Optim Lett (2015) 9:465–482
DOI 10.1007/s11590-014-0761-7

ORIGINAL PAPER

Upper and lower bounds for the permutation flowshop scheduling problem with minimal time lags

Imen Hamdi · Taïcir Loukil

Received: 6 September 2013 / Accepted: 31 May 2014 / Published online: 18 June 2014

Abstract In this paper, we consider the problem of scheduling n jobs in an m-machine permutation flowshop with minimal time lags between consecutive operations of each job. The processing order of jobs is to be the same for each machine. The time lag is defined as the waiting time between two consecutive operations of each job. Upper bounds for the problem are provided by applying heuristic procedures based on known and new rules. Lower bounds based on Moore's algorithm and logic-based Benders decomposition are developed. For the last one, we define a long time horizon on the last machine divided into many segments of time. We combine a mixed integer linear programming to allocate jobs to time segments and scheduled using the constraint programming. Then, computational results are reported.

Keywords Scheduling · Permutation flowshop · Time lags · Upper bounds · Lower bounds

1 Introduction

We address an important scheduling problem that occurs frequently in manufacturing and production contexts. We study the permutation flowshop scheduling problem with minimal time lags. According to the standard notation given by Graham et al. [8], this problem can be noted $F_\pi|\theta_{i,k}^{min}|\sum_i U_i$. It can be described as follows: There are n jobs to be scheduled for processing on m machines. Each machine can process at most one operation at a time and preemption is not allowed. Each job $i \in \{1, 2, \ldots, n\}$

I. Hamdi (✉) · T. Loukil
Research Unit LOGIQ, High Institute of Industrial Management, University of Sfax, Sfax 3018, Tunisia
e-mail: imen.hamdi2007@yahoo.fr

T. Loukil
e-mail: taicir.loukil@fsegs.rnu.tn

is processed on the m machines during $p_{i,1},\ p_{i,2},\ldots,\ p_{i,m}$ time units successively subject to a due date d_i. We consider here additional constraints that extend the classical model: for each job, a definite amount of time must elapse between each two consecutive operations which has to be greater than or equal to a non negative value called minimal time lag (θ^{min}). Our objective is to find a schedule for which the total number of tardy jobs is minimized.

Although this problem arise in many industrial sectors, it proves to be quite difficult to solve. Many industrial situations involving specific manufacturing processes may be modelled using minimal time lags, for example in the field of biotechnologies and chemistry where the chemical reactions with variable processing times may be represented by minimal time lags [1]. Other examples are given by Deppner [4].

Over the scheduling literature, Hariri and Potts [9] present the first exact algorithm for the NP-hard $F_m||\sum U_i$ problem. They use a branch and bound algorithm that can solve problems with up to 3 machines with 25 jobs. For the same problem, Lenstra et al. [17] show that the problem is NP-hard for 2 machines. Lodree et al. [18] propose a new sequencing rule, Earliest Adjusted Due Date (EADD) for the $F_m|r_i|\sum U_i$ problems with up to 50 jobs on 3 machines. The rule is derived by considering a variation of the Moore's algorithm [19] which is then proven to outperform the shortest processing time dispatching rule.

Recently, Dhouib et al. [5] turn to propose two mathematical formulations for the permutation flowshop scheduling problem with setup times and time lags constraints; then a simulated annealing algorithm is developed to solve this problem. Ruiz-Torres et al. [20] solve minimizing the number of tardy jobs in flow shop environment with operation and resource flexibility. They develop lower bound procedure and efficient solution approaches to solve each sub-problem. For the case of common due dates, Croce et al. [3] treat the 2-machine to minimize the total number of tardy jobs, which is an NP-hard in the ordinary sense, a compact multidimensional knapsack problem formulation of the problem is introduced together with several structural properties. Then, they propose a branch and bound procedure to find an optimal solution to the problem.

Solution methods based on partitioning the problem into many smaller problems make the problem more easy to be solved. Recently, great interests are oriented to the logic-based Benders decomposition. We can model a problem as a Mixed Integer Linear Programming ($MILP$) or a Constraint Programming (CP) which is not perfomant especially with the complex structure. The $MILP$ is effective only when we are interested in the optimization aspect and the CP is more effective only for the feasibility criterion especially for highly constrained discrete optimization problems. To exploit the strength of the $MILP$ and the CP, we integrate them by the logic-based Benders decomposition. According to Hooker [14], the hybrid method generally achieves speedups of two or three orders of magnitude on larger instances when minimizing the number of late tasks, and it solves significantly more problems to optimality.

We consider a permutation flowshop problem with minimal time lags between operations. The objective function is to minimize the total number of tardy jobs which have different due dates. We try to adopt this technique to our problem to generate lower bounds. To make the proposed approach viable, we focus on the last machine

m to define a long time horizon divided into many segments of time. We develop a lower bound on the completion time on the first $m - 1$ machines for each job. This bound is considered as a release date for each job to be processed to the last machine after its assignment to the time segment. So that, an assignment problem for each job on each segment of time becomes the master problem and is solved by $MILP$, while the scheduling problem becomes the subproblem which separates into several independent scheduling problems is solved by CP. It is then compared to another lower bound based on Moore's algorithm. The developed lower bounds are evaluated by determining the relative deviation from a developed upper bound. This last one is chosen as the best one among some proposed heuristic algorithms.

The organization of the remainder of this paper is as follows: in Sect. 2, we present the problem under consideration and we describe the used notations. Section 3 is devoted to present three proposed heuristic procedures useful to provide upper bounds. In Sect. 4, we present the developed lower bounds. We remind some previous works about the logic-based Benders decomposition over the literature, and we detail its application. Thereafter, we describe the lower bound based on Moore's algorithm. Then, computational results are reported in Sect. 5, and finally in Sect. 6 we discuss concluding remarks.

