Minimizing the number of late jobs on parallel machines with alpha time windows

Vincent Jeauneau, Philippe Chrétienne

MISTA 2013

Minimizing the number of late jobs on parallel machines with α time windows

Vincent Jeauneau · Philippe Chrétienne

Abstract We consider the problem of minimization the number of late jobs on parallel machines in presence of α time windows ($d_i = r_i + p_i + \alpha$ for all jobs). We prove the NP-completeness of this problem even when $m = 2$ and $d_i = d$ for all jobs. We develop a lower bound by solving a minimum cost flow problem associated to a relaxation of the problem. A set of heuristics are presented to minimize the number of late jobs based on a job-focused and machine-focused approach. A simulation experiment is realized to test the effectiveness of heuristics regarding the lower bound. This simulation has been computed for $1, 2, 5, 10, 15$ and 20 machines and it shows that the best heuristic has an average deviation to the bound of 4.88%, 6.35%, 7.38%, 7.5%, 7.87% and 7.64% respectively.

Keywords Parallel machines scheduling, number of late jobs, α time windows, lower bound

1 Introduction

Given a set of jobs $J = J_1, ..., J_n$ where each job has a release date r_i, a due date d_i, and a processing time p_i. Given a set of parallel identical machines $M = M_1, ..., M_m$. An important scheduling problem is to determine a schedule that minimizes the number of late jobs ($P|r_i|\sum U_i$). A late job is defined as a job which finishes its processing later than its due date. This criterion is important since, in many case, the cost penalty incurred by a late job depends by the fact that it is late and not how much it is. This problem is NP-hard even when all jobs have the same release date and the number of machines is two [2] or the due dates are common [7].

In this paper, we consider a special case of the general parallel machines scheduling problem in which all jobs satisfy $d_i = r_i + p_i + \alpha$ with α a positive constant of the problem.

Vincent Jeauneau
Thales Air Systems, Technical Directorate
voie Pierre-Gilles de Gennes, 91470 Limours, France
E-mail: vincent.jeauneau@thalesgroup.com

Philippe Chrétienne
LIP6, Université Pierre et Marie Curie
4 place Jussieu, 75252 Paris Cedex 05, France
E-mail: philippe.chretienne@lip6.fr

Since each job cannot delay its starting time by more than α, we call our problem the parallel machines scheduling problem with α time windows. Following the standard three-field notation proposed by Graham et al. [4], this problem is noted $P|d_i = r_i + p_i + \alpha|\sum U_i$. For the one-machine case, Jeauneau and Chrétienne [6] extend Garey et al. works [3] and develop a polynomial time algorithm to minimize the number of late jobs optimally.

The rest of the paper is organized as follow. In Section 2, we prove the NP-Completeness of the problem. Section 3, presents a method to obtain a lower bound. Section 4 are dedicated of the development of some heuristics. Experimental results are analyzed in Section 5. Finally, Section 6 summarizes the paper.

2 On the complexity of the parallel machines scheduling problem with α time windows

In this section, we show that the scheduling problem of minimizing the number of late jobs on parallel machines in which all jobs satisfy $d_i = r_i + p_i + \alpha$ is NP-hard. This result come from a reduction of the NP-Hard problem *PARTITION* [8] to our problem. Consider the *PARTITION* problem, in which we are given a finite set A of positive integers having an overall sum $2b$ and are asked whether there exists a subset of A that sums exactly to b ?

From an instance of *PARTITION*, we generate the following instance of the parallel machines scheduling problem with α time windows:

- $m = 2$,
- $\alpha = b$,
- J_i such as $r_i = 0$, $p_i = a_i$, $d_i = p_i + b$ for $i = 1, ..., n$,
- J_{n+1} such as $r_{n+1} = 0$, $p_{n+1} = b + 1$, $d_{n+1} = 2b + 1$,
- J_{n+2} such as $r_{n+2} = 0$, $p_{n+2} = b + 1$, $d_{n+2} = 2b + 1$.

if *PARTITION* has a solution $\left(A^{'}, A^{''}\right)$, then there exists a schedule where the jobs of $A^{'}$ are scheduled on the first machine and the jobs of $A^{''}$ are scheduled on the second machine in the interval $[0, b]$. Then, the two additional jobs $\{J_{n+1}, J_{n+2}\}$ are scheduled in the interval $[b, 2b + 1]$, one on the first machine, the other on the second (Fig. 1). If the parallel machines scheduling problem has a solution, then the jobs $\{J_{n+1}, J_{n+2}\}$ cannot be scheduled on the same machine. Thus, one is scheduled on the first machine and the other on the second machine. Moreover, both can be scheduled in the interval $[b, 2b + 1]$, therefore the jobs scheduled in the interval $[0, b]$ form a solution of *PARTITION* problem. We prove that the parallel machines scheduling problem with α time windows is NP-hard even when the number of machines is two and all jobs have the same release date. It should be noted that a symmetric approach (fix the d_i-values instead of the r_i-values) proves the NP-Completeness of this problem even when all jobs have the same due date.

