Using Lagrangean Relaxation to Minimize the (Weighted) Number of Late Jobs on a Single Machine

Stéphane Dauzère-Pérès[a*] Marc Sevaux[b]

[a] Department of Finance and Management Science
Norwegian School of Economics and Business Administration
Helleveien 30
N-5035 Bergen-Sandviken, Norway

[b] Department of Automatic Control and Production Engineering
Ecole des Mines de Nantes
La Chantrerie, BP 20722
F-44307 Nantes Cedex 03, France

E-mail: {Stephane.Dauzere-Peres, Marc.Sevaux}@emn.fr

June 28, 1999

Abstract

This paper tackles the general single machine scheduling problem, where jobs have different release and due dates and the objective is to minimize the weighted number of late jobs. The notion of *master sequence* is first introduced, *i.e.*, a sequence that contains at least an optimal sequence of jobs on time. This *master sequence* is used to derive an original mixed-integer linear programming formulation. By relaxing some constraints, it is possible to design a Lagrangean relaxation algorithm which gives both lower and upper bounds. The special case where jobs have equal weights is analyzed. Computational results are presented and, although the duality gap becomes larger with the number of jobs, it is possible to solve problems of more than 100 jobs.

1 Introduction

A set of n jobs $\{J_1, .., J_n\}$, subject to *release dates* r_i and *due dates* d_i, have to be scheduled on a single machine. The *processing time* of jobs on the machine is denoted by p_i, and

*on leave from IRCyN/Ecole des Mines de Nantes a weight w_i is associated to each job. The machine can only process one job at a time. A scheduled job completed before its due date is said to be *early* or *on time*, and *late* otherwise. The objective is to minimize the weighted number of late jobs, or equivalently to maximize the weighted number of early jobs. A well-known and important remark is that there is always an optimal schedule in which late jobs are sequenced after all the early jobs.

This single-machine scheduling problem, noted $1|r_j|\sum w_jU_j$ in the standard classification, is strongly $\mathcal{NP}$-Hard [8]. When all weights are equal ($1|r_j|\sum U_j$), the problem remains $\mathcal{NP}$-Hard, but becomes polynomially solvable if all release dates are equal ($1||\sum U_j$) [9] ($O(n \log n)$), or if release and due dates are similarly ordered ($r_i < r_j \Rightarrow d_i \leq d_j$ $\forall (J_i, J_j)$) [6] ($O(n^2)$), [7] ($O(n \log n)$). However, some exact approaches have recently been proposed for this problem [1] [5]. Lawler [7] showed that the Moore's algorithm ([9]) could be applied when processing times and weights are aggeeable, *i.e.*, $p_i < p_j \Rightarrow w_i \geq w_j$ $\forall (J_i, J_j)$. Finally, branch-and-bound procedures have been developed to solve the case where all release dates are equal ($1||\sum w_jU_j$) in [12] and [11]. To our knowledge, no algorithm has been proposed to solve the general problem $1|r_j|\sum w_jU_j$.

In this paper, based on the notion of *master sequence i.e.*, a sequence from which an optimal sequence can be extracted, a new mixed-integer linear programming formulation is introduced. Using this formulation, a Lagrangean relaxation algorithm is derived. Lagrangean relaxation is a powerful optimization tool from which heuristic iterative algorithms can be designed, where both upper and lower bounds are determined at every iteration. It is thus possible to always know the maximum gap between the best solution found and the optimal solution, and stop the algorithm when this gap is small enough. One condition that is often associated to the efficiency of Lagrangean relaxation approaches is to relax as few constraints as possible, in order to obtain good bounds when solving the relaxed problem. This is why our formulation compares very favorably to other known ones (see [4] for a study of classical formulations for this problem). Only one constraint type, coupling variables of different jobs, needs to be relaxed to obtain an easily solvable problem, that can be solved independently for each job.