2 Problem's definition and nomenclatures

The considered problem ($F_\pi|\theta_{i,k}^{min}|\sum_i U_i$) is characterized by n jobs and each one is composed of m operations as we described previously. These jobs have to be processed on a set of m machines ($k = 1, 2, \ldots, m$) following the same order: first on machine 1, second on machine 2, and so on until machine m. The order of the execution of the jobs is the same on all machines. So, only one sequence is considered. The processing time of job i on machine k ($p_{i,\,k}$), the minimal time lag which is a waiting time that must exist between the end of the operation k and the beginning of the operation $k + 1$ of the job i, and the due date (d_i) are fixed in advance. Then, for a considered sequence of jobs, we denote by $C_{i,k}$ the completion time of job i on machine k ($i = 1, 2, \ldots, n$; $k = 1, 2, \ldots, m$). If job i finishes on machine k at $C_{i,k}$ then i can start on machine $k + 1$ at time $t_{i,k+1}$ such that $t_{i,k+1} \geq C_{i,k} + \theta_{i,k}^{min}$. We consider the number of tardy jobs as a criterion to minimize. A job is called tardy if its completion time exceeds its due date ($C_j > d_j$), otherwise it is early. We define $T = \sum_i U_i$ as the number of tardy jobs where $U_i = 1$ if the job is tardy and 0 otherwise. As same, we define "E" as the number of early jobs ($E = \{i|C_i < d_i\}$). As we are interested in the permutation flowshop problem, we consider the completion time on the last machine m for each job i ($C_{i,m} \forall i = 1, 2, \ldots, n$) to define the set of tardy and early jobs ($T = \{i|C_{i,m} > d_i\}$ and $E = \{i|C_{i,m} < d_i\}$ respectively).

It is known that the general problem of permutation flowshop without time lags is NP-hard in the strong sense even if all jobs have a common due date [17], then the considered problem is obviously NP-hard in the strong sense. $\pi = \{\pi(1), \pi(2), \ldots, \pi(n)\}$ is used to design the permutation or the sequence of jobs where $\pi(1)$ for example means the job in the first position. The following example is

Table 1 The data

	$p_{i,1}$	$p_{i,2}$	$p_{i,3}$	$\theta_{i,1}^{min}$	$\theta_{i,2}^{min}$	d_i
Job 1	6	8	5	2	1	23
Job 2	3	5	4	3	2	20
Job 3	5	5	4	5	3	33

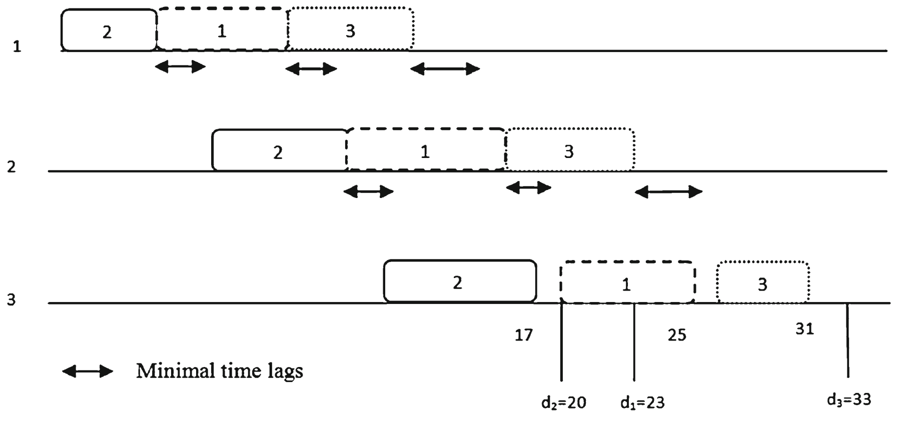

Fig. 1 Scheduling figure

used to explain better the problem which shows the scheduling of 3-jobs on 3 machines. The data are given in Table 1.

Here, $p_{i,1}$ for example design the processing time of job i on machine 1, and $\theta_{i,1}^{min}$ design the minimal time lag of job i between machines 1 and 2. Then, the scheduling is presented in Fig.1. It shows that for a given sequence of jobs ($\pi = 2 - 1 - 3$), there are one tardy job which is the job in the second position in the sequence as its completion time (the noted value on the last machine which is equal for this job 25) exceeds its due date (23), then $T = \{1\}$. However, the jobs in the first and third positions in the considered sequence are early as their completion times (17 and 31 respectively) doesn't exceed their due dates (20 and 33 respectively) and then $E = \{2, 3\}$.

Then the used nomenclatures in this paper are described as follows:

- n: number of jobs ($i \in \{1, 2, \ldots, n\}$)
- m: number of machines ($k \in \{1, 2, \ldots, m\}$)
- J : segments of time ($j \in \{1, 2, \ldots, J\}$)
- $p_{i,k}$: processing time of job i on machine k
- $\theta_{i,k}^{min}$: minimal time lag of job i between machine k and machine $k + 1$
- $C_{i,k}$: completion time of job i on machine k
- d_i : due date of job i
- r_i : release date of job i
- $d_{i,k}$: due date of job i on machine k
- $r_{i,k}$: release date of job i on machine k
- E: set of early jobs

- T: set of tardy jobs
- U_i =1 if the job i is tardy and 0 else
- $\pi(i), k$: the job in position i in the schedule sequence on machine k
- $C_{\pi(i),k}$: completion time of job in position i on machine k
- $d_{\pi(i)}$: due date of job in position i
- $p_{\pi(i),k}$: processing time of job in position i on machine k

3 Upper bounds

Consider a permutation flowshop scheduling problem with minimal time lags. Some heuristic procedures are proposed where the generated solution is considered as an upper bound of the optimal solution. They are based on three different rules: the well known rule “Shortest Processing Time (SPT)”, and two other new proposed rules “Shortest Sum of Processing Times ($SSPT$)” and “Adjustment on the Bottleneck Machine (ABM)”. Then the obtained sequences by using the proposed rules are scheduled by applying the “$Algorithm\, C$” described later to obtain the total number of tardy jobs. Here $\pi(i), k$ denote the job in position i in the schedule sequence π on machine k; $i = 1, 2, \ldots, n$ and $k = 1, 2, \ldots, m$.

3.1 Shortest processing time (SPT) rule

The SPT rule states to sequence the jobs in nondecreasing order of processing times. As we are interested in a permutation flowshop problem, we may find m different sequences, one sequence for each machine $k \in \{1, \ldots, m\}$ and the jobs are arranged according to $p_{\pi(1),k} \leq p_{\pi(2),k} \leq \cdots \leq p_{\pi(n),k}$. We can determine another sequence which is the $(m + 1)^{th}$ one using the rule of Total Processing time $(TP_{\pi(1)} \leq TP_{\pi(2)} .. \leq TP_{\pi(n)})$ as same in [7]. Now, we determine the total tardiness of all the $m + 1$ sequences and we select the final schedule with the smallest total number of tardy jobs. This rule determines the final sequence through the following algorithmic steps:

Step 1: Let $k = 1$.

Step 2: Sequence jobs in non-decreasing order of $p_{i,k}(p_{\pi(1),k} \leq p_{\pi(2),k} \leq \cdots \leq p_{\pi(n),k})$ on machine k.