3 Lower Bound

In section 2, we have proved the NP-Completeness of the problem $1|d_i = r_i + p_i + \alpha|\sum U_i$. We will present some heuristics to solve it in section 4. In order to compare the results obtained from heuristics, we develop a lower bound in this section. To compute this lower bound, we first relax the constraint of non-preemption. This relaxed problem is noted $P|pmtn; d_i = r_i + p_i + \alpha|\sum U_i$ and every optimal solutions of this problem are less or equal than optimal solutions of $P|d_i = r_i + p_i + \alpha|\sum U_i$.

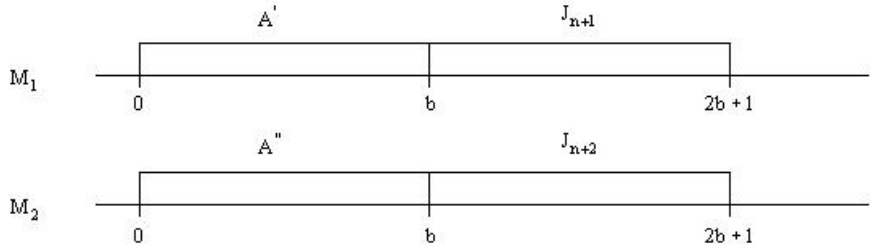

Fig. 1: Scheduling of jobs $\{J_1, ..., J_{n+2}\}$

Then we associate a minimum cost flow problem to the problem $P|pmtn; d_i = r_i + p_i + \alpha|\sum U_i$ such as solving the minimum cost flow problem leads to a lower bound for the problem $P|pmtn; d_i = r_i + p_i + \alpha|\sum U_i$ which is a lower for the problem $P|d_i = r_i + p_i + \alpha|\sum U_i$ too. Baker [1] solves the problem $P|pmtn; r_i|L_{\max}$ optimally with a similarly approach.

Let $J = \{J_1, ..., J_n\}$ be an instance of the problem $P|pmtr; d_i = r_i + p_i + \alpha|\sum U_i$ order as $d_1 \leq ... \leq d_n$. We define a minimum cost flow problem as follow:

- Let $t_1 < t_2 < ... < t_r$ be the ordered sequence of all different r_i-values and d_i-values.
- Let $I_K := [t_k, t_{k+1}]$ the intervals of length $l_k = t_{k+1} - t_k$ for $k = 1, ..., r-1$.
- A job vertex is associated to each job J_i and an interval vertex to each interval I_k.
- Let s a source and t a sink.
- There is an arc from s to each job vertex J_i of capacity p_i and cost 0.
- There is an arc from each interval vertex I_k to t of capacity $m \times l_k$ and cost 0.
- There exists an arc from J_i to I_k of capacity l_k and cost 0 iff job J_i can be processed in I_k (i.e $r_i \leq t_k$ and $t_{k+1} \leq d_i$).
- There is an arc from each job vertex J_i to t of capacity p_i and cost $1/p_i$.

Fig. 2 illustrate the graph $G(V, E)$ of the minimum cost flow problem defined above, where V is the set of vertices and E the set of arcs. Each arc is represented by a couple (x, y) where x is the capacity and y the unary cost of the arc.

A solution of maximum flow for such a graph $G(V, E)$ has a flow equal to $\sum_{i=1}^{n} p_i$. Indeed, for each job vertex J_i, there exists an arc (s, J_i) with a capacity p_i and another arc (J_i, t) with the same capacity. Since there only exits arcs from s to jobs vertices J_i, for $i = 1, 2, ..., n$, then we have a maximum flow solution in which the flow of arcs (s, J_i) and (J_i, t) are equal to p_i.

For each optimal solution of the minimum cost flow problem, we can build a schedule for the problem $P|pmtr; d_i = r_i + p_i + \alpha|\sum U_i$ as follow: for each interval vertex I_k, let x_{ik} denote the flow of arc (J_i, I_k), the job J_i executes a part of its processing time equal to x_{ik} in the interval of times $[t_k, t_{k+1}]$, for $i = 1, 2, ..., n$. Then, let θ_i denote the flow of arc (J_i, t), the job J_i executes, a part of its processing equal to θ_i, at the end of the schedule of an arbitrary machine, for $i = 1, 2, ..., n$.

