem, ranging from a basic greedy algorithm to sophisticated metaheuristics (e.g., tabu search, adaptive memory algorithm). In another work by the same authors (Thevenin et al., 2015), sequence-dependent setup times and setup costs between jobs of different families, release dates, deadlines and job rejection are taken into account. They propose and compare a constructive heuristic, local search methods, and population-based algorithms. Recent papers dealing with OAS in a single-machine environment take into account machine availability constraints, as in (Zhong et al., 2014). Here, the authors propose a pseudo-polynomial algorithm for fixed time intervals between two consecutive PMs.

2.1.2 Multiple machines

In (Ou and Zhong, 2017), the authors study the OAS problem for n jobs on m parallel machines where the number of rejected jobs should not exceed a given limit L. The objective is to minimize the completion time of the last scheduled job plus the total cost of rejected jobs. For the special case of a single machine, they present an exact algorithm of complexity $O(n \cdot log(n))$. For m machines, they first propose a heuristic of complexity $O(n \cdot log(n))$ with a worst-case bound of $2 - \frac{1}{m}$. They also develop a heuristic based on LP-relaxation and bin-packing techniques. The OAS with two machines in a flow shop is considered in (Wang et al., 2013). The authors present a heuristic and a branch-and-bound algorithm based on dominance rules and relaxation techniques. Their objective is to maximize the total net profit of accepted jobs, where the latter is the revenue minus the weighted tardiness. In (Wang et al., 2015), the authors solve a scheduling problem with two parallel machines with two heuristics and an exact algorithm, using some properties of optimal solutions to maximize the total profit. In another environment with parallel machines, Jiang et al. (2017) study the OAS problem with batch delivery in a supply chain consisting of a manufacturer and a customer. The objective is to minimize the weighted sum of the maximum lead times of accepted jobs and the total delivery cost. To solve the problem, two approximation algorithms are proposed. Finally, Emami et al. (2016) report a MILP model and a Lagrangian relaxation algorithm to solve an OAS problem with the objective of maximizing the total profit.

2.2 Scheduling problem with job rejection

The scheduling problem with job rejection has been studied in different contexts, as indicated in a recent survey (Shabtay et al., 2013) and motivated by industrial applications (Thevenin et al., 2017a), although mostly for single-machine problems.

In (Li and Chen, 2017), the authors consider the scheduling problem with job rejection and a maintenance activity that becomes less effective over time. The main objective is to determine the timing of the maintenance activity and the sequence of accepted jobs to minimize the scheduling cost of accepted jobs plus the total cost of rejected jobs. The authors provide polynomial time algorithms for this problem. Shabtay et al. (2012) propose a bicriteria analysis of a large class of single-machine scheduling problems with a common property, namely, the consideration of rejection costs plus other additional criteria (makespan, sum and variation of completion times, earliness and tardiness costs).

Since scheduling with rejection is mostly studied in bicriteria contexts (Shabtay et al., 2013), concepts from the theory of bicriteria scheduling are commonly used when dealing with such problems. Below, we review papers addressing the weighted sum of completion times and the total cost of rejected jobs. Cao et al. (2006) first prove that the problem for a single machine is NP-hard. A few years later, a pseudo-polynomial algorithm and a Fully Polynomial Time Approximation Scheme (FPTAS) for multiple parallel machines are proposed by Zhang et al. (2009). Engels et al. (2003) also report more general techniques such as linear programming relaxations. In (Moghaddam et al., 2012), the authors study a single-machine scheduling problem with job rejection, while considering again minimization of the weighted sum of completion times plus the total cost of rejected jobs. They propose a mathematical formulation and three different bi-objective simulated annealing algorithms to estimate the Pareto-optimal front for large-size instances. The authors in (Zhong et al., 2017) study a scheduling problem on two parallel machines with release times and job rejection. The objective is to minimize the makespan of accepted jobs plus the total cost of rejected jobs. They develop a $(1.5 + \varepsilon)$-approximation algorithm to solve the problem. Ou et al. (2015) consider m parallel machines in a context where job rejection is allowed. The objective is to minimize the makespan plus the total cost of rejected jobs. They develop a heuristic of complexity $O(n \cdot log(n) + n/\varepsilon)$ to solve the problem with a worst-case bound of $1.5 + \varepsilon$. With the same goal, Zhong and Ou (2017) present a 2-approximation algorithm with a complexity of $O(n \cdot log(n))$ by making use of specific data structures. The authors also propose a PTAS to solve the problem. In (Ma and Yuan, 2016), the authors consider that the information about each job, including processing time, release date, weight and rejection cost, is not known in advance. They develop a technique named Greedy-Interval-Rejection to produce good solutions. Finally, the authors in (Agnetis and Mosheiov, 2017) consider the minimization of the makespan in a flow shop with position-dependent job processing times and job rejection. A polynomial time procedure is proposed to solve this problem.

2.3 Periodic maintenance and multi-availability constraints

The authors in (Kaabi and Harrath, 2014) have written a comprehensive survey about scheduling in parallel-machine environments in the presence of availability constraints (which can be induced, in particular, by maintenance activities). Sun and Li (2010) consider two problems. In the first problem, they minimize the makespan on two parallel machines when maintenance activities are performed periodically. In the second problem, maintenance activities are determined jointly with job scheduling, while minimizing the sum of the job completion times. They introduce an algorithm of complexity $O(n^2)$ and show that the classical Shortest Processing Time algorithm (SPT) is efficient for the second problem with a worst-case bound less than or equal to $1 + 2 \cdot \sigma$, where $\sigma = t/T$, and T is the maximum continuous working time for each machine and t is the time required to perform each maintenance activity. Li et al. (2017) investigate a parallel-machine scheduling problem where each machine must undergo periodic maintenance. The authors propose two mathematical programming models and two heuristic approaches to address instances of large size. In (Qi et al., 2015), the authors investigate a scheduling problem on a single machine with maintenance, in which the starting time of the maintenance is given in advance but its duration depends on the previous machine load.

2.4 Multiobjective scheduling problem using lexicographic optimization

LO is particularly relevant for industrial applications, as highlighted by Gallay and Zufferey (2018). LO is widely used in control engineering and scheduling applications (T'kindt and Billaut, 2006; Aggelogiannaki and Sarimveis, 2006; Kerrigan and Maciejowski, 2002; Ocampo-Martinez et al., 2008; Respen et al., 2016). In a work closely related to ours, the authors in (Thevenin et al., 2017b) model a parallel-machine scheduling problem with job incompatibility through an extension of the graph coloring problem. Different objectives like makespan, number of job preemptions and total time spent by the jobs in the production shop are considered and addressed through LO. A mathematical model, two greedy constructive algorithms, two tabu search methods and an adaptive memory algorithm are proposed to solve the problem.

3 Mathematical model

In the following, we first introduce some notation and a brief description of our problem. This is followed by the MILP.

3.1 Formal description of problem (P)

Let J be a set of n independent jobs to be scheduled on two parallel, non identical machines M_i, $i \in I = \{1, 2\}$, over a planning horizon of five days (i.e., 7200 minutes). Accordingly, we define $\tilde{d} = 7200$ minutes as the common deadline for all jobs in set J. If a job cannot be feasibly scheduled during the current week, it is then postponed to the next week and a rejection cost is incurred. A feasible solution S of problem (P) is illustrated in Figure 1. It is made of two schedules on machines M_1 and M_2, with the corresponding sets J_S and $\overline{J}_S$ of accepted and rejected jobs, respectively. Each machine M_i must undergo a PM at intervals that cannot exceed T_i minutes. In other words, the interval between the end time of a given PM and the start time of the next PM cannot exceed T_i minutes. The jobs scheduled between two consecutive PMs define a block, where B_k^i is the k^{th} block on machine M_i scheduled between the $(k-1)^{\text{th}}$ and k^{th} PMs (the 0^{th} and last PMs are the start and end of the schedule, respectively). The scheduling of a PM activity on each machine M_i is flexible and can actually occur before T_i minutes have elapsed, if it is not possible to avoid it or if it is beneficial to do so. Accordingly, the time length of a block is variable, although it can never exceed T_i for machine M_i. The duration of a PM activity on machine M_i is denoted by δ_i. As illustrated in Figure 1, there is no idle time in a schedule between two consecutive jobs or between a job and a PM.

