Lower bounds for the earliness-tardiness scheduling problem on single and parallel machines

Safia Kedad-Sidhoum
Yasmin Rios Solis
Francis Sourd*
Laboratoire LIP6
4, place Jussieu,
75 252 Paris Cedex 05, France

Abstract

This paper addresses the parallel machine scheduling problem in which the jobs have distinct due dates with earliness and tardiness costs. New lower bounds are proposed for the problem, they can be classed into two families. First, two assignment-based lower bounds for the one-machine problem are generalized for the parallel machine case. Second, a time-indexed formulation of the problem is investigated in order to derive efficient lower bounds throught column generation or Lagrangean relaxation. A simple local search algorithm is also presented in order to derive an upper bound. Computational experiments compare these bounds for both the one machine and parallel machine problems and show that the gap between upper and lower bounds is about 1%.

Keywords: Parallel machine scheduling, earliness-tardiness, Just-in-Time, lower bounds, IP time-indexed formulation.

1 Introduction

The twenty-year old emphasis on the Just-in-Time policy in industry has motivated the study of theoretical scheduling models able of capturing the main features of this philosophy. Among these models, a lot of research effort was devoted to earliness-tardiness problems—where both early completion (which results in the need for storage) and tardy completion are penalized. However, as shown by the recent surveys of T'kindt and Billaut [27] and Hoogeveen [13], most of this effort was dedicated to the one-machine problem. In this paper, we consider the earliness-tardiness problem in a parallel machine environment.

A set $\mathcal{J} = \{1, \cdots, n\}$ of n tasks are to be scheduled on a set of m identical machines. The single-machine case ($m = 1$) will be considered in the computational tests but no specific result is presented for this case. Let p_j and r_j respectively

*Corresponding author. E-mail: Francis.Sourd@lip6.fr

denote the processing time and the release date for job j. Each job j has also a distinct due date $d_j \geq r_j$. In any feasible schedule, C_j is the completion time of job j. If $C_j > d_j$, the job is said to be *tardy* and the tardiness is penalized by the cost $\beta_j T_j$ where $T_j = \max(0, C_j - d_j)$ and $\beta_j > 0$ is the tardiness penalty per time unit. Similarly, if $C_j < d_j$, the job is *early* and it is penalized by the cost $\alpha_j E_j$ where $E_j = \max(0, d_j - C_j)$ and $\alpha_j > 0$ is the earliness penalty per time unit. We also assume that all the release dates, due dates and processing times are integer, which ensures that there exists an optimal solution with integer start times. In the standard three-field notation scheme [10], this problem is denoted by $P|r_j|\sum_j \alpha_j E_j + \beta_j T_j$. The problem is known to be NP-complete even if there is only one machine and no earliness penalties [16].

Chen and Powell [6] study the special case where the jobs have an unrestrictively large common due date $d \geq \sum_j p_j$. This problem is formulated as an integer linear programming. By using column generation, a strong lower bound is derived and a branch-and-bound algorithm is proposed to solve the problem to optimality.

More recently, Ventura and Kim [29] study a related problem with unit execution time tasks and additional resource constraints. From the Lagrangean relaxation of a zero-one linear programming formulation of the problem, both lower bound and heuristics are derived. The authors use the property that the special case $P|r_j; p_j = 1|\sum_j \alpha_j E_j + \beta_j T_j$ is solved as an assignment problem.

In this paper, we study computational issues related to the use of standard mathematical formulations for machine scheduling problems in order to derive new lower bounds for the problem $P|r_j|\sum_j \alpha_j E_j + \beta_j T_j$. This computational analysis was particularly motivated by the use of time-indexed formulations [7] for which the bounds provided by the solution of LP-relaxation or Lagrangean relaxations are very strong [28]. We will focus on two of these formulations – namely the x_{jt}-formulation and the y_{jt}-formulation according to the terminology of Savelsbergh *et al.* [22]. In these formulations, $x_{jt} = 1$ means that job j starts at time t while $y_{jt} = 1$ means that it is in process at time t. These formulations have been useful in the design of strong lower bounds for problem with different regular criteria. The reader can refer to the works of Luh et al. [17], De Sousa and Wolsey [26], van den Akker et al. [28] for single machine scheduling problems. In a more theoretical approach, Queyranne and Schulz [21] study the polyhedral properties of such formulations.

The main contribution of this paper is to study these formulations for earliness-tardiness scheduling problems which are renowned for being hard to solve due to the difficulty of devising good lower bounds. The lower bounds tested on this paper are not all new —references are given in each section— but, to the best of our knowledge, they have not been tested and compared for earliness-tardiness problems. Some of the lower bounds as well as the heuristic algorithm can be considered as new since they are generalization to the parallel machine case of lower bounds and algorithms previously developed for the one-machine problem. Finally, experimental comparison of these algorithms is of importance because it helps choose the best algorithm in function of the problem parameters.

The paper is organized as follows. Section 2 provides lower bounds based on the linear and Lagrangean relaxations of time-indexed problem formulation. In Section 3, a new lower bound based on the single-machine bound of Sourd and Kedad-Sidhoum [25] is presented. The generalization of the bound of Sourd [23] is also introduced. Section 4 is devoted to a simple heuristic based on local search. and, in Section 5, we give some computational results which illustrate the effectiveness of the lower bounds. Some conclusions and extensions are finally discussed in Section 6.

2 Lower bounds based on the time-indexed formulation

2.1 Time-indexed formulation

We present an Integer Program (IP) for the problem $P|r_j|\sum_j \alpha_j E_j + \beta_j T_j$ with integer start times—we recall that all the release dates, due dates and processing times are integer so that there exists an optimal schedule with integer start times. We use the time-indexed formulation (or x_{jt}-formulation) [7]. It is based on time-discretization where time is divided into *periods* (or *time slots*), where period t starts at time t and ends at time $t+1$. Let T denote the scheduling horizon, thus we consider the time-periods $0, 1, 2, \cdots, T-1$. A simple interchange argument shows that there is an optimal schedule that completes before $T^\star = \max_j d_j + \max_j p_j + \left\lceil \frac{\sum_j p_j}{m} \right\rceil$ (we use the assumption $d_j \geq r_j$): we can indeed suppose that, in an optimal schedule, there is no idle period after $\max_j d_j$, so, if a job completes after $T^\star$ on a machine, it can be processed earlier by another machine. So we will consider that $T = T^\star$. In general, an optimal schedule completes much before T so that this discretization is not good. We will show in Section 2.2 a way to remedy this problem.