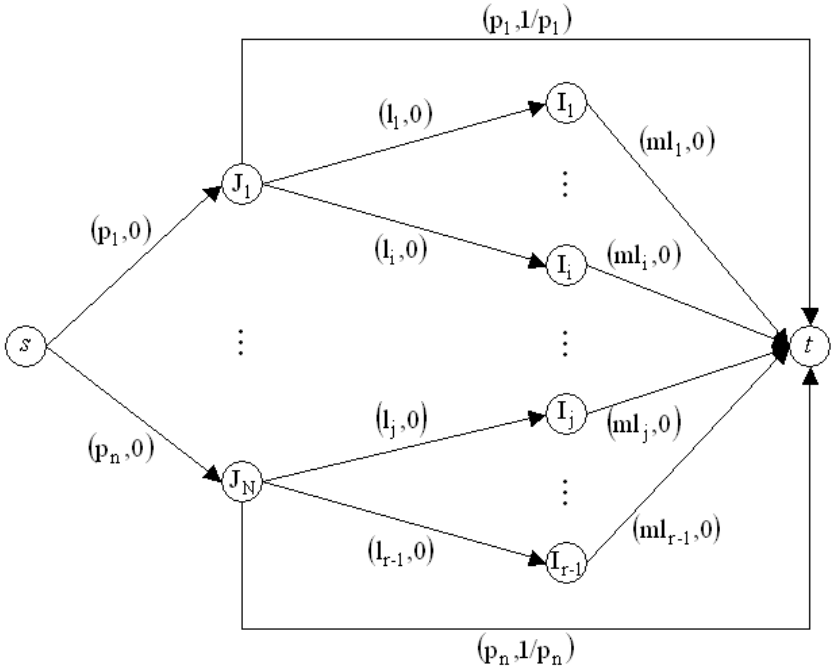

Fig. 2: Graph $G\,(V, E)$ of the minimum cost flow problem

The graph $G\,(V, E)$ is defined such as we have:

$$x_{ik} \leq l_j \text{ for } i = 1, 2, ..., n \text{ for } k = 1, 2, ..., r-1 \tag{1}$$

$$\sum_{i=1}^{n} x_{ik} \leq m \times l_k \text{ for } k = 1, 2, ..., r-1 \tag{2}$$

And from optimality of the minimum cost flow solution, we have:

$$\sum_{k=1}^{r_1} x_{ik} + \theta_i = p_i \quad \text{for } i = 1, 2, ..., n \tag{3}$$

From (1), we know that for each job vertex J_i and for each interval vertex I_k, the flow x_{ik} is less or equal than l_k. Since $l_k = t_{k+1} - t_k$, it implies that each job J_i cannot executes a part of its processing time greater than the length of the interval of time on which this part is executed. From (2), we know that for each interval vertex I_k we have $\sum_{i=1}^{n} x_{ik} \leq m \times l_k$. Therefore, the parts of processing times of all jobs J_i to be executed in $[t_k, t_{k+1}]$ have a total duration less or equal to $m \times l_k$. Since we dispose of m machines, these parts can be executed without overlapping whit another interval of time. From (3), we know that the flow of each job vertex J_i is exactly equal to p_i. Therefore, each job J_i are fully executed in the schedule.

We deduce that our schedule is a legal schedule for the problem $P|pmtn; d_i = r_i + p_i + \alpha| \sum U_i$ and we have the following property:

Property 1 An optimal solution of the minimum cost flow problem on the graph $G(V,E)$ is a lower bound of the problem $P|d_i = r_i + p_i + \alpha| \sum U_i$.

Proof Let Π an optimal schedule for an instance of the problem $P|d_i = r_i + p_i + \alpha| \sum U_i$. Let p_{ij} denote the j^{th} unity of p_i and c_{ij} its completion time in Π. We build a maximal flow f for the graph $G(V,E)$, associated to the problem instance, as follow:

- for each arc (s, J_i), there is a flow p_i;
- for each job J_i, if $c_{ij} \in [r_i, d_i]$ then increase by one the flow of arcs (J_i, I_k) and (I_k, t) such as $c_{ij} \in I_k$, otherwise increase by one the flow of (J_i, t).

The cost c of f is equal to $\sum_{i=1}^{n} y_i \times 1/p_i$, where y_i is the part of the job J_i which is not legally scheduled in Π, i.e $y_i = \sum_{j=1}^{p_i} p_{ij}$ such as $c_{ij} \notin [r_i, d_i]$. Then, for each job J_i, we have $y_i \leq p_i$ which imply $y_i \times 1/p_i \leq 1$. Moreover, when $U_i = 0$, the job J_i is not late scheduled, therefore $c_{ij} \in [r_i, d_i]$ for $j = 1, 2, ..., p_i$ and $y_i = 0$. Since $y_i \neq 0$ iff $U_i \neq 0$, we deduce that $\sum_{i=1}^{n} y_i \times 1/p_i \leq \sum_{i=1}^{n} U_i$.

From the definition of an optimal solution for the minimum cost flow problem, we have $c^* \leq c$ where c^* is the minimum cost of an optimal solution of maximum flow on the graph $G(V,E)$. We deduce that $c^* \leq \sum_{i=1}^{n} U_i$ and conclude that an optimal solution of the minimum cost flow problem on the graph $G(V,E)$ is a lower bound of the problem $P|d_i = r_i + p_i + \alpha| \sum U_i$.

