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 through 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 \( J = \{1, \cdots, n\} \) 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 \([10]\).

Chen and Powell \([6]\) study the special case where the jobs have an unrestrictedly 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 \( P|r_j|\sum_j \alpha_j E_j + \beta_j T_j \) 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 \( T^* = \max_j d_j + \max_j p_j + \lceil \frac{\sum_j p_j}{m} \rceil \) (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 \( T^* \) on a machine, it can be processed earlier by another machine. So we will consider that \( T = T^* \). 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 \( \text{est}_{jt}(t) = \max(r_j, t-p_j+1) \) 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

\[
\begin{align*}
\min & \quad \sum_{j \in J} \sum_{t=r_j}^{T-p_j} c_{jt} x_{jt} \\
\text{s.t.} & \quad \sum_{t=r_j}^{T-p_j} x_{jt} = 1 \quad \forall j \in J \quad (1) \\
& \quad \sum_{j \in J} \sum_{s=\text{est}_{jt}(t)} x_{js} \leq m \quad \forall t \in [0, T-1] \quad (2) \\
& \quad x_{jt} \in \{0, 1\} \quad \forall j \in J, \forall t \in [r_j, T-p_j] \quad (3)
\end{align*}
\]

Equations (2) ensure that each job is proceeded once. Inequalities (3), also referred to as resource constraints, state that at most \( m \) jobs can be handled at any time. Clearly, this formulation allows the occurrence 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 \cite{28} provides a strong lower bound which dominates the bounds provided by other mixed integer programming formulations \cite{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 $(S_1, \ldots, S_n)$ is first computed with the heuristic described in Section \cite{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 reduced 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 occurrences

Another way to cope with the difficulty of the $x_{jt}$-formulation is to consider the Lagrangean relaxation of the equalities \cite{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. \cite{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 $\mu_j$ for each constraint \cite{2}. For each vector $\mu = (\mu_1, \ldots, \mu_n)$, a lower bound denoted by $LR_1(\mu)$ is obtained by solving the
Lagrangean problem

\[
\min_{x_{jt}} \sum_{j \in J} \sum_{t=r_j}^{T-p_j} (c_{jt} - \mu_j) x_{jt} + \sum_{j \in J} \mu_j
\]
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 \( t + p_j \) 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 \( LR_1(\mu) \).

This is a generalization of the work of Périddy 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 Périddy 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 \( LR_1(\mu) \) 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, \( \max_{\mu} LR_1(\mu) \) 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 \( \mu_i \geq 0 \) 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 \( \mu = (\mu_0, \ldots, \mu_{T-1}) \geq 0 \), a lower bound denoted by \( \text{LR}_2(\mu) \) is obtained by solving the Lagrangean problem

\[
\min_{x_{jt}} \sum_{j \in J} \sum_{t=r_j}^{T-p_j} c_{jt} x_{jt} + \mu_t \left( \sum_{s=\text{est}_j(t)}^t x_{js} - m \right)
\]

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 \( \sum_t \left( c_{jt} + \sum_{s=\text{est}_j(t)}^t \mu_s \right) x_{jt} \) 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, \( \text{LR}_2(\mu) \) is maximized through the subgradient method and again the integrality property of the Lagrangean problem shows that \( \max_\mu \text{LR}_2(\mu) \) 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-Sikhoum [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 \( y_{jt-1} = 0 \)).

\[
\min \sum_{j \in J} \sum_{t=r_j}^{T-p_j} c_{jt} y_{jt}
\]

\[
\text{s.t.} \quad \sum_{t=r_j}^{T-p_j} y_{jt} = p_j \quad \forall j \in J
\]

\[
\sum_{j \in J} y_{jt} \leq m \quad \forall t \in 0, \ldots, T - 1
\]

\[
z_{jr_j} \geq y_{jr_j} \quad \forall j \in J
\]

\[
z_{jt} \geq y_{jt} - y_{jt-1} \quad \forall j \in J, \forall t \in \{r_j + 1, \ldots, T - p_j\}
\]

\[
\sum_{t=r_j}^{T-p_j} z_{jt} = 1 \quad \forall j \in J
\]

\[
y_{jt}, z_{jt} \in \{0, 1\} \quad \forall j \in J, \forall t \in \{r_j, \ldots, T - p_j\}
\]
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
\[
\sum_{s=t-p_j}^{t-1} c'_{js} = \max(\alpha_j(d_j - t), \beta_j(t - T_j)) \quad \forall t \forall j \in J
\]
and a simple solution proposed by Sourd and Kedad-Sidhoum [25] is
\[
c'_{jt} = \begin{cases} 
\alpha_j \left\lfloor \frac{d_j - t - 1}{p_j} \right\rfloor & \text{if } t < d_j, \\
\beta_j \left\lceil \frac{t + 1 - d_j}{p_j} \right\rceil & \text{if } t \geq d_j.
\end{cases}
\]

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 $P|\text{pmtn}, r_j| \sum \alpha_j E_j + \beta_j T_j$. Indeed, the latter problem can be shown to be equivalent to $P|\text{pmtn}, r_j| \sum w_j T_j$, which means that the earliness costs are relaxed when “usual” preemption is allowed.