Let x_{jt} be a binary variable equal to 1 if the task j starts at time t and 0 otherwise. Let $[t, t']$ denotes the set of the discrete instants between t and t' and let $est_j(t) = \max(r_j, t-p_j+1)$ denote the earliest start time of j such that it is processed in time slot t. Let us also define the *start cost* $c_{jt} = \max(\alpha_j(d'_j - t), \beta_j(t - d'_j))$ where $d'_j = d_j - p_j$ is the *target start time* of task j. The time-indexed formulation of the problem is

$$
\begin{aligned}
\min \quad & \textstyle\sum_{j\in\mathcal{J}} \sum_{t=r_j}^{T-p_j} c_{jt} x_{jt} & & (1)\\
\text{s.t.} \quad & \textstyle\sum_{t=r_j}^{T-p_j} x_{jt} = 1 & \forall j \in \mathcal{J} & \quad (2)\\
& \textstyle\sum_{j\in\mathcal{J}} \sum_{s=est_j(t)}^{t} x_{js} \leq m & \forall t \in [0, T-1] & \quad (3)\\
& x_{jt} \in \{0,1\} & \forall j \in \mathcal{J},\, \forall t \in [r_j, T-p_j] & \quad (4)
\end{aligned}
$$

Equations (2) ensure that each job is proceeded once. Inequalities (3), also refered to as *resource constraints*, state that at most m jobs can be handled at any time. Clearly, this formulation allows the occurence of idle time. The integer program renders a solution that corresponds to an optimal schedule for the problem $P|r_j|\sum_j \alpha_j E_j + \beta_j T_j$.

The MIP solver ILOG CPLEX 9.0 is able to solve all our smallest instances with $n = 30$ jobs and $m = 2, 4, 6$ machines in less than one hour. In these test instances, the mean job processing time is about 50, clearly, for shorter processing times the formulation would be more efficient. For the single machine case, preliminary tests show that only instances with about 20 jobs can be solved within one hour. So, for larger instances (or if CPU time is limited), a relaxation of the formulation must be considered.

2.2 Linear relaxation with column generation

An important advantage of the x_{jt}-formulation is that the linear relaxation obtained by dropping the integrality constraints (4) provides a strong lower bound which dominates the bounds provided by other mixed integer programming formulations [28]. A major drawback of this formulation is its size. For instance, in our preliminary tests, the linear relaxation of our 50-job instances cannot be solved due to lack of memory.

In order to overcome this difficulty, we tested and compared two classical remedies. This subsection is devoted to *column generation* and the next two subsections presents two different *Lagrangean relaxations*.

The x_{jt}-formulation has $O(nT)$ binary variables but it can be observed that an optimal solution has only n variables set to 1. In a solution of the linear relaxation, most variables are also null. Therefore, we implemented the following column generation algorithm to help ILOG CPLEX solve the linear relaxation. A good feasible schedule $(S_1, \cdots, S_n)$ is first computed with the heuristic described in Section 4. The linear program restricted to the n variables x_{jS_j} (for $1 \leq j \leq n$) is initially considered and solved in order to get the reduced costs of the variables x_{jt} that have not been added to the linear program yet. All the variables with nonpositive reduced costs are added to the program and the procedure is iterated until there is no variable with a negative reduce cost. The linear relaxation is then solved.

According to our tests, the efficiency of the algorithm is improved with the following modification. All the variables whose reduced cost is less than a small value (equal to 5 in our implementation) are added instead of adding only variables with nonpositive costs. In this way, the number of iterations and the computation time are significantly decreased.

2.3 Relaxing the number of occurences

Another way to cope with the difficulty of the x_{jt}-formulation is to consider the Lagrangean relaxation of the equalities (2), which means that a job can be allowed to be processed several times in the relaxed problem. This approach is very related to the one proposed by Péridy *et al.* [20] for the one-machine problem in order to minimize the weighted number of late jobs. However, we do not generalize their so called short term memory technique which would be too time-consuming for our parallel machine problem.

We introduce a Lagrangean multiplier μ_j for each constraint (2). For each vector $\mu = (\mu_1, \cdots, \mu_n)$, a lower bound denoted by $\mathrm{LR}_1(\mu)$ is obtained by solving the

Lagrangean problem

$$\min_{x_{jt}} \quad \sum_{j\in\mathcal{J}} \sum_{t=r_j}^{T-p_j} (c_{jt}-\mu_j)x_{jt} + \sum_{j\in\mathcal{J}} \mu_j \qquad (5)$$

subject to the resource constraints (3) and (4)

The solutions of this dual problem represent schedules in which jobs satisfy the resource constraints but they can be processed several times, exactly once, or not at all. Similarly to van den Akker *et al.* [28], we will refer in the sequel to such schedules as *pseudo-schedules*.

We now show that the dual problem can be solved as the following network flow problem. The so called *time-indexed graph* G_T is defined as a digraph in which the nodes are the time periods $0, 1, \cdots, T-1$ plus a node representing the horizon T. For each variable x_{jt}, we define a "*process*" arc between node t and node $t+p_i$ with a cost $c_{jt}-\mu_j$ and a unit capacity and, for each node $t < T-1$, we define an "*idle*" arc between t and $t+1$ with a null cost and a capacity m.

Clearly, there is a one-to-one relation between integer m-flows from 0 to T in G_T and the pseudo-schedules of the m machine scheduling problem: a (unit) flow in the arc $(t, t+p_j)$ corresponds to processing job j between t and $t+p_j$. Moreover, the cost of the flow in the arc and the cost of starting j at t are equal so that the minimum cost flow of capacity m renders $\mathrm{LR}_1(\mu)$.

This is a generalization of the work of Péridy *et al.* [20] for the one-machine problem: when $m = 1$, the minimum cost integer flow is a shortest path from 0 to T, which corresponds to the shortest path problem of the Lagrangean relaxation of Péridy *et al.* It can be observed that solving the Lagrangean problem can also be seen as the problem of coloring an interval graph with a set of m colors such that the total weight is minimum. The nodes of the graph correspond to the intervals $[t, t+p_j)$ in which the jobs are possibly processed. In the m-coloring, two intersecting intervals must receive distinct colors among the m available ones. This problem is described and solved by Carlisle and Lloyd [4].

The function $\mathrm{LR}_1(\mu)$ has now to be maximized in order to get the best possible lower bound. This can be made through the subgradient method. Finally, we observed that the Lagrangean problem (5) returns integral solutions even if the integrality constraints are relaxed. Therefore, $\max_\mu \mathrm{LR}_1(\mu)$ is equal to the linear relaxation of Section 2.2, that is the Lagrangean relaxation cannot find better lower bounds than the linear relaxation but may eventually find them in less CPU time by using the structure of the network flow.

2.4 Relaxing the resource capacity constraints

We now study the Lagrangean relaxation of the resource constraints (3) of the x_{jt}-formulation. We introduce a Lagrangean multiplier $\mu_t \geq 0$ for each constraint. This Lagrangean relaxation is presented by Luh *et al.* [17] for the minimization of the sum of weighted tardiness. It can be noted that this approach can be extended to deal with precedence constraints even if the Lagrangean problem becomes more complex: in the context of job-shop scheduling, Chen *et al.* [5] study the in-tree precedence constraints and, for the Resource Constrained Project Scheduling Problem, Möhring *et al.* [18] address the Lagrangean problem with a general precedence graph.