Orlin [10] proposes a polynomial time algorithm for the minimum cost flow problem. This algorithm runs in $O(V \log V (E + V \log V))$. There is exactly n jobs vertices and at most $2n$ intervals vertices in $G(V,E)$, therefore the total number of vertices is in $O(n)$. There is n arcs between s and the jobs vertices, n arcs between jobs vertices and t, at most $2n$ between intervals vertices and t, and at most $n \times 2n$ arcs between jobs vertices and intervals vertices. Thus the total number of arcs is in $O\left(n^2\right)$. Therefore, the lower bound can be computed in $O\left(n^3 \log^2(n)\right)$.

We propose an amelioration of this lower bound by reducing the processing time of some jobs. Indeed, the value of the bound depends on the flow θ_i of each arc (J_i, t). The unary cost of these arcs is equal to $1/p_i$. Therefore, if we are able to reduce the processing time of a job J_i without change the flow θ_i, then the value of the lower bound will be increased.

We consider each interval vertex I_k, for k increasing for 1 to $r-1$. For each such interval vertex, we suppose that each job vertex J_i with an arc (J_i, I_k) has a flow exactly equal to l_k (this assumption is a relaxation on the capacity of arc (I_k, t)). Fixed these flows in $G(v,E)$ is equivalent to build the graph $G'\left(V', E'\right)$ where $V' = V - I_k$ and $E' = E - I_k^- - I_k^+$ with I_k^- and I_k^+ the inputs arcs and outputs arcs respectively. The capacity of arcs (s, T_i) and (T_i, t) for each job vertex J_i with an arc (J_i, I_k) in $G(V,E)$ is reduced by l_k in $G'\left(V', E'\right)$. For the graph $G'\left(V', E'\right)$, the cost of arcs (J_i, t) can be updated to the reverse of their capacities without violate the integrities constraints, i.e the flow of each job vertex J_i cannot increase the cost by more than one. Let c'^* and c^* denote the cost of the minimum cost flow solution for $G'\left(V', E'\right)$ and $G(V,E)$ respectively. If $c'^* > c^*$, then the quality of the solution for the graph $G'\left(V', E'\right)$ is better than the one for the graph $G(V,E)$ and the next intervals vertices $I_{k+1}, I_{k+2}, ..., I_{r-1}$ are considered on $G'\left(V', E'\right)$ otherwise continue on the procedure on graph $G(V,E)$.

Finally, the upper integer part $\lceil c^* \rceil$ can be used as a lower bound. Indeed, our aim is to minimize the number of late jobs. This objective is integer since $U_i = 1$ if J_i is late and $U_i = 0$ otherwise. Therefore the smallest integer which is greater than c^* define a lower bound, noted LB. This procedure solves a minimum cost flow problem for each interval

vertex I_k. There is no more than $2n$ intervals vertices, so this lower bound can be computed in $O\left(n^4 \log^2(n)\right)$.

It should be noted that this lower bound is independent of the structure of jobs and can be used for the general problem $P|r_i|\sum U_i$.

4 Heuristic approaches

In this section, we use two heuristic approaches to solve the m-machines scheduling problem with α time windows. These heuristic approaches are called job-focused and machine-focused. In the first approach, the jobs are considered separately and for each job we have to determine which machine will process it whereas in the second approach it is the machines which are considered separately.

4.1 Job-focused approach

The job-focused approach considers jobs one by one, while it considers all m identical machines simultaneously. This approach involves three major decision steps:

- A job section rule to select the current job.
- A machine selection rule to select which machine has to assign the current selected job (the first job of the ordered list).
- An interchanging/removing rule which removes or interchanges the current job, with an already assigned job, when it is late no matter which machine is selected.

When the current job is interchanged with an already assigned job, the already assigned job takes the place of the current job in the ordered list because maybe it can be processed on an other machine. To ensure that the algorithm finishes, it should have no cycle between interchanged jobs during this step. So, when a job is interchanging with another it cannot be interchanged again while we are interchanging jobs. The schedule of assigned jobs on a machine is determined from Jeauneau and Chrétienne works [6] which propose an algorithm to solve the one machine scheduling problem with α time windows optimally in $O\left(n^3\right)$. We call this algorithm ATW (Alpha Time Windows). In their works, the schedule is decomposed into blocks $B = \{B_1, ..., B_q\}$ such as B_i is a maximal sequence of consecutive jobs (no intermediate idle time). The following notation is used to define the heuristics procedures:

- m_k The machine k.
- U The set of unscheduled jobs.
- R The set of rejected jobs.
- I The set of interchanged jobs.
- σ_k The set of jobs assigned on m_k.
- C_i The completion time of the block containing the job J_i.
- C_i^k The completion time of the block containing the job J_i for the schedule of jobs $\sigma_k \cup J_i$

Fig. 4 details the heuristic $H1E$. Then, based on the first major decision step (job selection rule), we introduce $H1L$ which uses LPT (longest processing time) rules to select the current job. For the second decision step (machine selection), Jeauneau and Chrétienne

[6] shown that there is a strong interaction between jobs of the same block, so we choice to compare the completion time of the block containing the current job for the machine selection. Moreover, select the machine k with the greatest C_c^k has the advantage to load at most as possible an interval of time of a machine and provide more space of the others machines on this interval. Hence, this strategy is most able to handle jobs having a common part of their time windows. In the three decision step, when the current job cannot be assigned to a machine without cause tardiness of a job, we choice to interchange the current job with the job which will release the most space.