Each job $j \in J$ is characterized by a known processing time p_j, a rejection cost u_j and a weight $w_j = h_j + b_j$ which is the sum of its inventory cost h_j and its priority level b_j. It should also be noted that no preemption is allowed. The two machines are non identical in the sense that PMs must be done more frequently on M_2. Thus, the maximum time interval T_2 between two consecutive PMs is smaller than T_1. A feasible solution S of problem (P) is evaluated first through objective f_1, which is the total

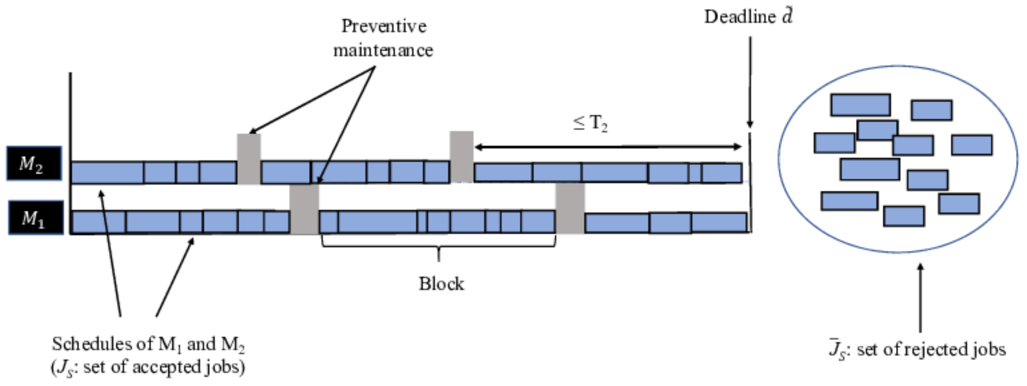
Preventive maintenance
Deadline $\bar{d}$
$\leq T_2$
M_2
M_1
Block
Schedules of M_1 and M_2 (J_S: set of accepted jobs)
$\bar{J}_S$: set of rejected jobs

Figure 1: Feasible solution of problem (P)

rejection cost of the jobs in $\overline{J}_S$, and second through objective f_2, which is the weighted sum of the completion times of the jobs in J_S. With regard to f_2, the WSPT (Weighted Shortest Processing Time) rule introduced in (Smith, 1956) is particularly important, because it optimally solves the $1 \mid\mid \sum_{j=1}^{n} w_j C_j$ scheduling problem, which minimizes the weighted sum of completion times on a single machine without side constraints. The WSPT rule states that the jobs should be scheduled in decreasing order of the w_j/p_j ratios. This rule will be exploited in our algorithms, although in a heuristic way since we have two machines with some operational constraints.

3.2 Model

The mathematical programming formulation of problem (P) is presented below. It involves five different types of decision variables.

$$C_j \quad : \quad \text{completion time of job } j \in J$$

$$m_k^i \quad : \quad \text{start time of the } k^{\text{th}} \text{ PM on machine } M_i$$

$$x_{lj}^i = \begin{cases} 1 & \text{if job } l \text{ is scheduled before job } j \text{ on machine } M_i \\ 0 & \text{otherwise} \end{cases}$$

$$z_j^i = \begin{cases} 1 & \text{if job } j \text{ is scheduled on machine } M_i \\ 0 & \text{if job } j \text{ is rejected} \end{cases}$$

$$y_{jk}^i = \begin{cases} 1 & \text{if job } j \text{ is scheduled in block } k \text{ on machine } M_i \\ 0 & \text{otherwise} \end{cases}$$

For the sake of the MILP formulation, two dummy jobs 0 and $n+1$ are added to the model with completion times $C_0 = C_{n+1} = 0$. Accordingly, we define the set $J^+ = J \cup \{0, n+1\}$. We also have $m_0^1 = m_0^2 = 0$. Note finally that M is an arbitrary large number.

Due to the lexicographic ordering of the two objectives, problem (P) can be solved optimally in two steps with an exact solver. A first model is solved with the objective of minimizing f_1 (while ignoring f_2). Next, a second model is solved with the objective of minimizing f_2 (with the constraint of not exceeding the optimal value found for f_1).

The first model is the following:

$$\min\,(f_1) = \sum_{j=1}^{n} u_j(1-(z_j^1+z_j^2)) \tag{1}$$

$$0 \le m_k^i \le m_{k-1}^i + T_i + \delta_i \qquad \forall k \in J,\ \forall i \in I \tag{2}$$

$$m_k^i \ge (\tilde{d} - \delta_i) \Rightarrow (m_k^i = 0) \qquad \forall k \in J,\ \forall i \in I \tag{3}$$

$$C_j \le m_b^i + M(1 - y_{jb}^i) \qquad \forall i \in I, \forall j \in J,\ b = 2,\dots,n \tag{4}$$

$$C_j - p_j y_{jb}^i \ge m_{b-1}^i + \delta_i - M(1 - y_{jb}^i) \qquad \forall i \in I, \forall j \in J,\ b = 2,\dots,n \tag{5}$$

$$\sum_{j=1,k\neq j}^{n+1} x_{kj}^i = z_k^i \qquad i \in I,\ k = 0,\dots,n \tag{6}$$

$$\sum_{k=0,k\neq j}^{n} x_{kj}^i = z_j^i \qquad \forall i \in I,\ j = 1,\dots,n+1 \tag{7}$$

$$x_{kj}^1 + x_{kj}^2 \le 1 \qquad k = 0,\dots,n \quad j = 1,\dots n+1 \tag{8}$$

$$C_j \ge p_j(z_j^1 + z_j^2) \qquad \forall j \in J,\ \forall i \in I \tag{9}$$

$$C_j \le \tilde{d}(z_j^1 + z_j^2) \qquad \forall j \in J,\ \forall i \in I \tag{10}$$

$$C_k \le C_j - p_j x_{kj}^i + \tilde{d}(1 - x_{kj}^i) \qquad \forall i \in I,\ k = 0,\dots,n \quad j = 1,\dots,n+1 \tag{11}$$

$$\sum_{j=1}^{n} p_j y_{jb}^i \le T_i \qquad \forall b \in J,\ \forall i \in I \tag{12}$$

$$\sum_{b=1}^{n} y_{jb}^i = z_j^i \qquad \forall j \in J,\ \forall i \in I \tag{13}$$

$$y_{jb}^1 + y_{jb}^2 \le 1 \qquad \forall (j,b) \in J \times J \tag{14}$$

$$z_j^1 + z_j^2 \le 1 \qquad \forall j \in J \tag{15}$$

$$z_k^i + z_j^i \ge 2(x_{kj}^i + x_{jk}^i) \qquad \forall (k,j) \in J \times J,\ \forall i \in I \tag{16}$$

$$C_0 = C_{n+1} = m_0^1 = m_0^2 = 0 \tag{17}$$

$$x_{kj}^i,\ y_{jb}^i,\ z_j^i \in \{0,1\} \qquad \forall (b,j,k) \in J \times J \times J,\ \forall i \in I \tag{18}$$

Equation (1) corresponds to the first objective function considered in this work. Constraints (2) allow to compute the starting time of the k^{th} PM on each machine M_i. When $\tilde{d}$ is reached, constraints (3) set to zero the start time of the k^{th} PM on each machine M_i. Constraints (4) and (5) forbid a scheduled job and a PM activity to overlap. Constraints (6) indicate that every job assigned to machine M_i, including the dummy job 0, should have a successor. Constraints (7) state that if a job j is assigned to machine M_i, at least one job should immediately follow, which includes the dummy job $n+1$. Constraints (8) state that two jobs scheduled consecutively should be assigned to the same machine. Constraints (9) and (10) define bounds on the completion time of each scheduled job (if a job is rejected, its completion time is set to 0). Constraints (11) indicate that two jobs scheduled on the same machine cannot overlap. Constraints (12) state that the sum of processing times of all jobs between two consecutive maintenance activities must be less than or equal to T_i. Since the number of blocks on each machine M_i is at most the number of jobs assigned to M_i, constraints (13) computes the number of accepted jobs in each block B^i_k. Constraints (14) enforce each accepted job j to be scheduled either on M_1 or M_2 but not both. Similarly, constraints (15) allow accepted jobs to be scheduled in a block B^i_k of either machine M_1 or M_2 but not both. Constraints (16) state that if two jobs l and j are scheduled on the same machine, then l is scheduled either before or after j. Constraints (17) set the completion times of dummy jobs 0 and $n+1$, and the start time of the first PM on each machine to 0. Finally, constraints (18) define the binary variables.