Step 3: Define a permutation schedule on all machines using this sequence and determine the total number of tardy jobs by applying the $Algorithm\, C$ described later.

Step 4: If $k = m$, go to step 5, otherwise let $k = k + 1$, and go to step 2.

Step 5: Sequence jobs in non-decreasing order of $TP_i(TP_{\pi(1)} \leq TP_{\pi(2)} \leq ..TP_{\pi(n)})$. Define a permutation schedule on all machines using this sequence and then calculate the total number of tardy jobs of all the jobs by applying the “$Algorithm\, C$” described later.

Step 6: Select the final schedule with the smallest total number of tardy jobs among the $m + 1$ schedules, and stop.

3.2 Shortest Sum of Processing Times (SSPT) rule

This rule is similar to the well known SPT rule, but instead of using the individual job processing time on each machine, we consider for each job the total processing times on all machines plus the exact time lags between each consecutive couple of operations. Then by obtaining a sequence, we calculate the total number of tardy jobs by applying the "*Algorithm C*" described later.

3.3 Adjustment on the Bottleneck Machine (ABM) rule

In this proposed algorithm, we focus on the bottleneck machine ($b \in \{1, 2, \dots, m\}$) to adjust the number of tardy jobs. This machine is defined as the machine that requires maximum sum of processing times of all jobs amongst all machines. We determine for each job its ready time to be processed on this machine and we derive its due date. Then, we can define two sets of jobs: a set of tardy jobs and a set of early jobs. The adjustment is done by sequencing the set of tardy jobs in a decreasing order of the tardiness values and the early jobs in an increasing order of the earliness values, then we concatenate the sequence of the early jobs to the set of tardy jobs to form the final sequence to be scheduled using the "*Algorithm C*" described later. It is described as follows:

Step 1.

- Determine the release date of job i on machine b

$$r_{i,b} = \left\{ \sum_{l=1}^{b-1} (p_{i,l} + \theta_{i,l}^{min}) \right\}$$

- Determine the due date of job i on machine b

$$d_{i,b} = d_i - \left(\sum_{l=b+1}^{m-1} \left(p_{i,l} + \theta_{i,l}^{min} \right) + p_{i,m} \right)$$

Step 2.

Determine two sets of jobs: set of early jobs (E) and set of tardy jobs (T) on the bottleneck machine.

$$T = \{i | r_{i,b} + p_{i,b} > d_{i,b}\} \text{ and we define } u_{i \in T} = r_{i,b} + p_{i,b} - d_{i,b}$$
$$E = \{i | r_{i,b} + p_{i,b} < d_{i,b}\} \text{ and we define } s_{i \in E} = d_{i,b} - p_{i,b} - r_{i,b}$$

Step 3.

Sequence the jobs in an increasing order of s_i $\forall i \in E$ to provide the sequence E_1

Sequence the jobs in a decreasing order of u_i $\forall i \in T$ to provide the sequence T_1

The final sequence (π) to be scheduled is defined by concatenating the sequence E_1 to $T_1 \Rightarrow \pi = T_1 + E_1$

The obtained sequence is scheduled using the following "*Algorithm C*"

Algorithm C

```
Let the set of tardy jobs $T = \{\varnothing\}$ and $w = 0$.
The first job is scheduled as soon as possible
a) Schedule the job in the first position of the sequence on the first machine
    $C_{\pi(1),1} = p_{\pi(1),1}$
b) Schedule successively the other operations of the job in the first position
    For $k = 1\ to\ m - 1$, do
        $C_{\pi(1),k+1} = C_{\pi(1),k} + \theta^{min}_{\pi(1),k} + p_{\pi(1),k+1}$
c) Verify if the job in the first position $\pi(1)$ is tardy or not
    If $C_{\pi(1),m} > d_{\pi(1)}$
    Then $T = T \cup \{\pi(1)\}$, $w = w + 1$
    End if
Step 6.
Schedule the other jobs as soon as possible
For $i = 2\ to\ n$, do
    a) Schedule the first operation of the job in the $i^{th}$ position in the sequence on machine 1
        $C_{\pi(i),1} = C_{\pi(i-1),1} + p_{\pi(i),1}$
    b) Schedule successively the other operations of the job in the $i^{th}$ position in the sequence with respect to the precedence constraints and minimal time lag constraints
        For $k = 1\ to\ m - 1$, do
            $C_{\pi(i),k+1} = max\{C_{\pi(i),k} + \theta^{min}_{\pi(i),k},\ C_{\pi(i-1),k+1}\} + p_{\pi(i),k+1}$
    c) Verify if the job in position $i$ $(\pi(i))$ is tardy
        If $C_{\pi(i),m} > d_{\pi(i)}$
        Then $T = T \cup \{\pi(i)\},\ w = w + 1$
The number of tardy jobs is $w$.
```

As it is shown from the "*Algorithm C*", the constructed schedule is feasible as the precedence and the minimal time lags constraints hold, and the job sequence follows the obtained permutation π. Indeed, it is not necessary to verify the minimal time lag constraints for the job in the first position as this one has none precedent and its operations are consequently placed exactly after the minimal time lags. Also, it is shown for the other jobs that their operations are scheduled as soon as possible with respect to the precedence and the minimal time lags constraints. The complexity of this algorithm is $O(nm)$.

4 Lower bounds

In this section, lower bounds are developed which are based on the logic-based Benders decomposition and Moore's algorithm.

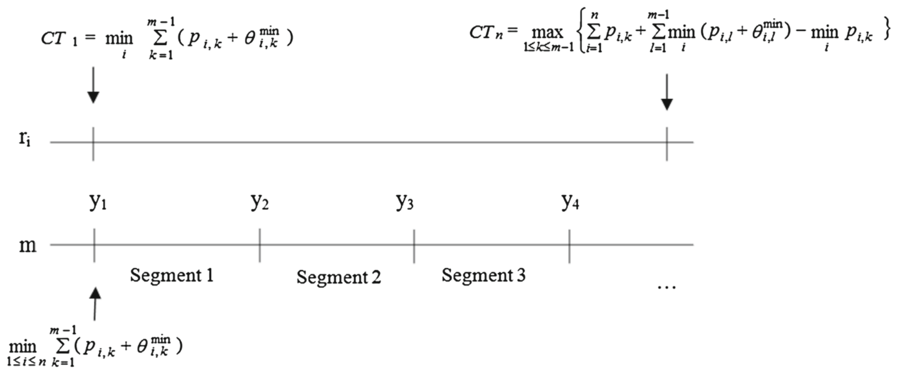
$CT_1 = \min_i \sum_{k=1}^{m-1} (p_{i,k} + \theta_{i,k}^{\min})$
$CT_n = \max_{1 \le k \le m-1} \left\{ \sum_{i=1}^{n} p_{i,k} + \sum_{l=1}^{m-1} \min_i (p_{i,l} + \theta_{i,l}^{\min}) - \min_i p_{i,k} \right\}$
r_i
y_1
y_2
y_3
y_4
m
Segment 1
Segment 2
Segment 3
...
$\min_{1 \le i \le n} \sum_{k=1}^{m-1} (p_{i,k} + \theta_{i,k}^{\min})$