Step 1. Let $R = \emptyset$ and $I = \emptyset$. Let $\sigma_k = \emptyset$, for $k = 1, ..., m$. Place the job J_i in U for $i = 1, ..., n$.
Step 2. Select a job by EDD rule (earliest due date) from U, call it J_c (the current job).
Step 3. If adding the job J_c of the set σ_k of at least one machine k gives a feasible set of jobs (all jobs can be scheduled before their due dates in the sequence obtained by ATW $(\sigma_k \cup J_c)$), then remove all jobs of I and go to Step 5.
Step 4. Let J_{x_k} the late job obtained by ATW $(\sigma_k \cup J_c)$ for the machine k. Let J_{x_y} the job such as $C_{x_y} - C_c^y = \max_{1 \leq k \leq m} \left\{ C_{x_k} - C_c^k | J_{x_k} \notin I \right\}$. If the job J_{x_y} is the same as the job J_c then put it in R, remove all jobs of I and go to Step 6. Otherwise remove the job J_{x_y} from σ_y, put it in U and put the job J_c in I.
Step 5. Remove the job J_c from U and put it in σ_b, where $C_c^b = \max_{1 \leq k \leq m} \left\{ C_c^k \right\}$ and $\sigma_b \cup J_c$ is feasible.
Step 6. If $U = \emptyset$, then append the jobs of R in any set σ_k, and stop. Otherwise, go to Step 2.

Fig. 3: Heuristic $H1E$

Each job-focused heuristic run in $O\left(mn^4\right)$ time. Indeed, there is n jobs to schedule. Each job call m times the ATW algorithm. As this algorithm run in $O\left(n^3\right)$ time, we have an overall complexity for the job-focused heuristics in $O\left(mn^4\right)$.

4.2 Machine-focused approach

The machine-focused approach is the opposite of the job-focused approach. It considers the machine one by one, while it considers the set of unscheduled jobs simultaneously. Indeed, this approach solves m times the one machine scheduling problem. The first one machine scheduling problem involves all the jobs, whereas the second machine scheduling problem is solved with the set of not yet scheduled jobs (i.e the set of jobs scheduled on the first machine are not considered), etc. This process is repeated until there is no more machine or no more unscheduled jobs. ATW algorithm is used to solve each of the m one machine scheduling problems. Among all optimum solutions, this algorithm determine the one with minimum makespan (the completion time of the scheduling is minimized). So, for each machine we know that there is no schedule which process more early jobs, but it may exist some schedules which process the same number of jobs with a higher load of the machine. Hence, after running ATW, we try to interchange some jobs in order to increase the load of the machine. Thus, after schedule a set of jobs on a machine, we try to interchange each scheduled job with each not scheduled job having a longer processing time and we test the feasibility of this interchanging with GTW algorithm (algorithm of Garey, Tarjan and Wilfong [3] which determines if a set of job is feasible and, if yes, gives a solution with minimum makespan). We obtain the heuristic $H2$ detailed in Fig. 4.

Step 1. Let $k = 1$ (The first machine). Let $R = \emptyset$ and $\sigma_k = \emptyset$, for $k = 1, ..., m$. Place the job J_i in U for $i = 1, ..., n$.
Step 2. Set $\sigma_k = ATW\,(U)$.
Step 3. If the jobs in σ_k are all early, then go to Step 5.
Step 4. Remove all late jobs from σ_k and put them in U.
Step 5. If $k < m$ and $U \neq \emptyset$ then
  Step 5.1. Order the jobs in U by LPT rule and let $i = 1$ the index of the first job in this reordered set.
  Step 5.2. Let $\sigma_k^{'}$ the set of jobs of σ_k ordered by SPT rule and let $j = 1$ the index of the first job in this set.
  Step 5.3. If $GTW\left(\sigma_k^{'} - \left\{\sigma_k^{'}\,[j]\right\} + \{U\,[i]\}\right)$ is feasible, then add $U\,[i]$ in σ_k and remove it from U, remove $\sigma_k^{'}\,[j]$ from σ_k and insert it in U with respect to LPT order, and go to Step 5.2.
  Step 5.4. If $j < |\sigma_k^{'}|$, then set $j = j + 1$ and go to Step 5.3.
  Step 5.4. If $i < |U|$, then set $i = i + 1$ and go to Step 5.2.
  Step 5.5. Set $k = k + 1$ and go to Step 2.
Step 6. Append the jobs in U to any machines and stop.