Clearly, the relaxed problem 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 [13]. 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
is null in general. Let us here consider that $T$ is no more equal to $T^*$ but satisfy $\max_j d_j < T < T^*$ 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 $\mathcal{N}(n, T^*)$. This suggests to first run the flow algorithm for some $T \in (\max_j d_j, T^*)$ 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 $T = \min(T^*, 3/2 \max(\max_j d_j, \max_j r_j + p_j, \sum_j p_j))$. 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 $\sum_{t=1}^{t-1} c'_{jt} \leq \max (\alpha_j (d_j - t), \beta_j (t - T_j))$:

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

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

\[
f_j(t) = \begin{cases} 
-\frac{\alpha_j}{2} + \frac{\alpha_j}{p_j} (d_j - t) & \text{if } t \leq d_j, \\
\frac{\beta_j}{2} + \frac{\beta_j}{p_j} (t - d_j) & \text{if } t > d_j.
\end{cases}
\]
It satisfies the inequality $\int_{t-p_1}^t f_i(s)ds \leq \max (\alpha_j(d_j-t), \beta_j(t-T_j))$ 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

$$\min \sum_{j \in J} \int_0^\infty f_j(t)\delta_j(t)dt$$

subject to

$$\int_0^\infty \delta_j(t)dt = p_j \quad \forall j \in J$$

(19)

$$\sum_{j \in J} \delta_j(t) \leq m \quad \forall t \geq 0$$

(20)

$$\delta_j(t) \in \{0, 1\} \quad \forall t \geq 0 \quad \forall j \in J$$

(21)

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

$$\min \sum_{j \in J} \mu_j p_j + \sum_{j \in J} \int_0^\infty (f_j(t) - \mu_j)\delta_j(t)dt$$

subject to

$$\sum_{j \in J} \delta_j(t) \leq m \quad \forall t \geq 0$$

In other words, computing $g_\mu(t)$ consists in choosing at most $m$ values in the multiset of real values \{\(f_1(t) - \mu_1, f_2(t) - \mu_2, \ldots, f_n(t) - \mu_n\)} 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 $\int g_\mu(t)dt$ 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 $f_i - \mu_i$ ($i = 0, \ldots, n$) 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

Figure 2: Continuous lower bound for $m = 2$ machines
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 \( \int g_\mu(t)dt \) can be computed in \( O(n^2m) \).

The function \( \mu \mapsto \int g_\mu(t)dt \) 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 Bülbil 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 \( \nu_i \) elements of \( L_i \) followed by the last \( n_j - \nu_j \) elements of \( L_j \) and a list formed by the first \( \nu_j \) 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 occurrences 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 \( T = T^* \)) 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 \textsc{LagOcc}, \textsc{LagRes} and \textsc{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, \textsc{LinCG}, \textsc{AssSKS}, \textsc{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. \cite{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. \cite{3} show that computation times are indeed dramatically decreased for their assignment based lower bound. We also recall that Sourd \cite{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 \( \tau \) and the range factor \( \rho \). Given \( \tau, \rho \) and \( J \), the due date of each job \( j \) is generated from the uniform distribution \( \{\theta_j, \cdots, \theta_j + \rho \sum_{k \in J} p_k\} \) where \( \theta_j = \max(\tau_{\max} + p_j, (\tau - \rho/2) \sum_{k \in J} p_k) \).