The master sequence is introduced in Section 2, and the resulting mixed-integer linear programming formulation is given and discussed in Section 3. Section 4 shows how the size of the master sequence, and thus the size of the model, can be reduced. Section 5 presents the Lagrangean relaxation algorithm, and Section 6 improves the algorithm. The non-weighted case is studied in more details in Section 7. Numerical results on a large set of test instances are given and discussed in Section 8. Finally, some conclusions and perspectives are drawn in Section 9.

2 The master sequence

In the remainder of this paper, because we are only interested in sequencing jobs on time (late jobs can be set after the jobs on time), the sequence of jobs will mean the sequence of *early* jobs. Many results in this paper are based on the following theorem.

Theorem 1 *There is always an optimal sequence of jobs on time that solves the problem* $1|r_j|\sum w_j U_j$*, in which every job* J_j *is sequenced just after a job* J_i *such that either condition (1)* $d_i < d_j$*, or (2)* $d_i \geq d_j$ *and* $r_k \leq r_j\ \forall\ J_k$ *sequenced before* J_j*, holds, or equivalently condition (3)* $d_i \geq d_j$ *and* $\exists\ J_k$ *sequenced before* J_j *such that* $r_k > r_j$ *is not satisfied.*

Proof: The proof goes by showing that, by construction, it is possible to change any optimal sequence into an optimal sequence that satisfies the conditions (1) or (2).
Suppose that we have a sequence in which some (or all) ready jobs do not satisfy one of the conditions. Starting from the beginning of the sequence, find the first pair of jobs (J_i, J_j) in the sequence that does not satisfy the two conditions, *i.e.*, for which condition (3) holds. If t_i and t_j denote the start times of the two jobs, the latter condition ensures that, after interchanging the two jobs, J_j can start at t_i (since $\exists\ J_k$ sequenced before J_j such that $r_j < r_k \leq t_i$). Hence, J_i will end at the same time than J_j before the interchange $(t_i + p_i + p_j)$, and thus will still be on time (since $t_i + p_i + p_j \leq d_j \leq d_i$).
The interchange should be repeated if J_j and the new job just before it do not satisfy conditions (1) or (2), until one of these conditions is satisfied for J_j and the job just before it, or J_j is sequenced first.
The procedure is repeated for all jobs until the conditions are satisfied for all jobs. Because once a job has been moved, it will never go back again, one knows that the procedure will not be repeated more than n times, *i.e.*, takes a finite amount of time. □

We will denote by $\mathcal{S}$ the subset of sequences in which jobs satisfy the conditions in Theorem 1. In the sequel, we will only be interested in sequences in $\mathcal{S}$, since we know that it always contains an optimal sequence.

Proposition 1 *If, in a sequence of* $\mathcal{S}$*, job* J_j *is after jobs* J_i *such that* $r_j < r_i$*, then there is at least a job* J_i *such that* $d_i < d_j$*.*

Proof: By contradiction, if all jobs J_i before J_j such that $r_j < r_i$ verify $d_i \geq d_j$, then none of the conditions (1) and (2) is satisfied. Thus, the sequence is not in $\mathcal{S}$. □

Corollary 1 *If, for every job* J_i *such that* $r_j < r_i$*, condition* $d_j \leq d_i$ *holds, then, in every sequence of* $\mathcal{S}$ *(i.e., in an optimal sequence), job* J_j *is sequenced before all jobs* J_i*.*

Corollary 2 *If, for every job J_j such that $d_j < d_i$, condition $r_j \leq r_i$ holds, then, in every sequence of $\mathcal{S}$ (i.e., in an optimal sequence), job J_i is sequenced after all jobs J_j.*

We want to show that it is possible to derive what will be called a *master sequence* and denoted by σ, and which "contains" every sequence in $\mathcal{S}$. Corollary 1 implies that there is only one position for J_j in the master sequence, and Corollary 2 that there is only one position for J_i.