Fig. 4: Heuristic $H2$

The heuristic H2 tries to minimize the number of late jobs machine by machine. By increasing the load on machine k, the next machine $k + 1$ will be able to process more early jobs. As the interchanging of jobs is used to increase the potential of processing more early jobs for the next machine, it is not relevant to do it for the machine m (the last machine). Therefore, we not apply it on the last machine. Moreover, when we interchange jobs, we increase the load of the machine. So, it is not necessary to retry to interchange jobs which have a longest processing time than the current interchanged job (if these jobs could have been interchanged, it will be done during their interchanging steps). Therefore, in Step 5.3 of H2, we not restart the interchanging rule from Step 5.1.

The machine-focused heuristic call m times the ATW algorithm to initiate the set of scheduled jobs on the current machine. Then, we apply the interchanging rule to this machine, it call at most n^2 times the GTW algorithm. As GTW runs in at most $(n \log n)$ steps, so H2 has an overall complexity in $O\left(mn^3 \log n\right)$ time.

5 Experimental results

Table 1: Results by machine machine level

m	$H1E$	$H1L$	$H2$	$H3$	LB	Avg. Error (%)
1	**4.3**	4.59	**4.3**	4.3	4.1	4.88
2	8.04	8.26	**7.76**	7.7	7.24	6.35
5	19.08	18.86	**17.83**	17.76	16.54	7.38
10	37.92	36.63	**34.96**	34.85	32.42	7.5
15	55.39	52.84	**50.68**	50.52	46.83	7.87
20	74.66	70.71	**68.16**	67.89	63.07	7.64

We develop a simulation experiment and the results are presented in this section. The number of machines is specified at six levels: 1, 2, 5, 10, 15 and 20 whereas the number of jobs is defined by the ration n/m. This ratio is set at four levels: 5, 10, 15 and 20. Processing

times are assumed to be $U(1, 99)$. Release dates follow $U(1, n\overline{p}/(mq))$, where $\overline{p}$ is the mean processing time, and q represents the level of congestion. The values of q are set at eight levels: 1.5, 2.0, 2.5, 3.0, 3.5, 4.0, 4.5 and 5.0. Ho and Chang [5] shown that this congestion indicator is useful in shop congestion. The shop congestion increases with q and the number of late jobs will be higher. The positive constant α of the problem is randomly computed from an uniform distribution $(1, n\overline{p}/(mq))$. As each job has to satisfy the relation $d_i = r_i + p_i + \alpha$, we calculate the due date d_i of each job from others parameters. We generate 50 instances of each problem set. Therefore, the total number of problems generated is 9,600. The heuristics $H1E$, $H1L$, $H2$ and the lower bound LB are implemented in JAVA in an Intel Core i7 2,4 GHz processor.

Table 2: Results by ratio n/m level

m	n/m	$H1E$	$H1L$	$H2$	$H3$	LB	Avg. Error (%)
1	5	**1.87**	2.07	**1.87**	1.87	1.79	4.47
	10	**3.54**	3.82	**3.54**	3.54	3.36	5.36
	15	**5.1**	5.43	**5.1**	5.1	4.87	4.72
	20	**6.68**	7.05	**6.68**	6.68	6.4	4.37
2	5	3.26	3.49	**3.1**	3.07	2.76	11.23
	10	6.54	6.75	**6.27**	6.22	5.8	7.24
	15	9.51	9.72	**9.17**	9.11	8.6	5.93
	20	12.85	13.08	**12.49**	12.4	11.8	5.08
5	5	7.6	7.86	**6.93**	6.87	5.71	20.31
	10	15.16	14.93	**13.99**	13.9	12.8	8.59
	15	23.41	23.07	**21.98**	21.9	20.59	6.36
	20	30.14	29.58	**28.44**	28.35	27.06	4.76
10	5	14.61	14.3	**12.82**	12.72	10.32	23.25
	10	30.50	29.33	**27.65**	27.56	25.21	9.32
	15	44.74	42.95	**41.23**	41.1	38.88	5.71
	20	61.84	59.93	**58.15**	58	55.28	4.90
15	5	21.39	20.19	**18.22**	18.1	14.31	26.49
	10	44.42	42.1	**39.94**	39.79	36.32	9.55
	15	67.09	64.17	**61.95**	61.76	57.99	6.5
	20	88.66	84.9	**82.61**	82.43	78.71	4.73
20	5	28.24	26.26	**23.92**	23.71	18.86	25.72
	10	59.38	55.43	**52.95**	52.7	48.11	9.54
	15	91.15	86.60	**83.93**	83.64	78.7	6.28
	20	119.89	114.53	**111.83**	111.49	106.59	4.6

The results are given according to the number of machines (Table 1), and then detailed according the ratio jobs/machines (Table 2) and the shop congestion level (Table 3). For each problem, the value of the lower bound LB is computed such as the values of heuristics. An additional heuristic $H3$ is proposed which takes at each times the best solution of H1E, H1L and H2. We also calculate the average deviation of $H3$ to LB, i.e $(H3/LB - 1) \times 100\%$.

These results show that heuristic $H2$ perform better than $H1E$ and $H1L$ and that $H1L$ perform better than $H1E$ for at most every cases.