For each vector $\mu = (\mu_0, \cdots, \mu_{T-1}) \geq 0$, a lower bound denoted by $\mathrm{LR}_2(\mu)$ is obtained by solving the Lagrangean problem

$$\min_{x_{jt}} \quad \sum_{j \in \mathcal{J}} \sum_{t=r_j}^{T-p_j} c_{jt} x_{jt} + \mu_t \left(\sum_{s=est_j(t)}^{t} x_{js} - m \right) \tag{6}$$

subject to the constraints (2) and (4)

This problem can be decomposed into n independent problems (one problem per job). Ignoring the constant term, we have to minimize $\sum_t \left(c_{jt} + \sum_{s=est_j(t)}^{t} \mu_s \right) x_{jt}$ for all $j \in [1, n]$. For each j, an obvious solution consists in setting to 1 the variable x_{jt} with the smallest coefficient and letting the other variables x_{jt} to 0. Therefore, the Lagrangean problem is solved in $O(nT)$.

As for the previous Lagrangean relaxation, $\mathrm{LR}_2(\mu)$ is maximized through the subgradient method and again the integrality property of the Lagrangean problem shows that $\max_\mu \mathrm{LR}_2(\mu)$ is equal to the linear relaxation value.

3 Assignment-based lower bounds

3.1 Assignment-based IP formulation

We now consider the assignment-based formulation or y_{jt}-formulation. This formulation assumes the same time-discretization as for the x_{jt}-formulation in Section 2.1. Here, y_{jt} is a binary variable equal to 1 if the task j is processed in period t and 0 otherwise. However, we will see in Section 3.3 how the discretization can be avoided.

The rationale for this formulation is to regard the scheduling problem as the assignment of unit task segments to unit time slots. The idea originates in the article of Gelders and Kleindorfer [9] for the single machine weighted tardiness problem. Sourd and Kedad-Sidhoum [25] and Bülbül *et al.* [3] have independently generalized this approach for the earliness-tardiness one-machine problem. We show that the approach can also be generalized to the parallel machine case. In the following y_{jt}-formulation, the additional binary variables z_{jt} indicate that a new block of job j starts at time t (which means that $z_{jt} = 1$ if and only if $y_{jt} = 1$ and $y_{jt-1} = 0$).

$$\min \quad \textstyle\sum_{j \in \mathcal{J}} \sum_{t=r_j}^{T-p_j} c'_{jt} y_{jt} \tag{7}$$

$$\text{s.t.} \quad \textstyle\sum_{t=r_j}^{T-p_j} y_{jt} = p_j \qquad \forall j \in \mathcal{J} \tag{8}$$

$$\textstyle\sum_{j \in \mathcal{J}} y_{jt} \leq m \qquad \forall t \in 0, \cdots, T-1 \tag{9}$$

$$z_{jr_j} \geq y_{jr_j} \qquad \forall j \in \mathcal{J} \tag{10}$$

$$z_{jt} \geq y_{jt} - y_{jt-1} \qquad \forall j \in \mathcal{J}, \forall t \in \{r_j + 1, \cdots, T - p_j\} \tag{11}$$

$$\textstyle\sum_{t=r_j}^{T-p_j} z_{jt} = 1 \qquad \forall j \in \mathcal{J} \tag{12}$$

$$y_{jt}, z_{jt} \in \{0, 1\} \qquad \forall j \in \mathcal{J}, \forall t \in \{r_j, \cdots, T - p_j\} \tag{13}$$

Let us first observe that the objective function is reformulated. The cost of a task does not depend on its completion time but is split among all the time slots when it is processed. Then, the assignment costs c'_{jt} to schedule a part of job j in time slot t must satisfy

$$\sum_{s=t-p_j}^{t-1} c'_{js} = \max\left(\alpha_j(d_j - t), \beta_j(t - T_j)\right) \quad \forall t\ \forall j \in \mathcal{J} \tag{14}$$

and a simple solution proposed by Sourd and Kedad-Sidhoum [25] is

$$c'_{jt} = \begin{cases} \alpha_j \left\lfloor \frac{d_j - t - 1}{p_j} \right\rfloor & \text{if } t < d_j, \\ \\ \beta_j \left\lceil \frac{t + 1 - d_j}{p_j} \right\rceil & \text{if } t \geq d_j. \end{cases} \tag{15}$$

Equations (8) ensure that each job is entirely executed between r_j and T, constraints (9) state that at most m jobs are in process at any time. Equations (10) and (11) define z_{jt} such that it is equal to 1 when a block of job j starts at t and equations (12) force each job to be processed in only one block, that is without preemption.

This formulation has more variables and more constraints that the x_{jt}-formulation, which means that only small instances can be solved by ILOG CPLEX.

3.2 Discrete lower bounds

In the rest of the paper, we relax all the constraints related to the z_{jt} variables. Therefore, in other words, we consider a preemptive relaxation of the problem. Note however that the objective function has been reformulated so that the problem is *not* $P|pmtn, r_j|\sum \alpha_j E_j + \beta_j Tj$. Indeed, the latter problem can be shown to be equivalent to $P|pmtn, r_j|\sum w_j Tj$, which means that the earliness costs are relaxed when "usual" preemption is allowed.

Clearly, the relaxed problem (7) subject to (8), (9) and (13) is a minimum cost flow problem in a bipartite network $\mathcal{N}(n, T)$ (see Figure 1). The n sources are the n jobs and the supply of source i is at most p_i. There are T sinks with a demand at most m. Any source j is linked to any sink $t \geq r_j$ by an arc with a unit capacity and a cost equal to c'_{jt}. Since T has been chosen large enough, the maximum flow in $\mathcal{N}(n, T)$ is equal to $P = \sum_j p_j$. The solution of the relaxed problem, and thus a lower bound for our problem, is the minimum cost P-flow.

When $m = 1$, Sourd and Kedad-Sidhoum [25] show that the flow problem can be solved in $O(n^2 T)$ time by adapting the well-known Hungarian algorithm [15]. For $m > 1$, the problem can be efficiently solved by the algorithms of Ahuja *et al.* [1] that specialize several classical flow algorithms for the bipartite networks where the number of sinks is much larger than the number of sources (they are called *unbalanced bipartite networks*).