Example 1 *Let us consider a 5-job problem with the data of Table 1.*

Jobs	J_1	J_2	J_3	J_4	J_5
r_i	0	5	8	12	14
p_i	8	6	5	6	10
d_i	16	26	24	22	32

Table 1: Data for a 5-job problem

Considering sequences in $\mathcal{S}$, and because of Corollary 1, one knows that J_1 is set before all jobs (conditions $r_1 < r_i$ and $d_1 < d_i$ are satisfied for every job $J_i \neq J_1$), and all jobs are set before J_5 (conditions $r_i < r_5$ and $d_i < d_5$ are satisfied for every job $J_i \neq J_5$). Hence, in the master sequence σ, job J_1 will be set first and job J_5 last.

The master sequence has the following form:

$$\sigma = (J_1, J_2, J_3, J_2, J_4, J_3, J_2, J_5)$$

Every sequence of jobs in $\mathcal{S}$ can be constructed from σ. In this example, they are numerous sequences or early jobs (more than 40). For instance, the subset of sequences containing 5 jobs is:

$$\{(J_1, J_2, J_3, J_4, J_5), (J_1, J_2, J_4, J_3, J_5), (J_1, J_3, J_2, J_4, J_5), (J_1, J_3, J_4, J_2, J_5), (J_1, J_4, J_3, J_2, J_5)\}$$

One can check that each of these sequences is included in $\mathcal{S}$.

Proposition 2 *In the master sequence, if $r_i < r_j$ and $d_i > d_j$, then there is a position for J_i before J_j and a position for J_i after J_j.*

Proof: Because $r_i < r_j$, Condition (2) in Theorem 1 is satisfied for the pair of jobs (J_i, J_j), and because $d_i > d_j$, Condition (1) is satisfied for the pair (J_j, J_i). Hence, there is a position in the master sequence for J_i before and after J_j. $\square$

Hence, there must be a position in the master sequence for J_i after every job J_j such that $r_i < r_j$ and $d_i > d_j$. This shows that there will be **at most** $\frac{n(n+1)}{2}$ positions in the master sequence.

Corollary 3 *If, for every job J_j such that $r_i < r_j$, the condition $d_i \leq d_j$ holds, then there is only one position for job J_i in the master sequence.*

Corollary 3 shows that, when release and due dates are similarly ordered (as in Kise *et al.* [6]), the master sequence will be the sequence of jobs in increasing order of their release dates (or due dates if some jobs have equal release dates). In the non-weighted case ($w_i = 1$, $\forall J_i$), the problem is then polynomially solvable using the algorithm proposed in [6] (in $O(n^2)$) or in [7] (in $O(n \log n)$).

An interesting and important property of the master sequence is a kind of transitivity property. If job J_i is set before and after J_j in the master sequence because either Condition (1) or (2) of Theorem 1 holds, and if J_j is set before and after J_k in the master sequence because either Condition (1) or (2) holds, then either Condition (1) or (2) of Theorem 1 holds and J_i is set before and after J_k in the master sequence.

The algorithm to create the master sequence σ is sketched below. We suppose that the jobs are pre-ordered in non-decreasing order of their release dates, and $\overline{J}$ denotes the set of jobs already sequenced. Moreover, to speed up the algorithm, jobs added in $\overline{J}$ are ordered on non-decreasing order of their due dates.

`FOR every job` $J_i \in J$ `DO`

  $\sigma \leftarrow \sigma \cup J_i$

  $\overline{J} \leftarrow \overline{J} \cup J_i$

  `FOR every job` $J_j \in \overline{J}$ `such that` $d_j \geq d_i$ `DO`

    $\sigma \leftarrow \sigma \cup J_j$

The algorithm has a time complexity of $O(n^2)$. The job set at position k in σ is denoted $\sigma(k)$. The number of positions in the master sequence is denoted by P. Recall that $P \leq \frac{n(n+1)}{2}$. Actually, P will only be equal to its upper bound if the job with the smallest release date has also the largest due date, the job with the second smallest release date has the second largest due date, and so on (see Proposition 2). This is clearly a very special case and, in practical experiments, P will be much smaller than $\frac{n(n+1)}{2}$.