Fig. 2 Horizon time

4.1 Logic-based Benders decomposition based-lower bounds (LB1)

In what follows, we will remind the literature of the logic-based Benders decomposition technique, then we detail its application. We will determine the associated $MILP$ of the studied problem, its CP, and an hybrid approach by combining the $MILP$ and the CP. The hybrid approach proves a great performance to generate results more interesting and less consuming time than the ones provided by only $MILP$ or CP. The motivation to use this hybrid method comes from its meaningful efficiency found by many researches over the literature ($e.g$ [14–16]). The found result by each one is a lower bound of the optimal solution for the considered problem.

The way to adopt this hybrid method is to define a long time horizon noted H ($H \geq C_{\pi(n),m}$) divided into J segments of time as it is shown in the following Fig. 2. These segments can be inequal, but we will consider them equal for the seek of simplicity. Here, the considered problem can be decomposed into an assignment portion and a scheduling portion. Central managers assign each job $i \in \{1, \dots, n\}$ to the time segment $j \in \{1, 2, \dots, J\}$, whereas operations managers prepare detailed schedules for each segment. As the done assignment can be infeasible, the operations managers are used to call the central managers to ask for different allocations of jobs. The assignment problem becomes the master problem and is solved by $MILP$, while the scheduling problem becomes the subproblem which separates into several independent scheduling problems is solved by CP. Hence, the discretization of the time segments is relevant and a new decision variable $x_{i,t}$ is introduced which is equal to 1 if the job i starts at time t and 0 else where N discrete times are defined starting from 0.

We are interested in a permutation flowshop scheduling problem where the jobs have to be processed on each machine in the same order. When we define a time horizon for each machine, it will be hard to specify non-overlapping time intervals for the machines $1, \dots, k-1$ in advance, without sacrificing optimality. Suppose jobs i and i' are assigned to intervals j and $j+1$, respectively, on machine k. Then a valid lower bound for the start time of job i' on machine $k-1$ would be less than a valid upper bound for the start time of job i on machine $k-1$. The intervals on machine $k-1$ would therefore overlap. One way to make this approach viable to the studied problem is to focus on the last machine m to define a long time horizon. Then, we determine a lower bound on the completion time of each job $i \in \{1, 2, \dots, n\}$ on the $m-1$ machine, this bound is considered as a release date of each job (r_i) to be processed on the last machine. r_i is generated from the interval $[CT_1,\ CT_n]$ where CT_1 is a lower bound on the completion time of the job in the first position; and CT_n is a lower bound on the completion time of the job in the last position. Where

$$CT_1 = \min_i \sum_{k=1}^{m-1} (p_{i,k} + \theta_{i,k}^{min}))$$

$$CT_n = \max_{1 \le k \le m-1} \left\{ \sum_{i=1}^{n} p_{i,k} + \sum_{l=1}^{m-1} \min_i \left(p_{i,l} + \theta_{i,l}^{min} \right) - \min_i p_{i,k} \right\}$$

Then when the job $i \in \{1, \dots, n\}$ is assigned to the segment of time $j \in \{1, \dots, J\}$, its starting time $(t \times x_{i,t})$ can't be smaller than its release date (r_i) which is supposed to be the amount of time necessary to its completion on the first $m-1$ machines (the lower bound on the completion time defined already). If the generated value (r_i) from the interval $[CT_1,\ CT_n]$ is greater than the time segment in which the job is assigned, it will be result in an infeasibility of this assignment and another one is used. y_1 and y_{J+1} are used to design the start time and finish time of the long time horizon. The starting time of the horizon time can be assumed by considering the minimum sum of processing times on machines 1 until $m-1$ plus the minimal time lags between them among all the jobs, so it can be calculated as $\min_{1 \le i \le n} (\sum_{k=1}^{m-1} p_{i,k} + \theta_{i,k}^{min})$ as same for lower bound on the completion time of the job in the first position.

4.1.1 Previous work

The classical Benders decomposition is useful for solving problems that contain groups of variables of different natures. However, it requires that the subproblem can be a continuous linear or nonlinear programming problem. Scheduling is a highly combinatorial problem that has no practical linear or nonlinear programming model [15]. The logic-based Benders decomposition generalizes classical Benders. It requires great interests recently, some trial values are assigned to the variables of the master problem (in our case, define the assignment of jobs to segments of time) and then we try to find the best solution with these values. In logic-based Benders decomposition, the central element is the derivation of Benders cuts which are obtained by solving the inference dual of the subproblem rather than a linear programming dual in classical benders. The solution of the inference dual is a proof of optimality when the variables of the master problem are fixed to their current values. So that, the process continues until the master problem and subproblem converge in value.

Logic-based Benders decomposition is introduced by Hooker and Yan [11] in the context of verifying logic circuits. Hooker [12] extends the Benders decomposition to a logic-based context for propositional satisfiability and 0-1 programming problem.

Hooker [13] shows how to derive effective Benders cuts for at least one such case, minimum makespan problems. Computational tests show that combining $MILP$ and CP to be 100 to 1,000 times faster than $MILP$ or CP when all tasks have the same deadlines. In a related work, Hooker [14] applies a logic-based Benders decomposition method to minimize the number of late tasks, and minimizing total tardiness. The main contribution is a relaxation of the cumulative scheduling subproblem, also much better solutions for problems that cannot be solved to optimality are obtained. Motivated by the complementary strength of the $MILP$ and CP, Coban and Hooker [2] apply logic-based Benders to minimize tardiness in single-facility scheduling problems with many jobs and long time horizons. Also, Hooker [15] applies the same method to solve an important class of planning and scheduling problems where the tasks assigned to a facility may run in parallel subject to resource constraints (cumulative scheduling). The objective is to minimize cost, makespan, or total tardiness. Then, significant computational speedups are obtained, of several orders of magnitude for the first two objectives, relative to the state of the art in both $MILP$ and CP.