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 \( \tau \) 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, \textsc{AssSKS} and \textsc{AssBKY} can use the \( O(n^2 T) \) algorithm of \cite{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 \ref{table:results}. 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 \textsc{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 \textsc{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
<table>
<thead>
<tr>
<th>n</th>
<th>τ</th>
<th>ρ</th>
<th>m</th>
<th>Best</th>
<th>Heur</th>
<th>ASSK</th>
<th>ASSBKY</th>
<th>ContAss</th>
<th>LinCG</th>
<th>LagRes</th>
<th>LagOcc</th>
<th>LAGCOR</th>
<th>LAGCC</th>
</tr>
</thead>
<tbody>
<tr>
<td>30</td>
<td>0.5</td>
<td>0.2</td>
<td>1</td>
<td>47286</td>
<td>47286</td>
<td>47310</td>
<td>47286</td>
<td>47125</td>
<td>47135</td>
<td>47286</td>
<td>38.2</td>
<td>47286</td>
<td>87.2</td>
</tr>
<tr>
<td>30</td>
<td>0.5</td>
<td>0.2</td>
<td>1</td>
<td>19104</td>
<td>19200</td>
<td>19786</td>
<td>19722</td>
<td>19073</td>
<td>19094</td>
<td>19094</td>
<td>5.54</td>
<td>19094</td>
<td>5.54</td>
</tr>
<tr>
<td>4</td>
<td>580.5</td>
<td>5865.6</td>
<td>47</td>
<td>525</td>
<td>525</td>
<td>525</td>
<td>525</td>
<td>525</td>
<td>525</td>
<td>525</td>
<td>525</td>
<td>525</td>
<td>525</td>
</tr>
<tr>
<td>4</td>
<td>2066</td>
<td>2130.5</td>
<td>7.13</td>
<td>1431</td>
<td>1431</td>
<td>1431</td>
<td>1431</td>
<td>1431</td>
<td>1431</td>
<td>1431</td>
<td>1431</td>
<td>1431</td>
<td>1431</td>
</tr>
<tr>
<td>6</td>
<td>2097</td>
<td>2097.0</td>
<td>28.6</td>
<td>2097</td>
<td>2097</td>
<td>2097</td>
<td>2097</td>
<td>2097</td>
<td>2097</td>
<td>2097</td>
<td>2097</td>
<td>2097</td>
<td>2097</td>
</tr>
<tr>
<td>6</td>
<td>236</td>
<td>236</td>
<td>236</td>
<td>236</td>
<td>236</td>
<td>236</td>
<td>236</td>
<td>236</td>
<td>236</td>
<td>236</td>
<td>236</td>
<td>236</td>
<td>236</td>
</tr>
<tr>
<td>6</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>60</td>
<td>0.5</td>
<td>0.2</td>
<td>1</td>
<td>13780</td>
<td>13808</td>
<td>13685</td>
<td>13685</td>
<td>13685</td>
<td>13685</td>
<td>13685</td>
<td>13685</td>
<td>13685</td>
<td>13685</td>
</tr>
<tr>
<td>60</td>
<td>0.5</td>
<td>0.2</td>
<td>1</td>
<td>5865</td>
<td>5865</td>
<td>5865</td>
<td>5865</td>
<td>5865</td>
<td>5865</td>
<td>5865</td>
<td>5865</td>
<td>5865</td>
<td>5865</td>
</tr>
<tr>
<td>60</td>
<td>6</td>
<td>2066</td>
<td>2066</td>
<td>2066</td>
<td>2066</td>
<td>2066</td>
<td>2066</td>
<td>2066</td>
<td>2066</td>
<td>2066</td>
<td>2066</td>
<td>2066</td>
<td>2066</td>
</tr>
<tr>
<td>60</td>
<td>6</td>
<td>236</td>
<td>236</td>
<td>236</td>
<td>236</td>
<td>236</td>
<td>236</td>
<td>236</td>
<td>236</td>
<td>236</td>
<td>236</td>
<td>236</td>
<td>236</td>
</tr>
<tr>
<td>60</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