Let $f_1^\star$ be the optimal value of f_1 after solving the above model. In a second step, constraint $f_1 \le f_1^\star$ is added to the model and the latter is solved with objective f_2 only. In other words, the model below is considered:

$$\min\,(f_2) = \sum_{j=1}^{n} w_j C_j \tag{19}$$

$$s.t. \quad \text{Constraints } (2-18) \tag{20}$$

$$\sum_{j=1}^{n} u_j(1-(z_j^1+z_j^2)) \le f_1^\star \tag{21}$$

Equation (19) corresponds to the second objective, while constraint (21) bounds the value of the first objective. The solution obtained at the end of this second step is the optimal solution of (P). We observed that the CPLEX solver could only be used for small instances. More precisely, we were able to solve instances with up to 25 jobs within approximately 16 hours of computation time. But CPLEX had to be stopped after 24 hours of computation time, with a very large optimality gap, on instances with 40 jobs. These results support the use of heuristics and metaheuristics for instances of larger, more realistic, size.

In the following, our problem-solving methodologies are presented, starting with the greedy heuristic to generate a first feasible schedule, which is then improved with tabu search-based metaheuristics.

4 Greedy heuristic GrH

The greedy heuristic GrH calls a construction procedure which is aimed at producing a feasible schedule of good quality from a given set of jobs. In particular, GrH calls the construction procedure within a loop where the set of jobs is gradually reduced until all jobs can be scheduled, as it is explained below. In each proposed procedure of this work, ties are broken randomly if no other information is provided.

4.1 Main procedure

We can see from the description in Algorithm 1 that GrH starts by calling the greedy construction procedure (presented in Algorithm 2) with a set of jobs J', which is initially the set of all jobs J (steps 1 and 2). The construction procedure then returns a feasible solution S, which is associated with a set of accepted jobs J_S and a set of rejected jobs $\overline{J}_S$. If not all jobs in J' are accepted in solution S, we select the $|J_S|$ jobs in J with the largest u_j to obtain a smaller set J' (step 3a). The construction procedure is then called again with the new J' (step 3b). If the solution S obtained does not contain all jobs in J', we select again the $|J_S|$ jobs in J with the largest u_j to obtain an even smaller set J' (step 3a again), and the construction procedure is called with the latter (step 3b again). This is repeated until all jobs in J' are accepted in the obtained solution S, that is, when $J_S = J'$. Thus, the aim of the loop (step 3) is to schedule as many jobs as possible with the largest rejection costs, since f_1 is the main objective. Next (step 4), we consider the rejected jobs in the last solution obtained and we try to add them at the end of the schedule of machines M_1 and M_2. These jobs are considered one by one in decreasing order of rejection costs. First, we check if the current job j can be added without exceeding the deadline $\tilde{d}$ (if the addition of job j leads to exceeding the due time of the next PM, a PM must also be added before job j). If job j is feasible on a single machine, it is added to this machine; if job j is feasible on both machines, it is added to the machine with minimum completion time C_j (in order to account for f_2); if job j is not feasible on any machine, it is skipped.

Algorithm 1 GrH. Input: J. Output: S.

```
1. $J' \longleftarrow J$
2. $S \longleftarrow$ Construction($J'$)
3. Repeat until $J_S = J'$:
    (a) $J' \longleftarrow$ subset of $|J_S|$ jobs $j \in J$ with largest $u_j$
    (b) $S \longleftarrow$ Construction($J'$)
4. For each job $j \in \overline{J}_S$ (taken in decreasing order of $u_j$), do:
    (a) Try to add $j$ at the end of schedule of $M_1$ and $M_2$, while programming a PM before $j$ if required
    (b) If job $j$ is feasible on one machine, add $j$ to this machine
    (c) If job $j$ is feasible on both machines, add $j$ to the machine with minimum $C_j$
```

4.2 Construction procedure

The construction procedure, described in Algorithm 2, produces a solution S from scratch in a greedy way. Using the set J' of jobs provided in input, a new job j is selected and added, at each iteration, at the end of the schedule of M_1 or M_2. This is repeated (step 3) as long as there are jobs which can be added to the schedule of at least one machine without exceeding the deadline $\tilde{d}$ (if the addition of job j leads to exceeding the due time of the next PM, a PM must also be added before job j). This feasibility check is performed through calls to AssignFeasible, as described in Algorithm 3. AssignFeasible considers the set of jobs provided in input and returns only the subset of feasible jobs. In the process, each feasible job is tentatively assigned (but not scheduled) to a machine. If job j is feasible on a single machine, it is added to this machine; if job j is feasible on both machines, it is added to the machine with minimum completion time C_j (to account for f_2).

In the construction procedure, the selection of the next job is done as follows. First (step 3a), we consider the subset $J'_1 \subset J'$ of the q_1 (parameter $< n$) jobs with the largest w_j/p_j ratio, which is a good heuristic rule with regard to objective f_2. Second (steps 3b and 3c), we select the subset $J'_2 \subset J'_1$ containing the q_2 (parameter $< q_1$) jobs with the smallest completion times C_j, as determined in AssignFeasible. For each job $j \in J'_2$ (and its associated machine), we compute the slack time with the due time of the next PM, and we finally select the job $j^\star$ with the smallest slack time. The latter is then added (as well as a PM before $j^\star$, if required) at the end of the schedule of its associated machine. It should be noted that all jobs are considered in the first and second steps when the number of remaining jobs is smaller than q_1 and q_2, respectively.

Algorithm 2 Construction. Input: J'. Output: S.

```
1. $S \longleftarrow \emptyset$
2. $J' \longleftarrow$ AssignFeasible($J'$)
3. While $J' \neq \emptyset$, do:
    (a) Select subset $J'_1 \subset J'$ (of size $q_1$) of jobs $j$ with largest ratio $w_j/p_j$ ($J'$ is selected if $|J'| < q_1$)
    (b) Select subset $J'_2 \subset J'_1$ (of size $q_2$) of jobs $j$ with smallest recorded $C_j$ ($J'_1$ is selected if $|J'_1| < q_2$)
    (c) Select $j^\star \in J'_2$ with smallest slack time with the next required PM on assigned machine
    (d) Add $j^\star$ at the end of schedule of assigned machine, while programming a PM before $j^\star$ if required
    (e) $J' \longleftarrow$ AssignFeasible($J' \setminus \{j^\star\}$)
```

```
Algorithm 3 AssignFeasible.          Input: $J'$.          Output: $J'$.

1. For each job $j \in J'$, do:
    (a) Try to add $j$ at the end of schedule of $M_1$ and $M_2$, while programming a PM before $j$ if required
    (b) If $j$ is feasible on a single machine, assign $j$ to this machine and record $C_j$
    (c) If $j$ is feasible on both machines, assign $j$ to the machine with minimum $C_j$ and record $C_j$
    (d) If $j$ is not feasible on any machine, $J' \longleftarrow J' \backslash \{j\}$
```

5 Two tabu search metaheuristics *TSMN* and *CTS*

Introduced in (Glover, 1989), tabu search is a well-known metaheuristic for solving hard combinatorial optimization problems (Gendreau and Potvin, 2019), with a great success in job scheduling (Respen et al., 2016; Thevenin et al., 2016). Starting with some initial solution, a neighborhood of the current solution is generated at each iteration through local modifications (moves). The best solution in the neighborhood then becomes the new current solution, even if it does not provide an improvement. To avoid cycling in the solution space, a tabu list is also defined to forbid certain moves. Since tabu lists are not perfect filters, the tabu status of a move can always be revoked through aspiration criteria if there is no risk of cycling. The tabu search terminates when a stopping criterion is satisfied. The best solution found is returned at the end.