Jain and Grossmann [16] successfully apply such a method to least cost planning and scheduling problems to process a set of orders using dissimilar parallel machines subject to release and due dates constraints where the subproblems are disjunctive scheduling problems (jobs must be run one at a time). According to computational results, the hybrid models are shown to perform two to three orders of magnitude reduction in CPU time compared to standalone $MILP$ and CP models.

Harjunkoski and Grossmann [10] consider multistage scheduling problems in which the subproblems are feasibility problems rather than optimization problems. Thus, the Benders cuts are generally simple as the subproblem is a feasibility problem rather than an optimization problem.

4.1.2 Mixed integer linear programming

The problem can be formulated as follows: each job $i \in \{1, 2, \ldots, n\}$ has to be assigned to a segment of time $j \in \{1, 2, \ldots, J\}$, p_i and d_i represent the processing time and the due date respectively of job i. The binary variable $x_{i,t}$ is already defined which is equal to 1 if the job i starts at time t, and 0 otherwise. t indicates the starting time of this job; then, to which segment job is assigned becomes trivial and the decision variable $x_{i,t}$ represent a segment. Here, we can refer to some proposed $MILP$ for other different problems and mainly the single machine scheduling problem (see i.g. [14,15]).

If we want modeling the scheduling problem as $MILP$, the model is:

$$Min \sum_i U_i \tag{1}$$

$$s.t \quad \sum_t x_{i,t} = 1 \quad \forall i \tag{2}$$

$$x_{i,t} + x_{\bar{i},\bar{t}} \leq 1 \quad for\, all\, i,\ t,\ \bar{i}\, and\, \bar{t} = t, \ldots, t + p_i - 1 \tag{3}$$

$$x_{i,t} = 0\ \forall t \leq r_i \quad and \quad t \geq y_{j+1} - p_i \quad \forall i \tag{4}$$

$$\sum_t (t + p_i)\, x_{i,t} - d_i \leq NU_i \quad \forall i \tag{5}$$

$$U_i \geq 0 \quad \forall i \tag{6}$$

$$x_{i,t} \in \{0,\ 1\} \quad \forall i,\ t \tag{7}$$

The objective (1) is to minimize the total number of tardy jobs ($\sum_i U_i$) where U_i is a binary variable equal to 1 if the job is tardy and 0 else. Each job has to be assigned to only one time interval is represented in constraints (2). Equation (3) prevents overlap from one segment to another one. Constraints (4) state that a job i can't start executing before its release date and cannot exceed the limit of the associated segment. Tardy jobs are defined in (5) where N is the number of discrete times starting with 0. Constraints (6) state that U_i is a non negative variable. Constraints (7) specify $x_{i,t}$ as a binary variable.

4.1.3 Constraint programming formulation

Recently, the constraint programming became a leading technique for solving complex decision support problems. In general, systems based on CP are much more expressive and hence easy to understand. It is increasingly used for solving scheduling problems as its flexibility is well suited for real-life scheduling problems. It works by incorporating some restrictions on the possible solution into a programming environment. Some recent works emphasize the CP application for different scheduling problems and mainly the single machine problem (see e.g. [13,14]).

Here, new decision variables are introduced: z_i is a decision variable representing the time interval of job's execution on the time segment, this definition is same of the processing time's job definition. Q_j is another decision variable defining the sequence of z_i time intervals at time segment j. It's known that this variable can be found easily with some software used to solve the CP. For example, jobs 2 and 3 are assigned to the segment $j = 2$ described by a start time y_2 and a finish time y_3; then Q_2 is defined as the sequence $Q_2 = 2,\ 3$. The CP for the problem can be formulated as follows:

$$Min\, sum\ (i\ in\ Jobs) U_i \tag{8}$$

$$s.t \quad StartOf(z_i) \geq r_i\ forall\, i \tag{9}$$

$$noOverlap\ (Q_j)\ forall\ j \tag{10}$$

$$(endOf\ (z_i) - d_i) \leq NU_i\ forall\, i \tag{11}$$

Minimizing the total number of tardy jobs is formulated by (8). Constraints (9) state that a job i can start only after its completion time on the precedent machines. By constraints (10) the sequence of jobs' time intervals cannot overlap on each machine as only one job have to be processed at a time. Constraints (11) define the tardy jobs. Some constraint-based tools are provided for imperative languages in the form of libraries. ILOG is one of the most successful companies to produce such tools. High level modeling languages exist for model constraint problems and specifying search strategies such that the OPL language.

4.1.4 The hybrid method

Logic-based Benders decomposition applies to problems of the form:

$$\begin{aligned} & Min\, f(x, y) \\ & s.t\, c(x,\ y) \\ & x \in D_x,\ y \in D_y \end{aligned} \tag{12}$$

$C(x,\ y)$ is a set of constraints containing variables $x,\ y$. $D_x$and D_y denotes the domains of x and y respectively. A general Benders algorithm begins by fixing x at some trial value $\overline{x} \in D_x$. This results in the subproblem:

$$\begin{aligned} & Min\, f(\overline{x},\ y) \\ & s.t\ \ c(\overline{x},\ y) \\ & y \in D_y \end{aligned} \tag{13}$$

The inference dual is the problem of inferring the tightest possible lower bound on $f(\overline{x},\ y)$ from $c(\overline{x},\ y)$. The inference dual of the subproblem is:

$$\begin{aligned} & Max\, \nu \\ & s.t\quad c(\overline{x},\ y) \rightarrow f(\overline{x},\ y) \geq \nu \end{aligned} \tag{14}$$

The dual problem is to find the best possible lower bound ν^* on the optimal cost that can be inferred from the constraints, assuming x is fixed to $\overline{x}$. This bound is expressed as a function $\nu_{\overline{x}}(x)$ of x yielding a Benders cut $z \geq \nu_{\overline{x}}(x)$. The algorithm proceeds as follows. At each iteration h, we solve a master problem where its constraints are the Benders cuts so far generated.

$$\begin{aligned} & Min\, z \\ & s.t\ \ \nu_{x^h}(x),\ h = 1, \ldots, H \\ & x \in D_x \end{aligned} \tag{15}$$

For each iteration h, we define x^h the solution of the h master problem from iteration 1 until $H - 1$ to provide $\nu_1^*, \ldots, \nu_{H-1}^*$ optimal solutions. Then the solution $\overline{x}$ of the previous subproblem to define the next one. z_h^* provides the lower bound of the optimal value of the problem and $\nu^* = min\{\nu_1^*, \ldots, \nu_{H-1}^*\}$. Then, the algorithm continues until $z_h^* = \nu^*$.