It was observed in Section 2.1 that T is usually much larger than the makespan of the schedule, which means that the flow going to the sinks with the highest indices

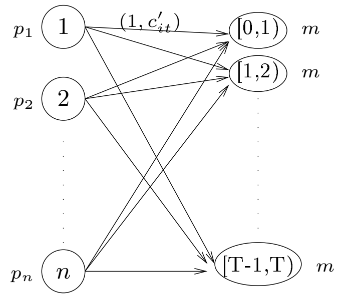

Figure 1: The discrete assignment network.

is null in general. Let us here consider that T is no more equal to $T^\star$ but satisfy $\max_j d_j < T < T^\star$ and let us assume that, in an optimal flow in $\mathcal{N}(n,T)$, there is no flow going to the sink $T-1$. As the assignment costs are non-decreasing when T increases (for $T > d_j$), a simple interchange argument shows that the flow in $\mathcal{N}(n,T)$ is also an optimal flow for $\mathcal{N}(n,T^\star)$. This suggests to first run the flow algorithm for some $T \in (\max_j d_j, T^\star)$ and to iteratively add sinks until an optimal flow with no flow going to the last sink is null. In our tests, we start with $T = \min(T^\star, 3/2 * \max(\max_j d_j, \max_j r_j + p_j, \sum_j p_j))$. Very often, the optimality can be proved at the first step of the algorithm.

Bülbül *et al.* [3] relax the equality (14) and propose assignment costs that satisfy the inequality $\sum_{s=t-p_j}^{t-1} c'_{js} \leq \max\left(\alpha_j(d_j - t), \beta_j(t - T_j)\right)$:

$$c'_{jt} = \begin{cases} \frac{\alpha_j}{p_j}\left((d_j - p_j/2) - (k - 1/2)\right) & \text{if } t < d_j, \\ \frac{\beta_j}{p_j}\left((k - 1/2) - (d_j - p_j/2)\right) & \text{if } t \geq d_j. \end{cases} \tag{16}$$

Clearly, the corresponding minimum cost flow is still a lower bound for the problem and experimental results show that this lower bound is often better than the lower bound proposed by Sourd and Kedad-Sidhoum. This variant in the definition of the assignment cost is also tested in Section 5.

3.3 Continuous lower bound

The computation of the lower bound presented above is not polynomial because the number of time slots is not polynomial in n. For the one-machine problem, Sourd [23] presents a similar lower bound that avoids to discretize the scheduling horizon. We show in this section how to adapt this approach for the parallel machine problem.

The main idea is that preemption of tasks is allowed at any time instead of constraining it to be at integer time points only. Therefore, instead of defining T values c'_{jt} for each task j, a piecewise linear function $f_j(t), t \in \mathbb{R}$ with only two segments is used to represent the assignment costs in a so-called *continuous assignment problem*. The proposed function is

$$f_j(t) = \begin{cases} -\frac{\alpha_j}{2} + \frac{\alpha_j}{p_j}(d_j - t) & \text{if } t \leq d_j, \\ \frac{\beta_j}{2} + \frac{\beta_j}{p_j}(t - d_j) & \text{if } t > d_j. \end{cases} \tag{17}$$

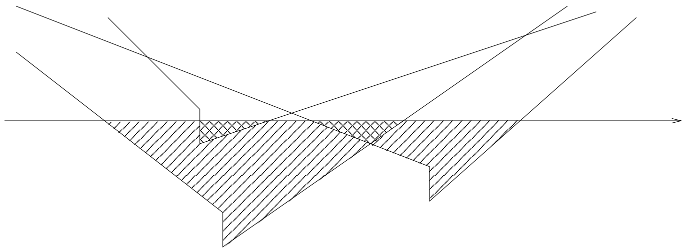

Figure 2: Continuous lower bound for $m = 2$ machines

It satisfies the inequality $\int_{t-p_i}^{t} f_i(s)\mathrm{d}s \leq \max(\alpha_j(d_j - t), \beta_j(t - T_j))$ which can be seen as a continuous relaxation of (14).

The counterpart of the decision variables x_{jt} are decision function variables $\delta_j(t)$ that must be piecewise constant with values in $\{0, 1\}$, $\delta_j(t)$ is equal to 1 when job j is in process at time t and 0 otherwise. Then, the continuous version of the problem in Section 3.2 is

$$\begin{aligned}
\min \quad & \textstyle\sum_{j\in\mathcal{J}} \int_0^\infty f_j(t)\delta_j(t)\mathrm{d}t && && (18)\\
\text{s.t} \quad & \textstyle\int_0^\infty \delta_j(t)\mathrm{d}t = p_j && \forall j \in \mathcal{J} && (19)\\
& \textstyle\sum_{j\in\mathcal{J}} \delta_j(t) \leq m && \forall t \geq 0 && (20)\\
& \delta_j(t) \in \{0, 1\} && \forall t \geq 0 \quad \forall j \in \mathcal{J} && (21)
\end{aligned}$$

In order to get a lower bound, we consider the Lagrangean relaxation of constraints (19). For a vector $\mu = (\mu_1, \cdots, \mu_n)$, we have

$$\begin{aligned}
\min \quad & \textstyle\sum_{j\in\mathcal{J}} \mu_j p_j + \sum_{j\in\mathcal{J}} \int_0^\infty (f_j(t) - \mu_j)\delta_j(t)\mathrm{d}t \\
\text{s.t} \quad & \textstyle\sum_{j\in\mathcal{J}} \delta_j(t) \leq m \qquad \forall t \geq 0
\end{aligned}$$

Therefore, we have independent problems for each time t and, by defining

$$\begin{aligned}
g_\mu(t) = \min \quad & \textstyle\sum_{j\in\mathcal{J}} (f_j(t) - \mu_j)\delta_j(t)\mathrm{d}t \\
\text{s.t} \quad & \textstyle\sum_{j\in\mathcal{J}} \delta_j(t) \leq m
\end{aligned}$$

the solution of the problem is $\sum_{j\in\mathcal{J}} \mu_j p_j + \int_0^\infty g_\mu(t)\mathrm{d}t$.

In other words, computing $g_\mu(t)$ consists in choosing at most m values in the multiset of real values $\{f_1(t) - \mu_1, f_2(t) - \mu_2, \cdots, f_n(t) - \mu_n\}$ such that the sum of these values is minimal. This can be achieved by sorting these values and selecting the m smallest values if there are at least m negative values or selecting all the negative values otherwise. Figure 2 gives a geometric illustration of the computation of $\int g_\mu(t)\mathrm{d}t$ for a two-machine problem. The integral is the sum of the two hatched areas.

Let us define, for simple notations, $f_0(t) = 0$ and $\mu_0 = 0$ and let us consider an interval $I \subset \mathbb{R}_+$ on which the functions $f_i - \mu_i$ $(i = 0, \cdots, n)$ do not crossover, that is for any $0 \leq i < j \leq n$, we have either $f_i(t) - \mu_i \neq f_j(t) - \mu_j$ for any $t \in I$ or $f_i(t) - \mu_i = f_j(t) - \mu_j$ for any $t \in I$. Since the indices of the m smallest negative values are invariant when t varies in I, the function $g_\mu(t)$ is linear on I. Therefore, we simply have to compute $g_\mu(t)$ for the time points where two functions intersect.