Two metaheuristics based on tabu search are proposed in this section. They are called *Tabu Search with Multiple Neighborhoods* (*TSMN*) and *Consistent Tabu Search* (*CTS*). *TSMN* explores only feasible solutions by exploiting different types of neighborhoods, whereas *CTS* admits moves leading to infeasible solutions which will be immediately repaired to restore feasibility (consistency). These two metaheuristics are now described.

5.1 Tabu search with multiple neighborhoods *TSMN*

TSMN improves the initial starting solution produced by the greedy heuristic *GrH*, while always maintaining feasibility. As shown in Algorithm 4, *TSMN* has three different phases with different neighborhood structures. The algorithm stops when I^{TSMN} global iterations have been performed (step 2) and the best-encountered solution $S^{\star}$ is returned. The latter is updated after each step of Algorithm 4 with respect to the lexicographic ranking $f_1 > f_2$.

Each global iteration corresponds to three consecutive tabu search phases. Phase 1 optimizes objective f_1, whereas the sequence of scheduled jobs obtained at the end of Phase 1 is modified in Phases 2 and 3 to optimize f_2. In these two last phases, no scheduled job can be rejected, thus only the sequences of jobs on the two machines are modified.

The neighborhood structures of the tabu search procedures exploit different types of moves for updating the current solution (as explained below). The best non-tabu move – over a random proportion Pr of all possible moves – is performed at each iteration of each tabu search procedure. The following values have been tested for parameter Pr in our computational study: 0.25, 0.5, 0.75 and 1. Each modification to the current solution needs to be correctly evaluated. It implies that jobs may have to be shifted to the right or to the left (in the latter case, to fill any idle time between two consecutive jobs). However, this is done only from the point of insertion of a new job to the end of the schedule, since nothing changes before the insertion point.

When a move is performed, its reverse move is forbidden for tab iterations, where tab is an integer randomly chosen in $[5, 10]$ for Phases 1 and 3, and in $[3, 7]$ for Phase 2 (these intervals were tuned after preliminary experiments). *TSMN* comprises a standard criterion aspiration: The tabu status of a move is revoked if it leads to a solution which is better than the best-encountered solution. There is no risk of cycling in this case, since this new best solution has clearly not been previously visited. The stopping criterion for each Phase $l \in \{1, 2, 3\}$ corresponds to a maximum number of iterations, denoted as I_l^{TSMN}. The value for each parameter was determined through parameter sensitivity analysis, as explained in Section 6.

```
Algorithm 4 TSMN.            Input: $J$.            Output: $S^\star$

1. $S \longleftarrow GrH(J)$
2. For $t = 1$ to $I^{TSMN}$, do:
    (a) Phase 1: $S \longleftarrow Tabu(S;\ SWAP^1;\ INSERT^1)$
    (b) Phase 2: $S \longleftarrow Tabu(S;\ SWAP^2)$
    (c) Phase 3: $S \longleftarrow Tabu(S;\ SWAP^3)$
```

The neighborhood structures used in Phases 1, 2 and 3 of *TSMN* are the following:

Phase 1. The tabu search $Tabu(S;\ SWAP^1;\ INSERT^1)$ optimizes only f_1 (rejection cost) using a neighborhood structure based on $SWAP^1$ and $INSERT^1$. More precisely, a move consists in sequentially swapping two jobs $j \in J_S$ and $j' \in \overline{J}_S$ ($SWAP^1$), and then, in trying to insert in the schedule jobs $j'' \in \overline{J}_S$ with a large rejection cost ($INSERT^1$). In $SWAP^1$, every pair of jobs $j \in J_S$ and $j' \in \overline{J}_S$ are considered for exchange. That is, a scheduled job is rejected and replaced by a previously rejected job. After each such potential exchange, the jobs $j'' \in \overline{J}_S$ are sorted in decreasing order of rejection cost $u_{j''}$. Then $INSERT^1$ considers the jobs in $\overline{J}_S$ one by one for insertion in the schedule, with the goal of inserting as many jobs as possible while keeping solution feasibility. Indeed, when a swap is applied between $j \in J_S$ and $j' \in \overline{J}_S$, the processing time $p_{j'}$ can well be greater than p_j, which may lead to exceeding the deadline $\tilde{d}$ (if it occurs, such a swap move is ignored). Conversely, when $p_{j'}$ is smaller than p_j, some idle time is created in the schedule and this flexibility can then be exploited by $INSERT^1$. Note that each swap move is evaluated with regards to objective f_1 after having performed the subsequent insertion moves.

Phase 2. The tabu searches in Phases 2 and 3 are aimed at improving the scheduling of accepted jobs, as identified in Phase 1, with regard to objective f_2. The neighborhood structure $SWAP^2$ exchanges every pair of blocks scheduled on the same machine only. Indeed, after some preliminary tests, we discovered that swapping blocks between the two machines was not beneficial because the blocks are not of the same size ($T_2 < T_1$). Consequently, the schedule of machine M_2 often exceeded the deadline after such a move, as illustrated in Figure 2 for the exchange of blocks B_1^2 and B_2^1.

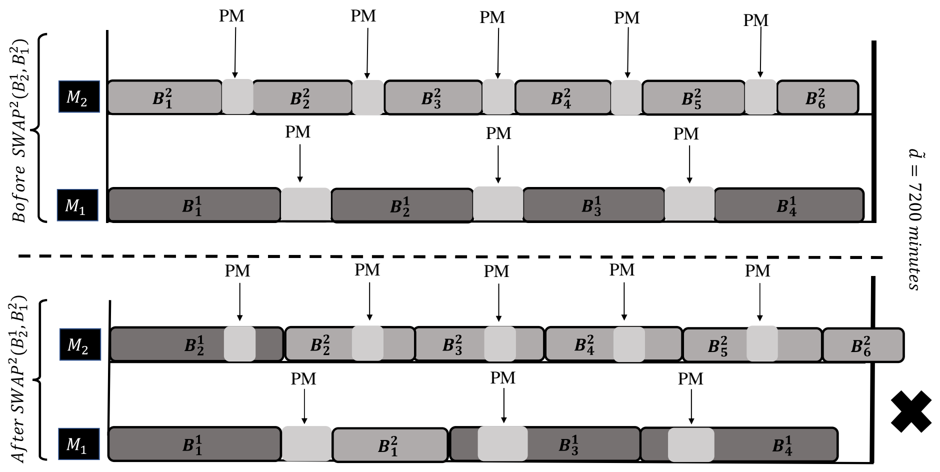
Before $SWAP^2(B_2^1, B_1^2)$
After $SWAP^2(B_2^1, B_1^2)$
M_2
M_1
PM
B_1^2
B_2^2
B_3^2
B_4^2
B_5^2
B_6^2
B_1^1
B_2^1
B_3^1
B_4^1
$\tilde{d}$ = 7200 minutes

Figure 2: Infeasible solution after swapping two blocks between M_1 and M_2

Phase 3. The neighborhood structure in Phase 3 is based on SWAP^3 moves where pairs of jobs, scheduled on the same machine or not, are exchanged. We consider all possible swaps between two jobs j and j', except when j appears before j' in the schedule of a given machine and $w_{j'}/p_{j'} < w_j/p_j$ (to be in line with the WSPT rule). The goal here is to obtain a better scheduling of the jobs within the blocks with regard to objective f_2.