For this hybrid method, the scheduling problem which is the subproblem separates into several independent scheduling problems according to the number of the time segments. Let Y_j is the set of time intervals for J segments where $Y_j = \{t : y_j \leq t \leq y_{j+1} - 1\}$. The Benders approach formulates a master problem that assigns jobs to time segment j and a subproblem that schedules the jobs assigned to time segments. We write the master problem using an $MILP$ model that minimizes the number of tardy jobs. New decision variables are introduced in the master problem $U_{i,j}$ and U which represent the number of tardy jobs in the segment j and the total number of tardy jobs respectively. In iteration h of the Benders algorithm, the master problem is:

$$Min\, U \tag{16}$$
$$s.t \quad \sum_t x_{i,t} = 1 \quad \forall i \tag{17}$$
$$x_{i,t} = 0,\ \forall t \leq r_i \quad and\, t \geq y_{j+1} - p_i\ \forall i \tag{18}$$
$$\sum_{t \in Y_j} (t + p_i - d_i)\, x_{i,t} \leq N\, U_{i,j} \quad \forall i,\ j \tag{19}$$
$$U \geq \sum_i \sum_j U_{i,j} \quad \forall i,\ j \tag{20}$$
$$x_{i,t} \in \{0,\ 1\} \quad \forall i,\ t \tag{21}$$
$$U_{i,j} \geq 0 \quad \forall i,\ j \tag{22}$$
$$Benders\, cuts\, in\, iterations\ 1, \ldots, h-1 \tag{23}$$
$$relaxation\, of\, subproblem \tag{24}$$

The objective function (16) is minimizing the total number of tardy jobs. (17), (18), and (19) are same as the constraints of the MILP defined previously where Y_j is the set of job's time intervals. (20) and (22) are used to define the number of tardy jobs in the segment j. The main idea is to determine optimal assignments of jobs to time segments $t \in Y_j$, and then perform feasible sequencing of the jobs for the given assignments at a lower level. The binary variable $x_{i,t} = 1$ when a job i is assigned to the time interval t. Once, an assignment of jobs to time intervals is determined by solving the master problem; then, the scheduling subproblem is solved by CP. The subproblem decouples into a single-segment scheduling problem. Let J_{hj} the set of jobs assigned to segment j in the iteration h. Then, the subproblem becomes:

$$Min\, sum\, (i \in J_{hj})\, U_i \tag{25}$$
$$s.t\, startOf\, (z_i) \geq r_i \quad forall\, i \in J_{hj} \tag{26}$$
$$startOf\, (z_i) \geq y_j \quad forall\, i \in J_{hj} \tag{27}$$
$$endOf\, (z_i) \leq y_{j+1} \quad forall\, i \in J_{hj} \tag{28}$$
$$noOverlap\, (Q_j) \quad forall\, j \tag{29}$$
$$(endOf(z_i) - d_i) \leq NU_i \quad forall\, i \in J_{hj} \tag{30}$$

For each segment j, CP is solved separately. (25) and (26) are same as in the previous formulated CP. (27) and (28) state that the start and the end of each job's time interval have to respect the time limits of the respective segment. Constraints (36) and (37) are same as defined previously. $Let U = \{U(i)\}$, if $U^*_{hj}$is the optimal value of the previous problem, then $\sum_j U^*_{hj}$is the minimum number of tardy jobs for all the time segments.

We note $\underline{U_{hj}}$a valid lower bound on the number of tardy jobs. Since $x_{i,t} = 0$ when job i is not assigned to the time segment t we have

$$\underline{U_{hj}} \geq U^*_{hj} - U^*_{hj} \sum_{i \in J_{hj}} (1 - x_{i,t}) \quad \forall j \tag{31}$$

$$\underline{U_{hj}} \geq 0, \; all \; j \tag{32}$$

Over all the time intervals, we have a lower bound on the total number of tardy jobs:

$$U \geq \sum_j \underline{U_{hj}} \tag{33}$$

Then, the benders cuts (23) in the master problem for the iteration h consist of the inequalities (31), (32), and (33) obtained in the iterations $1, \ldots, h-1$. It is straightforward to relax the subproblem when minimizing the number of tardy jobs. As we described and defined above, let r_i considered as a lower bound on the completion time of job i on machines from 1 until $m-1$. When executed on the time interval j, these jobs span a time interval at least: $M = y_j + \sum^n_{i \in \{y_j,\, y_{j+1}\}} (r_i + p_i)$. We added $y_j$as each job i can't start earlier than the start time of the time segment.

If $M > y_{j+1} - y_j$ then at least one task is tardy, and the number of tardy jobs in the time interval k is at least

$$\frac{M - (y_{j+1} - y_j)}{\underset{i \in \{y_j,\, y_{j+1}\}}{max} \{r_i + p_i\}} \tag{34}$$

rounded up to the nearest integer. As the same way we determine the number of tardy jobs for each time segment. Then, a lower bound on the number of tardy jobs can be obtained for all the horizon.

The jobs have to be indexed according to their deadline ($d_1 \leq d_2 \leq \cdots \leq d_n$), $l = 1,\, 2 \ldots,\, n$. $J(0,\, d_l)$ is a set of jobs with time interval between 0 and d_l. For any j, the last scheduled job in this set can finish no sooner than time $t = \sum_{i \in J(0,\, d_l)} (r_i + p_i)$ and the last job has due date no later than d_l. Then, we can obtain the following relaxation:

$$U \geq \sum_i U_i$$

$$U_i \geq \frac{\sum^n_{i \in J(0,\, d_l)} (r_i + p_i) - d_l}{max_{i \in J(0,\, d_l)} \{r_i + p_i\}} \quad \forall i$$

which becomes (24) in the master problem.