3 A new mixed-integer linear programming formulation

Based on the master sequence, one can derive the following model:

$$
\begin{cases}
c^* = \min \ c = \sum_{i=1}^{n} w_i U_i & & (1) \\
t_k - t_{k-1} - p_{\sigma(k-1)} u_{k-1} \geq 0 & k = 2, .., P & (2) \\
t_k - r_{\sigma(k)} u_k \geq 0 & \forall k & (3) \\
t_k + p_{\sigma(k)} u_k - d_{\sigma(k)} - D_k(1 - u_k) \leq 0 & \forall k & (4) \\
\sum_{\substack{k=1 \\ \sigma(k)=i}}^{P} u_k + U_i = 1 & \forall i & (5) \\
u_k \in \{0, 1\} & \forall k & (6) \\
U_i \in \{0, 1\} & \forall i & (7)
\end{cases}
$$

where D_k is chosen big enough to not constrain the jobs sequenced before k, for instance

$$
D_k = \max_{\substack{r=1,..,k-1 \\ d_{\sigma(r)} > d_{\sigma(k)}}} (d_{\sigma(r)} - d_{\sigma(k)}) \ \ (= \max_{r=1,..,k-1} (0, d_{\sigma(r)} - d_{\sigma(k)})).
$$

By Constraint (2) we ensure that, if the job at the k^{th} position in the master sequence is set on time ($u_k = 1$), then the job at position $k + 1$ cannot start before the completion of the job at position k. If $u_k = 0$, the constraint only ensures that $t_{k+1} \geq t_k$. Constraint (3) specifies that, if the job is scheduled on time, it cannot start before its release date. By Constraint (4), if the job at position k is set on time ($u_k = 1$), then it has to be completed before its due date. If $u_k = 0$, the constraint is redundant. Finally, Constraint (5) ensures that at most one position is used for each job, or the job is late ($U_i = 1$).

In the previous model, it is possible to replace Constraint (3) by $t_k - r_{\sigma(k)} \geq 0$ (or equivalently to remove u_k from Constraint (3)). The new constraint is numbered (3'). Theorem 2 will proove the validity of the resulting model.

In the non-weighted case ($w_j = 1, \forall J_j$), if Constraint (4) is replaced by $t_k + p_{\sigma(k)} u_k - d_{\sigma(k)} \leq 0$ (or equivalently $D_k = 0$ in Constraint (4)), then the resulting formulation still provides an optimal solution to the problem. The new constraint is numbered (4'). Although the non-weighted case will be studied in more details in Section 7, the following theorem is introduced here because its also useful for the weighted case.

Theorem 2 *In the non-weighted case, there is always an optimal sequence of $\mathcal{S}$ that satisfies Constraints (2), (3'), (4'), and (5)-(7).*

Proof: The proof goes by showing that the only case where there is a problem is when J_j can be sequenced before and after J_i in the master sequence, and $r_j < r_i$ and $d_j > d_i$, and J_i is not sequenced in the optimal sequence. It can be shown that Constraints (2), (3), and (4) prevent job J_j to start between $d_i - p_j$ (Constraint (4)) and r_i (Constraint

(3)). This is only a problem if $d_i - p_j < r_i$. If this is the case, then $p_i < p_j$ (since J_i is not late if started at its release date r_i). Hence, in an optimal solution where J_j starts in the interval $[d_i - p_j, r_i]$, *i.e.*, ends in the interval $[d_i, r_i + p_j]$, J_j can be replaced by J_i, and the sequence will remain optimal since J_i starts after r_i and ends before d_i. □

The proof of Theorem 2 is based on equal weight for jobs. In the weighted case, following the proof of Theorem 2, D_k can be chosen as follows :

$$D_k = \max_{\substack{r=1,..,k-1; \\ d_{\sigma(r)} > d_{\sigma(k)}}} (0, r_{\sigma(r)} - d_{\sigma(k)})$$

Hence, the case where $d_i - p_j < r_i$, discussed in the proof of Theorem 2, is avoided. In numerical experiments, D_k is very often equal to zero.