5.2 Consistent tabu search *CTS*

This tabu search is inspired from the work in (Zufferey and Vasquez, 2015), where satellite range scheduling problems are addressed. As opposed to *TSMN*, infeasible neighbor solutions are considered but are immediately repaired to restore feasibility. This approach leads to the design of a simpler algorithmic scheme, as shown in Algorithm 5. There are two main phases in *CTS*, each based on a tabu search which is aimed at optimizing one of the two objectives.

An initial solution S is first generated using the greedy heuristic GrH (step 1). A total number of I^{CTS} global iterations is then performed (step 2). First, objective function f_1 is optimized in Phase 1 through insertion moves (based on the below-described $INSERT^2$ neighborhood structure). A maximum number of I_1^{CTS} iterations is performed with this tabu search, but the procedure is repeated as long as an improvement to the best-encountered solution is observed (step 2a). Next, f_2 is optimized in Phase 2 with the $SWAP^4$ neighborhood structure (step 2b), which is similar to the $SWAP^3$ used in *TSMN*. This tabu search is stopped after a maximum number of I_2^{CTS} iterations. As opposed to Phase 1, the procedure is not repeated as long as an improvement to the best-encountered solution is observed, because f_2 is a secondary objective.

Like *TSMN*, a tabu tenure is associated with each move. In Phase 1, a rejected job is tabu for reinsertion in the schedule for *tab* iterations, whereas the reverse swap move is tabu in Phase 2. In both cases, *tab* is an integer randomly chosen in the interval $[5, 10]$, based on preliminary experiments. The aspiration criterion is the same as in *TSMN*. The neighborhood structures are now presented.

Phase 1. Objective f_1 (rejection cost) is optimized using the neighborhood structure $INSERT^2$, where every rejected job is considered for insertion at every position in the schedule of machines M_1 and M_2. It is important to note that the insertion is enforced even if the deadline $\tilde{d}$ is exceeded. In such a case, the tentative solution is immediately repaired by removing accepted jobs that are positioned from the maintenance occurring just before j (or from the first job if there is no maintenance before j) to the end of the schedule (the selection of such candidate jobs to be removed limits the impact of a job removal on the solution structure, while facilitating the evaluation). More precisely, while the solution is not feasible, we sequentially remove a job j' from J_S in increasing order of their rejection costs (i.e., focus on f_1), and we break ties with the smallest ratio $w_{j'}/p_{j'}$ (i.e., focus on f_2).

Phase 2. Objective f_2 (weighted sum of completion times) is optimized using the neighborhood structure $SWAP^4$ (see $SWAP^3$ in *TSMN*). When a swap leads to exceeding the deadline $\tilde{d}$ on a machine, feasibility is restored as in Phase 1.

Algorithm 5 *CTS*. Input: J. Output: $S^\star$

```
1. $S \longleftarrow GrH(J)$
2. For $t = 1$ to $I^{CTS}$, do:
    (a) Repeat as long as $S^\star$ is improved: **Phase 1:** $S \longleftarrow Tabu(S;\ INSERT^2)$
    (b) **Phase 2:** $S \longleftarrow Tabu(S;\ SWAP^4)$
```

6 Computational study

This section reports computational results obtained with the proposed algorithms. The generation of test instances is first discussed in Subsection 6.1. Next, experiments conducted to determine the best value for different parameters are described in Subsection 6.2. Finally, comparisons between *TSMN* and *CTS* are reported and discussed in Subsection 6.3. All algorithms were coded in Java and the computational experiments were performed on an i7 Intel Core at 2.50 GHz with 16 GB of RAM. The MIP solver is CPLEX 12.7 coupled with Concert Technology for the Java interface.

Given that f_1 (rejection cost) is the main objective and subsumes f_2 (weighted sum of completion times), we will sometimes report only the values of f_1 in the following results for brevity purposes. Also, to allow a fair comparison between *TSMN* and *CTS*, their global iteration counters I^{TSMN} and I^{CTS} will be replaced by a computation time limit of one hour. Since *TSMN* and *CTS* are both stochastic algorithms, they are run 10 times on each instance and the best run is recorded (note by the way that the relative standard deviation is smaller than 0.2).

Apart from the results in Subsection 6.2.1, which is about GrH, the solution values are reported as improvement gaps (in percent) with respect to the solution values produced by GrH. Formally, $GAP = 100 \times [(GrH - TS)/GrH]$, where TS is either *TSMN* or *CTS*, and the improvement is calculated either with regard to objective f_1 or f_2, depending on the context.

6.1 Test instances

Since there are no available benchmark instances in the literature for problem (P), we carried out experiments based on randomly generated data, inspired from a real case in the pharmaceutical industry, as reported in (Zufferey et al., 2017).

The job processing time p_j in minutes is uniformly distributed in the interval [30, 120] with an average of 75 minutes. Thus, if we do not consider the maintenance activities, it is possible to schedule, on average, at most 96 jobs per machine over a scheduling horizon of 5 days (7200 minutes). For example, when $n = 200$, at least 8 jobs will be rejected. Accordingly, we generated instances of size $n \in \{200, 210, 220\}$. For each size, 10 instances were generated, for a total of 30 instances. Periodic preventive maintenance is performed on the two machines after a maximum of T_i minutes of use, with $T_1 = 480$ and $T_2 = 360$. The time required to perform a maintenance is set to 4% of T_i, which translates into $\delta_1 = 20$ minutes and $\delta_2 = 15$ minutes.

The weight $w_j = b_j + h_j$ is uniformly distributed in the interval [20, 60]. That is, the priority b_j is randomly selected in the set $\{10, 20, 30\}$, whereas h_j is uniformly distributed over the interval [10, 30]. Finally, u_j is uniformly distributed over the interval $[b_j p_j/2, 2b_j p_j]$, since the rejection cost of a job depends on its priority and its processing time.

6.2 Parameter calibration

In this subsection, we examine the impact of different parameters on the proposed algorithms. To this end, different values were tested for a given parameter, as it is explained below.

6.2.1 Parameters q_1 and q_2 of GrH

GrH was run with $q_1 \in \{0.05n, 0.1n, 0.2n, 0.3n, 0.4n, 0.5n, n\}$, whereas q_2 was set to $\frac{q_1}{2}$. Table 1 summarizes the average value of f_1 over the 10 instances for each size n and each q_1 value. It shows that GrH produces better solutions with smaller values of q_1. For a better view of the results, the entries in Table 2 provide the number of instances for which a given value of q_1 produces the best solution. Based on these results, the value $0.2n$ was selected.