4.2 Moore's algorithm based-lower bound (LB2)

Moore's algorithm is known to define the sets of tardy jobs and early jobs for the single machine scheduling. We adopt this algorithm to the studied problem to minimize the total number of tardy jobs. We define a sequence of early jobs σ and a set S of jobs not yet sequenced. $C(\sigma, k)$ is the earliest time at which machine k is available to process jobs of S. As we aim to define a single machine scheduling, we desaggregate the processing of jobs of S on machines $l = 1, 2, \ldots, k-1$ and we relax the constraints that machines $l = k+1, \ldots, m$ can process only job at a time. By considering a subproblem P_k on machine $k \in \{1, 2, \ldots, m\}$, each job $i \in S$ is available for processing at time $C(\sigma, k) = \min_{i \in S}\{\sum_{l=1}^{k-1}(p_{i,l} + \theta_{i,l}^{min})\}$, requires $p_{i,k}$ processing time on machine k, and requires time $t = \sum_{l=k+1}^{m-1}(p_{i,l} + \theta_{i,l}^{min})$ for its completion and has a due date d_i. Then, Moore's algorithm can be applied to generate a list of the tardy jobs $|T|$. It states to begin by sequencing jobs in increasing order of due dates, and then removing jobs with larger processing times. By applying the procedure to each machine, the lower bound on the total number of tardy jobs can be defined as $LB2 = max\{|T_1|, |T_2|, \ldots, |T_m|\}$. Then, the Moore's algorithm is applied as follows:

Step 1: sequence the jobs according to the Earliest Due Date rule to find the current sequence $(i_{\pi(1)}, i_{\pi(2)}, \ldots, i_{\pi(n)})$ such that $d_{\pi(1)} \leq d_{\pi(2)} \leq \cdots \leq d_{\pi(n)}$

Step 2: find the first tardy job, say $i_{\pi(i)}$ in the current sequence. If no such job is found. Go to step 4.

Step 3: reject longest job in $1 - i_{\pi(i)}$. Return to step 2 with a current sequence on job shorter than before.

Step 4: form an optimal schedule by taking the current sequence and appending to it the rejected jobs which may be sequenced in any order.

5 Computational results

In order to assess the quality of the proposed upper and lower bounds, we carry out series of experiments. Random instances are generated as follows: The processing times and the minimal time lags are generated as the same way in [6] from a uniform distribution between 20 and 50 and $[0, \theta^{min}]$ respectively where $\theta^{min} \in \{0, 7, 14\}$. The due dates are generated as $d_i = P_i \times D_{range}$, where $P_i = \sum_{j=1}^{m-1}(p_{i,j} + \theta_{i,j}) + p_{i,m}$ and $D_{range} = [0.8 - 1.2]$. The experimentations conducted in this research are run on a DELL PC/2.20GHz with 4.00Go RAM. The heuristic algorithms and the $LB2$ are implemented with MATLAB 7.6.

First experiments are done to test the performance of the proposed heuristic algorithms used to provide the upper bounds. We classify 7 different sizes of test problems, and we solve 10 problems for each size then the average value is considered. The 7 test problems consist of two different values of the number of machines (5 and 10) and 6 different values of the number of jobs (10, 15, 20, 30, 40, and 50). Here, the time lags interval is fixed to [0, 7]. The number of tardy jobs for each problem size and for each heuristic algorithm are shown in the following Table 2.

According to the results shown in Table 2, the three algorithms prove their efficiency to provide good results for different size problems. The three algorithms are almost

Table 2 Performance of the upper bounds

	(10, 5)	(10,10)	(15, 10)	(20, 5)	(30, 10)	(40, 5)	(50, 10)
SPT	6	7	13	15	22	31	43
$SSPT$	7	8	13	15	23	30	43
ABM	6	6	12	15	21	30	41

Table 3 Performance of the lower bounds

(n, m)	θ^{min}	UB		$LB1$	$LB1$					$Speedup$	$LB2$	
		Nb	CPU	$CP\,(\%)$	CPU	$MILP\,(\%)$	CPU	$Hybrid$ (%)	CPU		%	CPU
(10, 5)	0	6	0.06	0	0.12	50.5	0.14	2.5	0.09	1.55	50.3	0.08
	7	6	0.08	0	0.19	0	0.18	0	0.07	2.57	20.7	0.04
	14	7	0.10	16.1	1.22	4.3	1.11	14.3	0.85	1.30	0	0.13
(15, 5)	0	10	0.07	0	5.35	0	2.13	0	1.05	2.02	66.4	0.16
	7	11	0.09	22.3	2.43	22.7	1.13	10.6	0.96	1.17	57.6	0.17
	14	10	0.10	0	7.45	0	10.42	0	2.12	4.91	25.9	0.18
(15, 10)	0	12	0.11	9.2	13.25	33.5	9.51	0	0.95	10.01	50.6	0.16
	7	12	0.13	9.3	22.16	20.8	19.12	33.5	8.13	2.35	50.5	0.10
	14	12	0.11	0	44.00	0	38.13	9.4	20.34	1.87	0	0.09
(20, 5)	0	14	0.12	7.5	65.89	16.3	28.11	13.5	25.14	1.11	7.6	0.20
	7	15	0.13	0	198.45	0	55.12	0	33.12	1.66	50.4	0.36
	14	16	0.14	6.5	223.66	13.2	42.19	13.4	19.12	2.20	45.3	0.13
(25, 5)	0	17	0.12	13.8	>3,600	6.3	312.72	0.13	117.12	2.67	21.9	0.27
	7	18	0.14	0	>3,600	0	>3,600	0	87.13	>41.31	28.5	0.21
	14	19	0.16	11.3	>3,600	11.5	>3,600	0.11	123.58	>29.13	35.4	0.72

similar for the different tested problems just the one based on the ABM rule is the best one. As its principle consists in adjusting the number of tardy jobs in the bottleneck machine, it is not surprising to obtain the best found sequence. The algorithms are solved in less than 1 second and we could not distinguish a meaningful difference between all problems in the CPU time.

Second experiments are done to assess the performance of the proposed lower bounds. Let H be the long time horizon. The length of each segment is generated between $[10,\ H \times \alpha]$ where α is a small coefficient generated between [0.1, 0.5]. We set four different configurations of number of jobs $n \in \{10,\ 15,\ 20,\ 25\}$, and two different configurations for number of machines $m \in \{5,\ 10\}$.

15 problems classified according to the problem sizes and the time lags intervals are presented. 10 instances are tested for each problem and the average value is considered. The results are summarized in Table 3. To evaluate the proposed lower bounds, we determine the relative deviation (%) for each found solution from the upper bound (UB) provided by the ABM algorithm (as it is the best one) as follows

$\% = \frac{UB-LB}{LB} \times 100$. ($Nb =$ Number of tardy jobs). The Table 3 is described as follows: the first column indicates the problem size (n, m) where n is the number of jobs and m is the number of machines. The second column corresponds to the time lags interval $[0, \theta^{min}]$ where $\theta^{min} \in \{0, 7, 14\}$. The columns 3 and 4 correspond to the result of the upper bound for the different problem sizes and the CPU time needed to solve it respectively. The next columns from 4 to 10 indicate the deviation percentage of the lower bounds from the upper bound for the three methods CP, $MILP$, and the $hybrid$ one respectively where each of them is followed by a column indicating the corresponding CPU time needed to solve it. We note $LB1$ for all of them. Column 11 presents the speedup factor of the $MILP$ method relative to the hybrid one which is determined as $\frac{Time\,(MILP)}{Time\,(hybrid)}$. The two last columns deal with the second developed lower bound based on Moore's algorithm and the corresponding CPU time respectively. (The CPU time is given in seconds).