In order to determine both the time points where two functions intersect and the m smallest negative values, we use an immediate adaptation of the well-known sweeping algorithm to determine all the intersections between segments [2]. This algorithm runs in $O(n^2)$ so that $\int g_\mu(t)\mathrm{d}t$ can be computed in $O(n^2 m)$.

The function $\mu \mapsto \int g_\mu(t)\mathrm{d}t$ is a concave function that can be maximized through a subgradient method. By adjusting the proof of [23] to the present case, we can show that this maximal value is equal to the optimum of the mathematical program (18-21), that is there is no duality gap. Finally, instead of using the simple—and satisfactorily efficient—subgradient method, the optimum can be computed in polynomial time by the ellipsoid method (see [23] for details).

4 Feasible solutions

Typically, good feasible solutions can be derived from good relaxations of a problem. For the one-machine case ($m = 1$), Sourd and Kedad-Sidhoum [25] and Bülbül et al. [3] present different heuristic algorithms grounded on the assignment-based lower bound. These heuristics could be directly adapted for the general case ($m > 1$) and very similar heuristics could be derived from the other time-indexed lower bounds (see *e.g.* [22]). It can however be observed that computing these lower bounds is somewhat time-consuming. Therefore, we present in this section a simple local-search algorithm which runs faster than the computation of the lower bounds presented in the previous sections. Despite its simplicity, it is experimentally very efficient.

A feasible solution is represented by m lists (or sequences) of jobs which correspond to the sequencing of jobs on each machine. Clearly, these m lists form a partition of the job set into m subsets (any job is processed by one and only one machine). These lists are denoted by $(L_1, L_2, \cdots, L_m)$ and n_j denotes the number of jobs in L_j. The associated cost is the minimum cost with respect to the job/machine assignment and to the sequencing. It can be estimated by computing the optimal timing for each sequence, which can be done in polynomial time (see [13] for a review of the algorithms, we use the algorithm of Sourd [24]).

The neighborhood of a feasible solution is the union of three basic neighborhoods corresponding to the three following basic moves :

1. *job-swap*: select two jobs (processed by the same machine or not) and interchange their machine assignment and position in the sequence.

2. *extract and reinsert*: select one job, remove it from its list and reinsert it in any list at any position.

3. *list-crossover*: select two lists, say L_i and L_j and two positions $0 \leq \nu_i \leq n_i$ and $0 \leq \nu_j \leq n_j$. Replace them by a list formed by the first ν_i elements of L_i followed by the last $n_j - \nu_j$ elements of L_j and a list formed by the first ν_j elements of L_j followed by the last $n_i - \nu_i$ elements of L_i.

In order to speed-up the neighborhood search, we adapted the technique presented in [11, 12], which makes use of the dynamic programming approach of [24] to store some partial results that can then be re-used to compute the timing of several neighbor sequences. As the adaptation is obvious, we do not give more details and the reader is refered to the above references.

The descent algorithm starts with a random partition of the jobs into m subsets. Each subset is randomly sequenced so that a feasible initial schedule is obtained. Then the local search is iterated until a local minimum is found. This descent procedure is iterated several times—in our tests, 10 times or 100 times—starting with different random initial solutions. The best local minimum is eventually returned.

5 Computational Experiments

5.1 Algorithms and implementation

All of our algorithms are implemented in C++. Here is the list of implementations with some notes including the libraries used by each algorithm.

LinCG It implements the linear relaxation with column generation presented in Section 2.2. This algorithm calls ILOG CPLEX 9.0 to solve the linear problems.

LagOcc It implements the Lagrangean relaxation of the number of job occurences presented in Section 2.3. The network flow problem is solved with the library GOBLIN 2.6 [8].

LagRes It implements the Lagrangean relaxation of the resource constraints presented in Section 2.4.

AssSKS It implements the discrete assignment-based lower bound (Section 3.2) with the assignment costs defined by Sourd and Kedad-Sidhoum [25]. Surprisingly, preliminary tests have shown that the dual simplex algorithm of ILOG CPLEX is faster than GOBLIN, so we have kept the implementation based on ILOG CPLEX. The tests have shown that the implementation in which the algorithm is initialized with the complete network (*i.e.* with $T = T^\star$) is about 50% slower than the implementation based on "sink generation" described in Section 3.2. Therefore, we only consider the latter implementation in the tables.

AssBKY It is the variant of the above algorithm with the assignment costs defined by Bülbül *et al.* [3].

ContAss It implements the continuous assignment-based lower bound presented in Section 3.3.

Heur It implements the local search algorithm presented in Section 4. The algorithm is run 10 times.

The algorithms LagOcc, LagRes and ContAss use the subgradient method. For each algorithm, the method is stopped when 100 iterations are met without any improvement or when the time limit of 600 seconds is reached. Due to the column generation method, LinCG, AssSKS, AssBKY are ensured to have a lower bound only at the very end of the computation so no time limit is fixed.

5.2 Instances

We randomly generated a set of instances using usual parameters of the earliness-tardiness scheduling literature (see e.g. [14]). The processing times are generated from the uniform distribution $\{p_{\max}/10, \cdots, p_{\max}\}$ with $p_{\max} = 100$. Clearly, this parameter is of prime importance for the computation of our lower bounds because the size of the scheduling horizon discretization is significantly decreased for small values of $p_{\max}$. When $p_{\max} = 10$ or $p_{\max} = 20$, Bülbül *et al.* [3] show that computation times are indeed dramatically decreased for their assignment based lower bound. We also recall that Sourd [23] shows that the "continuous" lower bound remains competitive for very large $p_{\max}$ because it avoids the discretization. For simplicity of the comparison, all the release dates are equal to 0. The earliness and tardiness factors are randomly chosen in $\{1, \cdots, 5\}$. There are two parameters to compute the due dates, namely, the *tardiness factor* τ and the *range factor* ρ. Given τ, ρ and $\mathcal{J}$, the due date of each job j is generated from the uniform distribution $\{\theta_j, \cdots, \theta_j + \rho \sum_{k\in\mathcal{J}} p_k\}$ where $\theta_j = \max(r_j + p_j, (\tau - \rho/2)\sum_{k\in\mathcal{J}} p_k)$.