	$q_1 = 0.05n$	$q_1 = 0.1n$	$q_1 = 0.2n$	$q_1 = 0.3n$	$q_1 = 0.4n$	$q_1 = 0.5n$	$q_1 = n$
$n = 200$	11270.5	**11159.7**	11462.8	11593	15200.2	16530.1	21020.6
$n = 210$	20970.5	21127.5	**16530.1**	18013.2	19292.3	20534.8	33035
$n = 220$	**31037.5**	31063.1	31365.5	33502	35590.3	38437.1	47697.5

Table 1: Average solution values of GrH based on f_1 for different values of q_1

	$q_1 = 0.05n$	$q_1 = 0.1n$	$q_1 = 0.2n$	$q_1 = 0.3n$	$q_1 = 0.4n$	$q_1 = 0.5n$	$q_1 = n$
$n = 200$	5/10	4/10	2/10	2/10	0/10	0/10	0/10
$n = 210$	4/10	1/10	6/10	2/10	0/10	0/10	0/10
$n = 220$	4/10	3/10	5/10	3/10	0/10	0/10	0/10

Table 2: Number of best solutions found by GrH based on f_1 for different values of q_1

6.2.2 Stopping criteria of *TSMN* and *CTS*

In *TSMN*, Phase l is stopped after I_l^{TSMN} iterations, with $l \in \{1, 2, 3\}$. First, we tested $I_1^{TSMN} \in \{\frac{n}{10}, \frac{n}{5}, n, 2n, 3n, 4n, 5n\}$. The gaps or improvements achieved on objective f_1 are summarized in Table 3 with the corresponding computation times in seconds. We observe that most of the optimization takes place in the first $\frac{n}{10}$ iterations, whereas the gap does not change much after $I_1^{TSMN} = 2n$ iterations. It should also be noted that the computation time is around one minute when the largest number of iterations $5n$ is performed. Based on these results, I_1^{TSMN} was set to $2n$.

Since Phases 2 and 3 are aimed at optimizing f_2, Table 4 shows the average gap values of objective f_2 at the end of Phase 3 for different values of I_3^{TSMN}, after fixing I_1^{TSMN} to $2n$. Again, most of the optimization takes place in the first $\frac{n}{10}$ iterations and the improvement is null or negligible after $3n$ iterations. The computations times never exceed 90 seconds, even when the largest number of iterations $5n$ is performed. Based on these results, I_3^{TSMN} was set to $3n$. It should be noted that I_2^{TSMN} was set to $n/5$ given the relatively small size of the corresponding neighborhood, where blocks are moved rather than individual jobs.

Since the two neighborhoods explored in *CTS* are similar to the ones in Phases 1 and 3 of *TSMN*, I_1^{CTS} and I_2^{CTS} were also set to $2n$ and $3n$, respectively. Also, as previously mentioned, the global iteration counters I^{TSMN} and I^{CTS} of *TSMN* and *CTS*, respectively, were replaced by a time limit of one hour.

	f_1 value			Computation time (s)		
	$n = 200$	$n = 210$	$n = 220$	$n = 200$	$n = 210$	$n = 220$
$n/10$	25.66	19.62	18.41	0.67	1.26	3.79
$n/5$	25.70	19.66	18.48	0.81	2.10	22.47
n	25.71	19.76	18.50	2.96	13.50	28.27
$2n$	25.78	19.77	18.57	9.77	23.65	41.59
$3n$	25.78	19.77	18.68	13.29	35.78	41.90
$4n$	25.78	19.77	18.68	37.74	36.96	49.46
$5n$	25.78	19.77	18.68	41.84	51.02	64.66

Table 3: Average value of f_1 with regard to I_1^{TSMN} and instance size n

	f_2 value			Computation time (s)		
	$n = 200$	$n = 210$	$n = 220$	$n = 200$	$n = 210$	$n = 220$
$n/10$	38.57	37.68	40.53	5.42	7.44	8.16
$n/5$	38.63	37.75	40.57	9.89	12.64	10.35
n	38.65	37.84	40.57	18.35	21.22	22.85
$2n$	38.70	37.87	40.64	26.12	29.67	32.18
$3n$	38.71	37.90	40.67	30.05	31.87	39.05
$4n$	38.73	37.90	40.67	45.17	25.76	28.18
$5n$	38.75	37.90	40.67	58.16	65.55	86.24

Table 4: Average value of f_2 with regard to I_3^{TSMN} and instance size n

6.2.3 Neighborhood size parameter Pr^N of *TSMN* and *CTS*

We ran Phases 1 and 3 of *TSMN* with $Pr^N \in \{1, \frac{3}{4}, \frac{1}{2}, \frac{1}{4}\}$, where Pr^N is the fraction of the neighborhood N explored. For these experiments, the number of iterations of *TSMN* in Phases 1, 2 and 3 was set to $2n$, $n/5$ and $3n$, respectively. Tables 5 and 6 summarize the results, considering that Phase 1 optimizes f_1 and Phase 3 optimizes f_2.

From these results, we can easily see that high values of Pr^N are associated with better objective values, although at a computational cost. For $n = 220$ in Table 5, for example, the average computation time with $Pr^N = 1$ is 27.6 seconds, whereas it is only 7.18 seconds with $Pr^N = \frac{1}{4}$. On the other hand, the average solution value is substantially better with $Pr^N = 1$.

		$Pr^N = 1$	$Pr^N = \frac{3}{4}$	$Pr^N = \frac{1}{2}$	$Pr^N = \frac{1}{4}$
$n = 200$	f_1	25.68	25.50	25.29	24.77
	Computation time (s)	9.77	3.87	1.46	0.97
$n = 210$	f_1	19.71	19.58	19.55	18.98
	Computation time (s)	23.66	10.80	6.68	2.70
$n = 220$	f_1	18.56	18.23	18.21	18.12
	Computation time (s)	27.60	16.37	8.30	7.18

Table 5: Comparison of f_1 values for different instance sizes n and Pr^N values

		$Pr^N = 1$	$Pr^N = \frac{3}{4}$	$Pr^N = \frac{1}{2}$	$Pr^N = \frac{1}{4}$
$n = 200$	f_2	39.95	38.88	38.65	38.64
	Computation time (s)	28.05	28.97	22.71	11.74
$n = 210$	f_2	38.67	38.17	38.01	38.00
	Computation time (s)	30.28	31.16	19.95	11.17
$n = 220$	f_2	41.94	40.73	40.62	40.56
	Computation time (s)	38.46	30.76	17.45	13.18

Table 6: Comparison of f_2 values for different instance sizes n and Pr^N values

Based on these results, and given the benefits obtained from a complete exploration of each neighborhood, Pr^N was set to 1 in Phases 1 and 3. Furthermore, the whole neighborhood is also explored in Phase 2 of *TSMN*, due to its relatively small size. Based on the results obtained with *TSMN*, Pr^N was also set to 1 in Phases 1 and 2 of *CTS*.

6.3 Results

This subsection reports the results obtained with *TSMN* and *CTS*. First, we analyze the impact of each phase of *TSMN*. Second, we compare and discuss the final solutions produced by *TSMN* and *CTS* after one hour of computation time.

6.3.1 Analysis of each phase of *TSMN* on f_1 and f_2

Table 7 reports the improvement reached in objectives f_1 and f_2 after the first pass through each phase of *TSMN*. At this point, we must remember that Phase 1 is aimed at optimizing f_1, whereas Phases 2 and 3 optimize f_2. Thus, we report only the results for the corresponding objectives.

	f_1			f_2		
TSMN components	$n = 200$	$n = 210$	$n = 220$	$n = 200$	$n = 210$	$n = 220$
Phase 1	25.25	17.04	18.21	-	-	-
Phase 2	-	-	-	15.52	13.43	17.97
Phase 3	-	-	-	38.52	37.50	40.65

Table 7: Impact of different phases of *TSMN*

Tables 8 and 9 summarize the average run times consumed by each phase of *TSMN* and *CTS* over one hour of computation time. With regard to *TSMN*, Phase 3 is clearly the most time-consuming, since pairs of accepted jobs are considered for exchange. Conversely, Phase 2 is the least time-consuming because whole blocks of jobs are exchanged. With regard to *CTS*, Phase 1 must consider each job for insertion at every possible position (even infeasible ones) in the schedule, which explains the substantial amount of time required for evaluation and repair.