Table 1 reveals that the average deviation of $H3$ to LB is equal to 4.88%, 6.35%, 7.38%, 7.5%, 7.87% and 7.64% for $1, 2, 5, 10, 15$ and 20 machines respectively. It is interesting to note that the average of $H3$ to LB stays stable when the number of machines increases. Indeed, for more than 2 machines the average deviation is around 7.5%. Moreover, when the number of machine is 1, heuristics $H1E$ and $H2$ solve the problem optimally ([6]). Therefore, in this case, we know that the lower bound LB has a deviation of 4.88% to

Table 3: Results by shop congestion level

m	q	$H1E$	$H1L$	$H2$	$H3$	LB	Avg. Error (%)
1	1.5	**1.36**	1.48	**1.36**	1.36	1.19	14.28
	2.0	**2.59**	2.79	**2.59**	2.59	2.39	8.37
	2.5	**3.59**	3.87	**3.59**	3.59	3.39	5.9
	3.0	**4.38**	4.7	**4.38**	4.38	4.18	4.78
	3.5	**4.92**	5.25	**4.92**	4.92	4.74	3.8
	4.0	**5.47**	5.83	**5.47**	5.47	5.28	3.6
	4.5	**5.87**	6.24	**5.87**	5.87	5.67	3.53
	5.0	**6.2**	6.58	**6.2**	6.2	6.01	3.16
2	1.5	2.54	2.49	**2.29**	2.2	1.78	23.59
	2.0	4.01	3.88	**3.65**	3.54	3.18	11.32
	2.5	6.53	6.67	**6.26**	6.19	5.74	7.84
	3.0	8.11	8.41	**7.77**	7.73	7.28	6.18
	3.5	9.44	9.76	**9.19**	9.19	8.69	5.75
	4.0	10.38	10.73	**10.12**	10.07	9.57	5.22
	4.5	11.15	11.57	**10.84**	10.81	10.34	4.54
	5.0	12.16	12.59	**11.95**	11.91	11.34	5.03
5	1.5	4.41	3.73	**3.63**	3.43	2.63	30.42
	2.0	10.93	10.02	**9.5**	9.33	7.96	17.21
	2.5	15.43	15.03	**14.25**	14.15	12.93	9.43
	3.0	19.05	18.82	**17.78**	17.74	16.55	7.19
	3.5	22.29	22.21	**20.99**	20.94	19.69	6.35
	4.0	24.69	24.83	**23.41**	23.37	22.1	5.75
	4.5	26.81	26.9	**25.35**	25.35	24.07	5.32
	5.0	29.01	29.34	**27.74**	27.73	26.4	5.04
10	1.5	8.02	**6.39**	6.50	6.02	4.54	32.6
	2.0	19.9	17.16	**16.58**	16.32	14.09	15.83
	2.5	30.89	29.03	**27.8**	27.7	25.48	8.71
	3.0	38.07	36.68	**34.72**	34.71	32.44	7
	3.5	44.95	43.67	**41.47**	41.43	39	6.23
	4.0	50.41	49.77	**47.31**	47.29	44.67	5.86
	4.5	53.56	52.67	**50.32**	50.3	47.47	5.96
	5.0	57.96	57.62	**55**	54.98	51.68	6.35
15	1.5	11.31	**8.18**	8.54	7.92	5.92	33.78
	2.0	27.50	22.81	**22.49**	22.02	19.05	15.59
	2.5	44.07	40.81	**39.245**	39.13	35.91	8.97
	3.0	55.37	52.87	**50.42**	50.37	47.29	6.51
	3.5	64.905	62.81	**59.93**	59.9	56.19	6.6
	4.0	73.81	72.03	**68.77**	68.76	64.5	6.6
	4.5	81.03	79.66	**76.16**	76.15	71.11	7.08
	5.0	85.1	83.56	**79.89**	79.89	74.68	6.98
20	1.5	15.44	**10.94**	11.50	10.69	8.32	28.49
	2.0	37.25	**30.3**	30.76	29.65	26.17	13.3
	2.5	60.65	56	**54.27**	54.12	49.54	9.24
	3.0	74.44	70.24	**67.41**	67.38	63.11	6.77
	3.5	90.14	86.84	**83.12**	83.1	78.19	6.28
	4.0	99.05	96.18	**91.85**	91.84	86.42	6.27
	4.5	107	104.47	**100.14**	100.13	93.83	6.71
	5.0	113.34	110.69	**106.2**	106.18	98.90	7.36

optimum. By assumption on the fact that the quality of the lower bound not increases when the number of machines increases, then H has a deviation $\approx 3\%$ to optimum.

Table 2 give the results according the number of machines which are detailed for th ratio n/m. Expected when the ratio $n/m = 5$, results show that the average deviation of H to LB is $< 10\%$. Moreover, if the quality of the bound not increases when the number of machines

increases, then H has a deviation to optimum of $\approx 4.5\%$, $\approx 2\%$ and $\approx 1\%$ for a ratio n/m equal to $10, 15$ and 20 respectively. For $n/m = 5$, results show a degradation of the average deviation when the number of machines increases. Indeed, the average deviation of H to LB is equal to 4.47%, 11.23%, 20.31%, 23.25%, 26.49% and 25.72% for $1, 2, 5, 10, 15$ and 20 machines. This degradation can be explained by: the degradation of the quality of LB when the number of machines increases for a small n/m ratio, the medium performance of heuristics when n/m is small. Globally, table 2 reveals the average deviation of H to LB decreases when the number of jobs increases.