We generated job sets for n varying in $\{30, 60, 90\}$, $\tau \in \{0.5, 1\}$ and $\rho \in \{0.2, 0.6\}$. We then obtained 12 different job sets and we tested them for m equal to 1, 2, 4 and 6 so that each algorithm was run for 48 instances. Our choice for relatively small values for the tardiness factor τ was motivated by the fact that in most practical scheduling problems the due dates are not very distant and these instances are moreover computationally harder. Clearly, some of the algorithms could benefit from a special implementation for the special case $m = 1$ (for example, AssSKS and AssBKY can use the $O(n^2T)$ algorithm of [25]). As the single machine problem is not the main topic of this paper, we only compare here the general implementation for any value of m. To the best of our knowledge, the lower bounds derived from the x_{jt}- and y_{jt}-formulation have never been compared in the literature so that we think that the experience must be reported in our tests.

5.3 Results

The results for each instance and each algorithm are reported in Table 1. The first four columns indicate the instance parameters. The next column provides the best upper bound known for the instance which correspond to the best solution provided by ten runs of the algorithm Heur. The other columns present for all the tested algorithms the final result and the CPU time. The instance $(n = 30, \tau = 0.5, \rho = 0.6, m = 6)$ is shown by Heur to have a feasible solution at no cost so there is no need to run the lower bound algorithms. It can also be observed that there are missing

n	τ	ρ	m	Best	Heur		AssSKS		AssBKY		ContAss		LinCG		LagRes		LagOcc	
					UB	s	LB	s	LB	s	LB	s	LB	s	LB	s	LB	s
30	0.5	0.2	1	47293	47293	0.19	45771	7.72	47076	46.4	47082	42.5	47293	1038	47280	38.8	44611	600
			2	19194	19200	0.44	17876	2.52	18722	12.6	18973	30.5	19194	112	19191	5.54	14868	600
			4	5865	5865	0.47	5126	0.77	5255	2.32	5512	12.8	5865	31.0	5864	4.03	5835	399
			6	2066	2090	0.5	1606	0.41	1431	1.02	1583	8.79	2065	8.36	2059	1.68	2044	474
30	0.5	0.6	1	29767	29767	0.68	28356	13.3	29307	28.0	29492	53.9	29658	662	29640	43.9	25984	600
			2	6991	6991	0.67	6130	3.54	6322	4.05	6640	31.3	6920	59.4	6910	7.53	6583	600
			4	216	222	0.55	99	0.41	57	0.94	0	8.02	215	0.76	211	0.92	180	86.5
			6	0														
30	1.0	0.2	1	13780	13780	0.99	13083	10.8	13566	28.5	13395	74.3	13678	304	13599	48.2	13092	600
			2	5994	5995	1.16	5485	3.07	5820	4.9	5657	19.8	5977	89	5970	2.11	5966	600
			4	2264	2270	0.59	1976	1.33	1993	2.8	1805	49.4	2257	13.12	2254	2.7	2199	239
			6	975	983	0.59	802	0.91	725	2.14	511	6.73	974	2.2	973	4.12	915	96.1
30	1.0	0.6	1	12671	12671	0.69	11649	10.8	11943	22.8	12029	32	12398	319	12381	25.2	11626	600
			2	2898	2928	1.07	2404	2.48	2296	3.14	2441	14.8	2872	62.4	2865	2.29	2820	600
			4	350	356	0.62	170	0.69	28	1.73	35	8.1	346	0.63	345	1.59	342	78
			6	78	78	0.44	41	0.5	0	1.29	0	8.1	78	0.18	78	2.34	71	53.6
60	0.5	0.2	1	153955	153955	3.49	151133	289	153689	687	152128	600	153898	5527	153835	415	138394	601
			2	64358	64390	7.17	61811	86.6	63744	239	63897	600	64343	1039	64329	47.4	49465	600
			4	20593	20675	4.94	18695	16.3	19535	38.7	20290	455	20547	143	20517	7.71	16738	600
			6	6996	7000	4.03	5778	5.30	5657	9.01	6549	271	6900	63.1	6885	9.52	5082	600
60	0.5	0.6	1	100764	100764	5.61	98289	199	99870	640	98923	600	100607	4241	100496	512	89755	600
			2	19564	19742	9.04	17846	42.8	18124	68.2	18338	349	19294	585	19262	32.7	17947	601
			4	833	833	9.99	519	3.72	278	5.42	216	89.1	833	4.85	832	6.40	775	348
			6	111	111	3.83	93	2.07	0	3.05	0	89.7	111	0.72	111	62.2	101	242
60	1.0	0.2	1	63264	63264	10.5	61203	181	62855	424	62031	600	63002	2163	62966	396	58956	600
			2	26253	26269	17.6	24803	49.7	26006	82.4	25802	643	26253	814	26245	16.2	25738	601
			4	10291	10339	9.05	9260	17.2	9763	20.7	9681	171	10247	96.8	1 231	8.58	9898	601
			6	5167	5167	6.1	4379	13.0	4510	14.2	4504	121	5123	36.8	5112	12.4	4826	601
60	1.0	0.6	1	66864	66864	13.1	63547	185	65081	457	64601	600	65526	2743	65374	487	51868	601
			2	16475	16487	20.1	14738	24.8	15505	58.6	15338	286	16347	490	16329	19.2	15926	601
			4	1343	1351	10.2	1026	6.32	744	7.97	759	69.0	1334	17.8	1332	7.86	1299	600
			6	279	279	5.02	186	3.54	6	5.42	3	600	279	0.53	279	29.1	253	356
90	0.5	0.2	1	403833	403833	21.3	398757	2093	403399	4198	384039	600			399476	600	354011	600
			2	170977	171023	42.0	166331	545	170140	1371	164503	600	170850	5711	170812	220	4101	614
			4	56745	56945	20.2	53474	110	55425	250	55211	600	56574	1186	56535	29.0	33097	602
			6	21724	21724	18.2	19471	42.9	20035	70.7	20440	600	21461	370	21423	19.4	12723	605
90	0.5	0.6	1	242737	242737	20.8	238203	1550	241376	3059	226229	600			238710	600	204849	601
			2	57213	57353	43.7	54150	292	55127	446	52677	600	56668	2451	56609	232	26671	601
			4	2792	2870	55.8	2060	17.8	1629	18.7	1558	600	2696	40.4	2687	17.5	2228	603
			6	290	310	27.5	177	9.33	0	10.0	0	254	290	0.54	290	63.6	256	603
90	1.0	0.2	1	144901	144901	54.6	140556	1317	143477	2706	136595	600			142885	600	128330	601
			2	60780	60800	120	58063	215	60039	507	58986	600	60717	2455	60682	45.1	54700	603
			4	21067	21300	45.8	19240	58.3	20079	70.2	19888	600	20974	567	20945	23.2	18016	603
			6	8740	8759	34.2	7358	35.7	7422	36.5	7289	364	8536	200	8509	16.7	5843	601
90	1.0	0.6	1	138667	138667	53.1	133985	1427	136191	2222	128058	600			134890	600	105645	601
			2	27289	27289	121	24812	124	25680	185	25103	600	26970	1743	26943	78.8	25210	600
			4	2013	2054	52.0	1375	20.5	885	23.3	857	421	1997	10.7	1987	6.26	1603	604
			6	150	150	20.6	68	12.7	0	11.3	0	349	150	0.64	150	57.9	77	600