	$n = 200$	$n = 210$	$n = 220$
Phase 2	7.032	6.97	7.27
Phase 3	3445.11	3433.04	3395.6

Table 8: Average computation time (in seconds) for each phase of *TSMN*

	$n = 200$	$n = 210$	$n = 220$
Phase 1	2586.09	2400.14	2013.84
Phase 2	1031.75	1203.11	1588.30

Table 9: Average computation time (in seconds) for each phase of *CTS*

6.3.2 Comparison between *TSMN* and *CTS*

Table 10 compares *TSMN* and *CTS* with regard to objectives f_1 and f_2 on our test instances with $n = 200$, 210 and 220 jobs. We note that *TSMN* produces significantly better solutions with regard to objective f_1, whereas *CTS* does slightly better for f_2.

	f_1			f_2		
	$n = 200$	$n = 210$	$n = 220$	$n = 200$	$n = 210$	$n = 220$
GAP_{TSMN}	25.7	20.2	18.9	38.4	37.3	36.3
GAP_{CTS}	22.9	16.5	11.0	40.9	38.7	37.7

Table 10: Comparison between *TSMN* and *CTS*

Table 11 shows the number of times that each method found the best solution, for each objective, over the 10 test instances of each size. These results are in line with those reported in Table 10, although it should be noted that the slight superiority on average of *CTS* over *TSMN* for objective f_2 holds on almost every instance. Since f_1 is the main objective, though, we conclude that *TSMN* is the best method on the considered set of instances.

		TSMN	*CTS*
$n = 200$	f_1	10/10	0/10
	f_2	1/10	9/10
$n = 210$	f_1	8/10	2/10
	f_2	0/10	10/10
$n = 220$	f_1	9/10	1/10
	f_2	0/10	10/10

Table 11: Number of best solutions for *TSMN* and *CTS* for $n = 200$, 210 and 220 jobs

Table 12 reports the average number of rejected jobs for each method (including *GrH*) and for each instance size. With *TSMN*, we observe an average improvement over *GrH* of 4.4, 3.3 and 10.2 jobs for $n = 200$, 210 and 220, respectively. Furthermore, *TSMN* improves over *CTS* by allowing a reduction of 2.1, 1.4 and 5.3 jobs for $n = 200$, 210 and 220, respectively. It is important to remember here that the number of rejected jobs is just a proxy for the true objective f_1 (total cost of rejected jobs), but *TSMN* still outperforms *CTS* here.

	$n = 200$	$n = 210$	$n = 220$
GrH	24.9	39.5	51.5
TSMN	20.5	36.2	41.3
CTS	22.6	37.6	46.6

Table 12: Average number of rejected jobs

Finally, Figure 3 shows the evolution over time of objective f_1 of the best solution for *TSMN* and *CTS*, respectively. At time $t = 0$, the value shown is the one obtained with *GrH*. In the case of *TSMN*, there is a noticeable improvement up to $t = 30$ minutes, after which the improvement is rather small. On the other hand, *CTS* reaches a plateau at about t= 45 minutes.

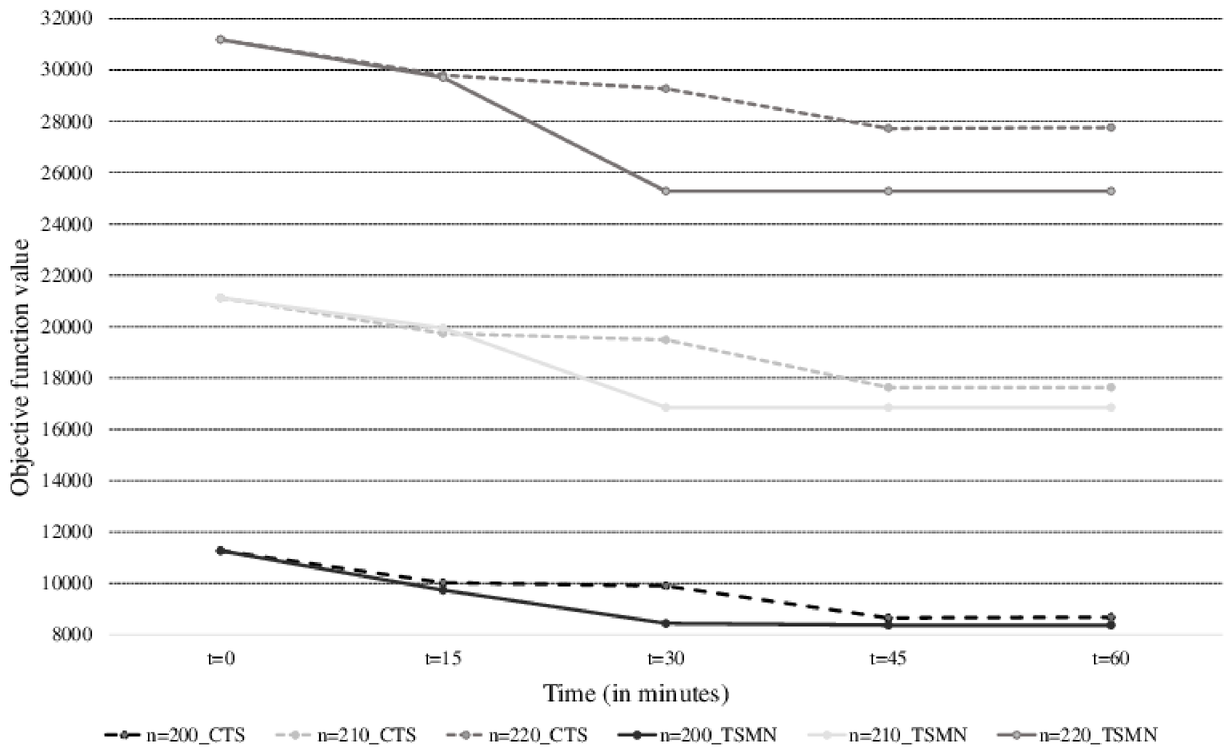

Figure 3: Evolution over time of f_1 for *TSMN* and *CTS*for each instance size

7 Conclusion and perspectives

In this work, we studied a parallel-machine scheduling problem with two non identical machines over a weekly planning horizon, while considering periodic preventive maintenance. Two objectives were considered and addressed with lexicographic optimization, namely, minimization of job rejection cost and weighted sum of job completion times (which can be seen as an inventory penalty). We first introduced a MILP formulation for the problem. Next, we developed a greedy heuristic and two tabu search-based metaheuristics, denoted *TSMN* and *CTS*. Computational experiments were performed on randomly generated data. They showed that *TSMN* outperforms *CTS* for the job rejection cost, which is the main objective, whereas *CTS* did slightly better for the weighted sum of job completion times.

Various research axis are possible for the future. On the one hand, alternative problem-solving methodologies could be explored for problem (P), like the Adaptive Large Neighborhood Search (ALNS). On the other hand, an extension of (P) could be studied, where several machines and/or optimization criteria are involved. Finally, a stochastic variant of (P) can be investigated, where random machine breakdowns can occur over time.

References

Aggelogiannaki, E. and Sarimveis, H. Multiobjective constrained MPC with simultaneous closed-loop identification. *International Journal of Adaptive Control and Signal Processing*, 20(4):145–173, 2006.

Agnetis, A. and Mosheiov, G. Scheduling with job-rejection and position-dependent processing times on proportionate flowshops. *Optimization Letters*, 11(4):885–892, 2017.

Bahriye, C., Ceyda, O., and Sibel, S. F. A tabu search algorithm for order acceptance and scheduling. *Computers & Operations Research*, 39(6):1197–1205, 2012.

Cao, Z., Wang, Z., Zhang, Y., and Liu, S. On several scheduling problems with rejection or discretely compressible processing times. *Theory and Applications of Models of Computation*, pages 90–98, 2006.

Ehrgott, M. *Multicriteria Optimization*, volume 491. Springer Science & Business Media, 2005.

Emami, S., Sabbagh, M., and Moslehi, G. A Lagrangian relaxation algorithm for order acceptance and scheduling problem: a globalised robust optimisation approach. *International Journal of Computer Integrated Manufacturing*, 29(5):535–560, 2016.