**Table 1:** Raw results
<table>
<thead>
<tr>
<th>$m$ = 1</th>
<th>$m$ = 2</th>
<th>$m$ = 4</th>
<th>$m$ = 6</th>
</tr>
</thead>
<tbody>
<tr>
<td>$n$ = 30</td>
<td>0.71%</td>
<td>23.92%</td>
<td>34.55%</td>
</tr>
<tr>
<td>$n$ = 60</td>
<td>0.02%</td>
<td>16.00%</td>
<td>30.53%</td>
</tr>
<tr>
<td>$n$ = 90</td>
<td>0.14%</td>
<td>17.85%</td>
<td>28.23%</td>
</tr>
<tr>
<td>(\tau = 0.5)</td>
<td>0.92%</td>
<td>17.11%</td>
<td>28.35%</td>
</tr>
<tr>
<td>(\tau = 1.0)</td>
<td>0.47%</td>
<td>20.97%</td>
<td>33.52%</td>
</tr>
<tr>
<td>(\rho = 0.2)</td>
<td>0.28%</td>
<td>10.55%</td>
<td>8.87%</td>
</tr>
<tr>
<td>(\rho = 0.6)</td>
<td>1.12%</td>
<td>28.15%</td>
<td>54.44%</td>
</tr>
<tr>
<td>Mean dev</td>
<td>0.68%</td>
<td>19.11%</td>
<td>31.0%</td>
</tr>
<tr>
<td>Max dev</td>
<td>6.90%</td>
<td>54.7%</td>
<td>100%</td>
</tr>
<tr>
<td>Mean CPU</td>
<td>20.4s</td>
<td>50.0s</td>
<td>102.6s</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>$m$ = 1</th>
<th>$m$ = 2</th>
<th>$m$ = 4</th>
<th>$m$ = 6</th>
</tr>
</thead>
<tbody>
<tr>
<td>$n$ = 30</td>
<td>0.71%</td>
<td>23.92%</td>
<td>34.55%</td>
</tr>
<tr>
<td>$n$ = 60</td>
<td>0.02%</td>
<td>16.00%</td>
<td>30.53%</td>
</tr>
<tr>
<td>$n$ = 90</td>
<td>0.14%</td>
<td>17.85%</td>
<td>28.23%</td>
</tr>
<tr>
<td>(\tau = 0.5)</td>
<td>0.92%</td>
<td>17.11%</td>
<td>28.35%</td>
</tr>
<tr>
<td>(\tau = 1.0)</td>
<td>0.47%</td>
<td>20.97%</td>
<td>33.52%</td>
</tr>
<tr>
<td>(\rho = 0.2)</td>
<td>0.28%</td>
<td>10.55%</td>
<td>8.87%</td>
</tr>
<tr>
<td>(\rho = 0.6)</td>
<td>1.12%</td>
<td>28.15%</td>
<td>54.44%</td>
</tr>
<tr>
<td>Mean dev</td>
<td>0.68%</td>
<td>19.11%</td>
<td>31.0%</td>
</tr>
<tr>
<td>Max dev</td>
<td>6.90%</td>
<td>54.7%</td>
<td>100%</td>
</tr>
<tr>
<td>Mean CPU</td>
<td>20.4s</td>
<td>50.0s</td>
<td>102.6s</td>
</tr>
</tbody>
</table>

* 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 $n$, $\tau$ and $\rho$. 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
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
industry\cite{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.

<table>
<thead>
<tr>
<th>Instance</th>
<th># jobs</th>
<th>Best LB</th>
<th>Best UB</th>
</tr>
</thead>
<tbody>
<tr>
<td>NCOS01</td>
<td>8</td>
<td>1010</td>
<td>1010</td>
</tr>
<tr>
<td>NCOS01a</td>
<td>8</td>
<td>975</td>
<td>975</td>
</tr>
<tr>
<td>NCOS02</td>
<td>10</td>
<td>2970</td>
<td>2970</td>
</tr>
<tr>
<td>NCOS02a</td>
<td>10</td>
<td>1490</td>
<td>1490</td>
</tr>
<tr>
<td>NCOS03</td>
<td>10</td>
<td>1990</td>
<td>1990</td>
</tr>
<tr>
<td>NCOS03a</td>
<td>10</td>
<td>2464</td>
<td>2464</td>
</tr>
<tr>
<td>NCOS04</td>
<td>10</td>
<td>3318</td>
<td>3118</td>
</tr>
<tr>
<td>NCOS04a</td>
<td>10</td>
<td>4491</td>
<td>4491</td>
</tr>
<tr>
<td>NCOS05</td>
<td>15</td>
<td>1010</td>
<td>1010</td>
</tr>
<tr>
<td>NCOS05a</td>
<td>15</td>
<td>7520</td>
<td>7520</td>
</tr>
<tr>
<td>NCOS11</td>
<td>20</td>
<td>5163</td>
<td>5163</td>
</tr>
<tr>
<td>NCOS11a</td>
<td>20</td>
<td>1490</td>
<td>1490</td>
</tr>
<tr>
<td>NCOS12</td>
<td>24</td>
<td>5843</td>
<td>5843</td>
</tr>
<tr>
<td>NCOS12a</td>
<td>24</td>
<td>5093</td>
<td>5093</td>
</tr>
<tr>
<td>NCOS13</td>
<td>24</td>
<td>4390</td>
<td>4390</td>
</tr>
<tr>
<td>NCOS13a</td>
<td>24</td>
<td>13649</td>
<td>13649</td>
</tr>
<tr>
<td>NCOS14</td>
<td>25</td>
<td>8540</td>
<td>8540</td>
</tr>
<tr>
<td>NCOS14a</td>
<td>25</td>
<td>5163</td>
<td>5163</td>
</tr>
<tr>
<td>NCOS15</td>
<td>30</td>
<td>10479</td>
<td>10479</td>
</tr>
<tr>
<td>NCOS15a</td>
<td>30</td>
<td>13475</td>
<td>13475</td>
</tr>
</tbody>
</table>

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


17