Table 1: Raw results

	Heur	AssSKS	AssBKY	ContAss	LinCG	LagRes	LagOcc
$m = 1$	0.00%	3.59%	1.43%	3.56%	0.73% *	1.16%	12.09%
	15.4s	607s	1210s	417s	2124s *	364s	600s
$m = 2$	0.21%	7.93%	5.17%	5.49%	0.56%	0.65%	19.74%
	29.6s	107s	229s	337s	1200s	54.7s	562s
$m = 4$	1.04%	23.65%	34.52%	37.60%	0.68%	1.01%	12.65%
	17.5s	21.1s	36.9s	257s	176s	9.65s	447s
$m = 6$	0.83%	26.32%	55.3%	55.60%	0.54%	0.66%	18.50%
	11.0s	11.5s	15.0s	243s	62.2 s	25.4s	440s
$n = 30$	0.71%	23.92%	34.55%	37.71%	0.39%	0.68%	6.37%
	0.54s	1.23s	3.21s	18.2s	25.3s	3.38	292s
$n = 60$	0.22%	16.0%	30.53%	29.89%	0.48%	0.59%	10.15%
	7.81s	19.8s	40.0s	270s	246s	18.8s	539s
$n = 90$	1.14%	17.8%	28.23%	29.59%	0.90%	1.05%	33.3%
	50.2s	124s	250s	516s	1228s	67.5s	604s
$\tau = 0.5$	0.92%	17.11%	28.35%	28.8%	0.62%	0.83%	23.89%
	14.6s	69.5s	150s	306s	694s	45.2s	517s
$\tau = 1.0$	0.47%	20.97%	33.52%	35.54%	0.57%	0.72%	10.34%
	25.9s	32.7s	57.8s	274s	367s	18.7s	480s
$\rho = 0.2$	0.28%	10.55%	8.87%	9.48%	0.45%	0.57%	19.64%
	18.5s	66.9s	152s	320s	719s	26.4s	536s
$\rho = 0.6$	1.12%	28.15%	54.44%	56.35%	0.75%	0.99%	14.03%
	22.5s	33.4s	50.2s	258s	322s	37.0s	458s
Mean dev	0.68%	19.1%	31.0%	32.2%	0.59%	0.77%	16.92%
Max dev	6.90%	54.7%	100%	100%	3.44%	3.76%	97.6%
Mean CPU	20.4s	50.6s	102.6s	289s	526s	31.5s	498s

* instances with $n = 90$ are not taken into account

Table 2: Mean deviations and mean CPU times

results for the instances with $n = 90$ and $m = 1$ because the algorithm LinCG failed to find a solution within two hours.

Table 2 shows, for different classes of instances, the absolute mean deviation of the upper and lower bounds from the best known solution and the mean CPU time. In the first part of the table, the instances are classified according to the number of machines m. Clearly, CPU times are significantly larger for the single-machine instances. Moreover, LinCG cannot be run for all the 1 machine instances so, in the rest of the table, we only consider instances with $m \in \{2, 4, 6\}$ when we indicate the mean results for the instances classified according to n, τ and ρ. The last part of the table shows the global mean behaviour of each algorithm for all the instances with $m > 1$.

The main conclusion of these results is that, first, the gap between the best lower bound and the best known solution is less than 1% on average and, second, the mean deviation between the heuristic Heur and the best known solution is also less than 1%. Consequently, the problem can be considered as well-solved in a practical view. Heur finds very good solutions in seconds, running ten times Heur renders near-optimal solutions in minutes. It can be observed that, for all the one-machine

Instance	# jobs	Best LB	Best UB	Instance	# jobs	Best LB	Best UB
NCOS01	8	1025	1025	NCOS13	24	6893	6893
NCOS01a	8	975	975	NCOS13a	24	5257	5257
NCOS02	10	3310	3310	NCOS14	25	8870	8870
NCOS02a	10	1490	1490	NCOS14a	25	4340	4500
NCOS03	10	7490	7490	NCOS15	30	16579	16579
NCOS03a	10	2050	2050	NCOS15a	30	10478	10479
NCOS04	10	2504	2504	NCOS31	75	35995	36485
NCOS04a	10	1733	1733	NCOS31a	75	26168	26240
NCOS05	15	4491	4491	NCOS32	75	35370	35370
NCOS05a	15	3118	3118	NCOS32a	75	25670	25670
NCOS11	20	8077	8077	NCOS41	90	15260	15422
NCOS11a	20	5163	5163	NCOS41a	90	11894	11922
NCOS12	24	10855	10855				
NCOS12a	24	7946	7946				

Table 3: Upper and lower bounds for NCOS-MS-ET instances

instances, HEUR finds the best known solution.

The best lower bound is the linear relaxation of the x_{jt}-formulation. However, the linear relaxation of the resource constraints gives very close results in significantly less time. Very interestingly, these lower bounds outperform the other lower bounds for all the classes of instances. AssBKY is better than LagRes on some instances with $m = 1$ and $n = 90$ but, for these instances, LagRes is stopped by its time limit whilst AssBKY has no time limit. The last lower bound based on the x_{jt}-formulation, namely LagOcc, is not so efficient. For some 30-job instances, with enough CPU time allowed, we managed to make the subgradient method reach the linear relaxation value but, in a general way, the convergence of LagOcc seems more difficult than for LagRes.

The lower bounds based on the y_{jt}-formulation are efficient when $m = 1$. Unfortunately, the efficiency decreases when the number of machines becomes larger. For some instances AssBKY and ContAss are unable to find out a better lower bound than the obvious null value, which explains that the maximal deviation is 100%. This phenomenon was already observed by [3] and [23] to appear when the competition between tasks for resource is quite weak. Finally, AssSKS is faster than AssBKY because some columns with identical assignment costs can be aggregated in the LP.

For each algorithm, computation times unsurprisingly increase with n but they decrease with m. The hardest case for the lower bounds is $m = 1$ (even if we recall that the CPU time could be decreased by using *ad hoc* data structures), while the most difficult case for HEUR is $m = 2$ (when $m = 1$ the list-crossover neighborhood is not used). The parameter associated to the due date generation are not as significant.