Table 3 gives the results according the number of machines which are detailed for the shop congestion level. When $q \geq 2.5$, results show that the average deviation of H to LB is lesser than 10%. For these shop congestion levels, the gap between results of 1 machine case and 20 machines case is $\approx 4\%$. Therefore, if the quality quality of LB not increases when the number of machines increases, then we have an average deviation of H to optimum around 4%. For shop congestion levels 1.5 and 2.0, the average deviation of H to LB increases when the number of machines increases to reach a threshold of $\approx 34\%$ and $\approx 17\%$. This threshold is obtained for 15 and 5 machines respectively and then the degradation decreases when the number of machines increases. It should be noted that, for the one machine case, the quality of LB is yet medium which let assume that degradation of the average deviation is due to the poor quality of LB when the shop congestion level is low.

Table 4: Average CPU time

m	CPU time of $H3$ (s)	CPU time of LB (s)
1	0.001	0.002
2	0.003	0.024
5	0.015	0.768
10	0.069	8.114
15	0.165	37.99
20	0.308	99.08

The average CPU time needed to compute LB and $H3$ is presented in table 4. These results show that the average time to compute the heuristic H3 is very low although the worth case complexity is in $O\left(mn^4\right)$. For example, the mean time needed to solve an instance of 20 machines and 400 jobs is 754 milliseconds. When we consider LB, results reveals that the average time needed to compute this lower bound increases very quickly. For example, when the number of machines is 2, LB is computed has an average CPU time of 24 milliseconds whereas when the number of machines is 20, 99 seconds are needed in average.

6 Conclusions

We have studied the m parallel machines scheduling problem with α time windows in order to minimize the number of late jobs. We have proved the NP-completeness of this problem even when the number of machines is two, when all jobs have the same release date or when all jobs have the same due date. We have developed a lower bound for this problem which can also be used for the general parallel machines scheduling problem. To solve the problem, we have proposed a set of heuristics. A simulation experiment is conducted to evaluate the effectiveness of these heuristics. The results show that the average deviation to

the lower bound is equal to 4.88%, 6.35%, 7.38%, 7.5%, 7.87% and 7.64% for $1, 2, 5, 10, 15$ and 20 machines respectively. These reveals that qualities of the lower bound and heuristics are independent of the numbers of machines and globally the average deviation to the lower bound is $\approx 7.5\%$. Moreover, for the one machine, heuristics solve the problem optimally. Therefore, if the quality of the lower bound not increases when the number of machines increases, then our solution has an average deviation to optimum around 3%. Since the average CPU time needed to solve problems with 20 machines is around 300 milliseconds, our approach is able to handle large problems in a real time environment.

Acknowledgements This research was supported by French MoD DGA/MRIS and by Thales Air Systems. We also graceful to F. Barbaresco to give us the opportunity to work on this topic and to follow us during this study.

References

1. K.R Baker, Introduction to sequencing and scheduling, 108-110, Wiley, New York (1974).
2. M.R. Garey, D.S.Johnson, Computers and Intractability: A Guide to the Theory of NP Completeness, Freeman, San Francisco (1979).
3. M.R. Garey, R.E. Tarjan, G.T. Wilfong, One-processor scheduling with Symmetric Earliness and Tardiness, Mathematics of Operations Research, 13, 330-348 (1988).
4. R.L. Graham, E.L. Lawler, J.K. Lenstra, A.H.G Rinnooy Kan, Optimization and approximation in deterministic sequencing and scheduling: a survey, Annal of Discrete Mathematics, 5, 287-326 (1979).
5. J.C. Ho, Y.-L. Chang, Heuristics for minimizing mean tardiness for m parallel machines, Naval Research Logistics, 38, 367-381 (1991).
6. V. Jeauneau, C. Chrétienne, An $O\left(n^3\right)$ algorithm for the one machine scheduling problem with α Time windows to minimize the number of late jobs, preprint.
7. H.G. Kahlbacher, T.C.E Cheng, Parallel machine scheduling to minimize costs and number of tardy jobs, Discrete Applied Mathematics, 47(2), 139-164 (1993).
8. R.M. Karp, Reducibility among combinatorial problems, 85-103, Complexity of Computer Computations, R.E. Miller, J.W. Thatcher,Eds., New York: Plenum Press (1979).
9. J.M. Moore, An n-job one-machine sequencing algorithm for minimizing the number of late jobs, Management Science, 15, 139-164 (1968).
10. J. Orlin, A faster strongly polynomial minimum cost flow algorithm, Operations Research, 41, 338-350 (1993).