Engels, D. W., Karger, D. R., Kolliopoulos, S. G., Sengupta, S., Uma, R., and Wein, J. Techniques for scheduling with rejection. *Journal of Algorithms*, 49(1):175–191, 2003.

Gallay, O. and Zufferey, N. Metaheuristics for lexicographic optimization in industry. In *Proceedings of the 19th EU/ME Workshop on Metaheuristics for Industry (EU/ME 2018)*, 2018.

Gendreau, M. and Potvin, J.-Y. *Handbook of Metaheuristics*, volume 146 of *International Series in Operations Research & Management Science*. Springer, 2019.

Glover, F. Tabu search - Part I. *ORSA Journal on Computing*, 1(3):190–206, 1989.

Graham, R. L., Lawler, E. L., Lenstra, J. K., and Kan, A. R. Optimization and approximation in deterministic sequencing and scheduling: a survey. *Annals of Discrete Mathematics*, 5:287–326, 1979.

Jiang, D., Tan, J., and Li, B. Order acceptance and scheduling with batch delivery. *Computers & Industrial Engineering*, 107:100–104, 2017.

Kaabi, J. and Harrath, Y. A survey of parallel machine scheduling under availability constraints. *International Journal of Computer and Information Technology*, 3(2):238–245, 2014.

Kerrigan, E. C. and Maciejowski, J. M. Designing model predictive controllers with prioritised constraints and objectives. In *Proceedings of IEEE International Symposium on Computer Aided Control System Design*, pages 33–38. IEEE, 2002.

Li, G., Liu, M., Sethi, S. P., and Xu, D. Parallel-machine scheduling with machine-dependent maintenance periodic recycles. *International Journal of Production Economics*, 186:1–7, 2017.

Li, S.-S. and Chen, R.-X. Scheduling with rejection and a deteriorating maintenance activity on a single machine. *Asia-Pacific Journal of Operational Research*, 34(2), 2017.

Ma, R. and Yuan, J.-J. Online scheduling with rejection to minimize the total weighted completion time plus the total rejection cost on parallel machines. *Journal of the Operations Research Society of China*, 4(1):111–119, 2016.

Moghaddam, A., Amodeo, L., Yalaoui, F., and Karimi, B. Single machine scheduling with rejection: Minimizing total weighted completion time and rejection cost. *International Journal of Applied Evolutionary Computation*, 3(2):42–61, 2012.

Nobibon, F. T. and Leus, R. Exact algorithms for a generalization of the order acceptance and scheduling problem in a single-machine environment. *Computers & Operations Research*, 38(1):367–378, 2011.

Ocampo-Martinez, C., Ingimundarson, A., Puig, V., and Quevedo, J. Objective prioritization using lexicographic minimizers for MPC of sewer networks. *IEEE Transactions on Control Systems Technology*, 16(1):113–121, 2008.

Oğuz, C., Sibel, S. F., and Bilgintürk, Y. Z. Order acceptance and scheduling decisions in make-to-order systems. *International Journal of Production Economics*, 125(1):200–211, 2010.

Ou, J. and Zhong, X. Order acceptance and scheduling with consideration of service level. *Annals of Operations Research*, 248(1-2):429–447, 2017.

Ou, J., Zhong, X., and Wang, G. An improved heuristic for parallel machine scheduling with rejection. *European Journal of Operational Research*, 241(3):653–661, 2015.

Potts, C. N. and Van Wassenhove, L. N. A branch and bound algorithm for the total weighted tardiness problem. *Operations Research*, 33(2):363–377, 1985.

Prats, X., Puig, V., Quevedo, J., and Nejjari, F. Lexicographic optimisation for optimal departure aircraft trajectories. *Aerospace Science and Technology*, 14(1):26–37, 2010.

Qi, Y., Wan, L., and Yan, Z. Scheduling jobs with maintenance subject to load-dependent duration on a single machine. *Mathematical Problems in Engineering*, 2015, 2015.

Respen, J., Zufferey, N., and Amaldi, E. Metaheuristics for a Job Scheduling Problem with Smoothing Costs Relevant for the Car Industry. *Networks*, 67 (3):246 – 261, 2016.

Shabtay, D., Gaspar, N., and Yedidsion, L. A bicriteria approach to scheduling a single machine with job rejection and positional penalties. *Journal of Combinatorial Optimization*, 23(4):395–424, 2012.

Shabtay, D., Gaspar, N., and Kaspi, M. A survey on offline scheduling with rejection. *Journal of Scheduling*, 16(1):3–28, 2013.

Slotnick, S. A. Order acceptance and scheduling: A taxonomy and review. *European Journal of Operational Research*, 212(1):1–11, 2011.

Slotnick, S. A. and Morton, T. E. Order acceptance with weighted tardiness. *Computers & Operations Research*, 34(10):3029–3042, 2007.

Smith, W. E. Various optimizers for single-stage production. *Naval Research Logistics Quarterly*, 3(1-2): 59–66, 1956.

Solnon, C., Cung, V. D., Nguyen, A., and Artigues, C. The car sequencing problem: Overview of state-of-the-art methods and industrial case-study of the ROADEF'2005 challenge problem. *European Journal of Operational Research*, 191(3):912–927, 2008.

Sun, K. and Li, H. Scheduling problems with multiple maintenance activities and non-preemptive jobs on two identical parallel machines. *International Journal of Production Economics*, 124(1):151–158, 2010.

Thevenin, S., Zufferey, N., and Widmer, M. Metaheuristics for a scheduling problem with rejection and tardiness penalties. *Journal of Scheduling*, 18(1):89–105, 2015.

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

Thevenin, S., Zufferey, N., and Glardon, R. Model and Metaheuristics for a Scheduling Problem Integrating Procurement, Sale and Distribution. *Annals of Operations Research*, 259 (1):437 – 460, 2017a.

Thevenin, S., Zufferey, N., and Potvin, J.-Y. Makespan minimisation for a parallel machine scheduling problem with preemption and job incompatibility. *International Journal of Production Research*, 55 (6):1588–1606, 2017b.

T'kindt, V. and Billaut, J.-C. *Multicriteria Scheduling: Theory, Models and Algorithms*. Springer Science & Business Media, 2006.

Wang, X., Xingzi, X., and Cheng, T. Order acceptance and scheduling in a two-machine flowshop. *International Journal of Production Economics*, 141(1):366–376, 2013.

Wang, X., Huang, G., Hu, X., and Cheng, T. E. Order acceptance and scheduling on two identical parallel machines. *Journal of the Operational Research Society*, 66(10):1755–1767, 2015.

Zhang, S.-X., Cao, Z., and Zhang, Y. Scheduling with rejection to minimize the total weighted completion time. volume 9, pages 111–114. International Symposium on Operations Research and Its Applications, 2009.

Zhong, X. and Ou, J. Parallel machine scheduling with restricted job rejection. *Theoretical Computer Science*, 690:1–11, 2017.

Zhong, X., Ou, J., and Wang, G. Order acceptance and scheduling with machine availability constraints. *European Journal of Operational Research*, 232(3):435–441, 2014.

Zhong, X., Pan, Z., and Jiang, D. Scheduling with release times and rejection on two parallel machines. *Journal of Combinatorial Optimization*, 33(3):934–944, 2017.

Zufferey, N. and Vasquez, M. A generalized consistent neighborhood search for satellite range scheduling problems. *RAIRO-Operations Research*, 49(1):99–121, 2015.

Zufferey, N., Molin, D. D., Glardon, R., and Tsagkalidis, C. *Handbook of Research on Applied Optimization Methodologies in Manufacturing Systems*, chapter A Simulation-Optimization Approach for the Production of Components for a Pharmaceutical Company (ISBN: 978-1-52252-944-6). IGI Global, 2017.

Zykina, A. V. A lexicographic optimization algorithm. *Automation and Remote Control*, 65(3):363–368, 2004.