We also tested the algorithms HEUR and LagRes on the single-machine earliness-tardiness instances of Masc Lib, a library of manufacturing scheduling problems from

Instance	# jobs	Best LB	Best UB	Instance	# jobs	Best LB	Best UB
NCOS01	8	1010	1010	NCOS11	20	7520	7520
NCOS01a	8	975	975	NCOS11a	20	5163	5163
NCOS02	10	2970	2970	NCOS12	24	10025	10029
NCOS02a	10	1490	1490	NCOS12a	24	7232	7232
NCOS03	10	7140	7140	NCOS13	24	5843	5843
NCOS03a	10	1990	1990	NCOS13a	24	5093	5093
NCOS04	10	2464	2464	NCOS14	25	8540	8540
NCOS04a	10	1733	1733	NCOS14a	25	4340	4390
NCOS05	15	4491	4491	NCOS15	30	13475	13649
NCOS05a	15	3318	3118	NCOS15a	30	10479	10479

Table 4: Upper and lower bounds for NCOS-MS-ET-UNP instances

industry [19]. We also adapted these algorithms in order that they can solve problems where some jobs can be left non-performed if an additional penalty is paid. Table 3 and 4 report the best lower and upper bounds obtained for the tested instances.

6 Conclusion

This paper has addressed the earliness-tardiness scheduling problem on parallel machines with a focus on lower bounds. Several lower bounds recently proposed for the one-machine problem have been extended to the parallel machine case. A simple but efficient local search heuristic has also been provided.

Experimental tests show that the best lower bound is the Lagrangean relaxation of the resource constraints in the time-indexed formulation and the gap between this lower bound and the heuristic is very weak. Clearly, an efficient branch-and-bound procedure could be derived from this lower bound. With this goal in mind, the subgradient method should be improved, heuristics to find good Lagrangean multipliers could be very helpful.

References

[1] R.K. Ahuja, J.B. Orlin, C. Stein, and R.E. Tarjan, *Improved algorithms for bipartite network flows*, SIAM Journal on Computation **23** (1994), 906–933.

[2] J.D. Boissonnat and M. Yvinec, *Algorithmic geometry*, Cambridge University Press, UK, 2001.

[3] K. Bülbül, P. Kaminsky, and C. Yano, *Preemption in single machine earliness/tardiness scheduling*, Working paper, 2004.

[4] M.C. Carlisle and Lloyd E.L., *On the $k-$coloring of intervals*, Discrete Applied Mathematics **59** (1995), 225–235.

[5] H. Chen, C. Chu, and J.M. Proth, *An improvement of the Lagrangean relaxation approach for job shop scheduling: A dynamic programming method*, IEEE Transactions on Robotics and Automation **14** (1998), 786–795.

[6] Z.L. Chen and W.B. Powell, *A column generation based decomposition algorithm for a parallel machine just-in-time scheduling problem*, European Journal of Operational Research **116** (1999), 220–232.

[7] M.E. Dyer and L.A. Wolsey, *Formulating the single machine sequencing problem with release dates as a mixed integer problem*, Discrete Applied Mathematics **26** (1990), 255–270.

[8] C. Fremuth-Paeger, *Goblin: A library for graph matching and network programming problems — Reference manual*, Universität Augsburg, 2004, http://www.math.uni-augsburg.de/opt/goblin.html.

[9] L. Gelders and P. Kleindorfer, *Coordinating aggregate and detailed scheduling decisions in the one-machine job shop. i. theory*, Operations Research **22**, 46–60.

[10] R.E. Graham, E.L. Lawler, J.K. Lenstra, and A.H.G. Rinnooy Kan, *Optimization and approximation in deterministic sequencing and scheduling: a survey*, Annals of Discrete Mathematics **4** (1979), 287–326.

[11] Y. Hendel and F. Sourd, *Calcul de voisinages pour l'ordonnancement avec critères irréguliers*, Proceedings of MOSIM'03, 2003, pp. 21–25.

[12] ______, *Efficient neighborhood search for just-in-time scheduling problems*, Available at www.optimization-online.org, 2004.

[13] J.A. Hoogeveen, *Multicriteria scheduling*, European Journal of Operational Research **to appear** (2005).

[14] J.A. Hoogeveen and S.L. van de Velde, *A branch-and-bound algorithm for single-machine earliness-tardiness scheduling with idle time*, INFORMS Journal on Computing **8** (1996), 402–412.

[15] H.W. Kuhn, *The Hungarian method for the assignment problem*, Naval Research Logistic Quaterly **2** (1955), 83–97.

[16] J.K. Lenstra, A.H.G. Rinnooy Kan, and P. Brucker, *Complexity of machine scheduling problems*, Annals of Discrete Mathematics **1** (1977), 343–362.

[17] P.B. Luh, D.J. Hoitomt, E. Max, and K.R. Pattipati, *Schedule generation and reconfiguration for parallel machines*, IEEE Transactions on Robotics and Automation **6** (1990), 687–696.

[18] R.H. Möhring, A.S. Schulz, F. Stork, and M. Uetz, *Solving project scheduling problems by minimum cut computations*, Management Science **49** (2003), 330–350.

[19] W. Nuijten, T. Bousonville, F. Focacci, D. Godard, and C. Le Pape, *Towards a real-life manufacturing scheduling problem and test bed*, Proceedings of PMS'04, 2004, `http://www2.ilog.com/masclib`, pp. 162–165.

[20] L. Péridy, E. Pinson, and D. Rivreau, *Using short-term memory to minimize the weighted number of late jobs on a single machine*, European Journal of Operational Research **148** (2003), 591–603.

[21] M. Queyranne and A.S. Schulz, *Polyhedral approaches to machine scheduling*, Technical Report 408-1994, Technische Universität Berlin, 1994.

[22] M.W.P. Savelsbergh, R.N. Uma, and Joel Wein, *An experimental study of lp-based approximation algorithms for scheduling problems*, INFORMS Journal on Computing **to appear** (2004).

[23] F. Sourd, *The continuous assignment problem and its application to preemptive and non-preemptive scheduling with irregular cost functions*, INFORMS Journal on Computing **16** (2002), 198–208.

[24] ———, *Optimal timing of a sequence of tasks with general completion costs*, European Journal of Operational Research (2005), to appear.

[25] F. Sourd and S. Kedad-Sidhoum, *The one machine problem with earliness and tardiness penalties*, Journal of Scheduling **6** (2003), 533–549.

[26] J.P. De Sousa and L.A. Wolsey, *A time-indexed formulation of non-preemptive single-machine scheduling problems*, Mathematical Programming **54** (1992), 353–367.

[27] V. T'kindt and J.-C. Billaut, *Multicriteria scheduling: Theory, models and algorithms*, Springer-Verlag, 2002.

[28] J.M. van den Akker, C.A.J. Hurkens, and M.W.P. Savelsbergh, *Time-indexed formulations for machine scheduling problems: column generation*, INFORMS Journal on Computing **12** (2000), 111–124.

[29] J.A. Ventura and D. Kim, *Parallel machine scheduling with earliness-tardiness penalties and additional resource constraints*, Computers & Operations Research **30** (2003), 1945–1958.