We use IBM ILOG CPLEX Optimization Studio to solve the $MILP$ (using CPLEX Optimizer), the CP (using ILOG CPLEX CP Optimizer), and the logic-based Benders method. All three methods are implemented using the OPL script language. We fix the time limit at 3600 s; then if the execution time overcomes this limit, we consider the value reached already at this time.

For each problem size, the number of tardy jobs increases with the increasing time lags intervals. It is an expected result as it is due to the increasing value of the completion times. In spite that the CP, the $MILP$, and the $hybrid\,method$ consume more CPU time than the $LB2$, they generate very tight lower bounds for some problems. We can confirm that the CP can provide the optimal solution with a percentage 47 %, the $MILP$ with a percentage equal to 40 %, and the hybrid method with 40 % too. For these three methods, the quality of the generated lower bounds is very interesting. The results of the hybrid method are obtained as the coincidence of both master and subproblem results developed for this method. According to Hooker [15], by linking $MILP$ and CP, we obtained substantial speedups relative to the existing state of the art in both $MILP$ and CP. The hybrid method is shown to be faster than the $MILP$ which is in turn faster than the CP. It is characterized by its ability to solve problems until the size (25, 5). Its fastness is proved by the speedup factor relative to the MILP which range between 1.11 and >41.31 as it is shown in the table above. The $LB2$ consume more less CPU time but the relative deviation is larger.

6 Conclusion

The permutation flowshop scheduling problem with minimal time lags to minimize the total number of tardy jobs is considered in this paper. Upper and lower bounds are developed. The upper bounds are given by three proposed heuristic algorithms based on different rules. The best one is used then to evaluate the quality of the developed lower bounds. $MILP$ and CP are proposed and then integrated through a logic-based Benders decomposition. They are shown to be efficient to provide tight lower bounds and the hybrid method is specified by its fastness against the CP and $MILP$ methods. The lower bound based on Moore's algorithm is shown to be easiest to be found. However, its quality is less interesting.

References

1. Chu, C., Proth, J.M.: Single machine scheduling with chain structured precedence constraints and separation time windows. IEEE Trans. Robot. Autom. **12**, 835–844 (1996)
2. Coban, E., Hooker, J.N.: Single-facility scheduling over long time horizons by logic-based benders decomposition. In: Proceedings of CPAIOR, pp. 87–91 (2010)
3. Croce, F.D., Gupta, G.N.D., Tadei, R.: Minimizing tardy jobs in a flowshop with common due date. Eur. J. Oper. Res. **120**, 375–81 (2000)
4. Deppner, F.: Ordonnancement d'atelier avec contraintes temporelles entre operations. Ph.D. Thesis, Institut National Polytechnique de Lorraine (in French) (2004)
5. Dhouib, E., Teghem, J., Loukil, T.: Minimizing the number of tardy jobs in a permutation flowshop scheduling problem with setup times and time lags constraints. J. Math. Model. Algorith. Oper. Res. **12**, 85–99 (2013)
6. Fondrevelle, J., Oulamara, A., Portmann, M.C.: Permutation flowshop scheduling problems with maximal and minimal time lags. Comput. Oper. Res. **33**, 1540–1556 (2006)
7. Ghassemi, T.F., Olfat, L.: A set of algorithms for solving the generalized tardiness flowshop problems. J. Ind. Syst. Eng. **4**, 156–166 (2010)
8. Graham, R., Lawler, E., Lenstra, J.K., Rinnooy Kan, A.H.G.: Optimization and approximation in deterministic sequencing and scheduling: a survey. Ann. Discrete Math. **5**, 287–326 (1979)
9. Hariri, A.M.A., Potts, C.N.: A branch and bound algorithm to minimize the number of late jobs in a permutation flowshop. Eur. J. Oper. Res. **38**, 228–237 (1989)
10. Harjunkoski, I., Grossmann, I.E.: Decomposition techniques for multistage scheduling problems using mixed-integer and constraint programming methods. Comput. Chem. Eng. **26**, 1533–1552 (2002)
11. Hooker, J.N., Yan, H.: Logic circuit verification by Benders decomposition. In: Saraswat, V., Van Hentenryck, P. (eds.) Principles and Practice of Constraint Programming: The Newport Papers, pp. 267–288. MIT Press, Cambridge (1995)
12. Hooker, J.N.: Logic-based Benders decomposition. Carnegie-Mellon University, Technical report (1995)
13. Hooker, J.N.: A hybrid method for planning and scheduling. In: Wallace, M. (ed.) Principles and Practice of Constraint Programming. Lecture Notes in Computer Science, vol. 3258, pp. 305–316. Springer, Berlin (2004)
14. Hooker, J.N.: Planning and scheduling to minimize tardiness. In: van Beek, P. (ed.) Principles and Practice of Constraint Programming. Lecture Notes in Computer Science, vol. 3709, pp. 314–327. Springer, Berlin (2005)
15. Hooker, J.N.: Planning and scheduling by logic-based benders decomposition. Oper. Res. **55**, 588–602 (2007)
16. Jain, V., Grossmann, I.E.: Algorithms for hybrid milp/cp models for a class of optimization problems. INFORMS J. Comput. **13**, 258–276 (2001)
17. Lenstra, J.K., Rinnooy Kan, A.H.G., Brucker, P.: Complexity of machine scheduling problems. Ann. Discrete Math. **1**, 343–362 (1977)
18. Lodree, E., Jang, W., Klein, C.M.A.: New rule for minimizing the nuımber of tardy jobs in dynamic flowshops. Eur. J. Oper. Res. **159**, 258–263 (2004)
19. Moore, J.M.: An n job one machine algorithm for minimizing the number of late jobs. Manage. Sci. **15**, 102–109 (1968)
20. Ruiz-Torres, A.J., Ablanedo-Rosasb, J.H., Johnny, C.H.: Minimizing the number of tardy jobs in the flow shop problem with operation and resource flexibility. Comput. Oper. Res. **37**, 282–291 (2010)