4 Reducing the master sequence

Because the size of the model is directly linked to the length of the master sequence, it is interesting to remove as many positions as possible from σ. Not only solution procedures will be more efficient, but the model will be tighter and will give better lower bounds by Lagrangean relaxation.

Because of Constraints (2) and (3), $t_k \geq \max_{r=1,..,k-1} r_{\sigma(r)}$. Hence, the first reduction will be done by removing positions k such that $\max_{r=1,..,k-1} r_{\sigma(r)} + p_{\sigma(k)} > d_{\sigma(k)}$.

Several dominance rules are proposed in [5] for the non-weighted case. However, if parameter D_k is changed according to Theorem 2, all of them do not apply. This is because, in the resulting formulation, when job J_j is before and after J_i in the master sequence and J_i is late, the position of J_j after J_i might need to be occupied in an optimal solution. One could show that this is not the case with the initial formulation. Our preliminary numerical experiments showed that reducing parameter D_k was more important than using the lost dominance rules.

We will describe here the dominance rules that still apply to our formulation, and which have been modified for the weighted case (see [5] for details).

In the master sequence, if Conditions (1) $r_i < r_j$, (2) $r_i + p_i \geq r_j + p_j$, (3) $r_i + p_i + p_j > d_j$, (4) $r_j + p_j + p_i > d_i$, (5) $d_i - p_i \leq d_j - p_j$, and (5) $w_j \leq w_i$ hold, then J_j dominates J_i and all positions of job J_i can be removed from the master sequence. Because of Conditions (3) and (4), only one of the two jobs can be scheduled on time. In an optimal solution, either both jobs are late, or it is always possible to find a solution in which job J_j is on time and the total weight of late jobs is as small than a solution with job J_i on time.

Another dominance rule is based on the fact that, if there is a position l and a job J_j ($J_j \neq \sigma(l)$) such that Conditions (1) $r_{\sigma(l)} + p_{\sigma(l)} \geq r_j + p_j$, (2) $p_{\sigma(l)} \geq p_j$, (3) $r_{\sigma(l)} +$

$p_{\sigma(l)} + p_j > d_j$, (4) $r_j + p_j + p_{\sigma(l)} > d_{\sigma(l)}$, (5) $d_{\sigma(l)} - p_{\sigma(l)} \leq d_j - p_j$, and (6) $w_{\sigma(l)} \geq w_j$ are satisfied, then J_j dominates position l, and thus the latter can be removed. This is because, if there is an optimal solution in which position l is occupied (*i.e.*, job $J_{\sigma(l)}$ is on time), then, by Condition (3), J_j is late. The solution can be changed to another optimal solution in which $J_{\sigma(l)}$ is replaced by J_j.

5 A Lagrangean relaxation algorithm

Following Theorem 2 and remarks from Section 3, the mixed-integer linear programming formulation is now:

$$\begin{cases} c^* = \min \ c = \sum_{i=1}^{n} w_i U_i & & (8) \\ t_k - t_{k-1} - p_{\sigma(k-1)} u_{k-1} \geq 0 & k = 2, .., P & (9) \\ t_k - r_{\sigma(k)} \geq 0 & \forall k & (10) \\ t_k + p_{\sigma(k)} u_k - d_{\sigma(k)} - D_k(1 - u_k) \leq 0 & \forall k & (11) \\ \sum_{\substack{k=1 \\ \sigma(k)=i}}^{P} u_k + U_i = 1 & \forall i & (12) \\ u_k \in \{0,1\} & \forall k & (13) \\ U_i \in \{0,1\} & \forall i & (14) \end{cases}$$

By relaxing Constraint (9) using Lagrangean multipliers λ_k ($k = 2, .., P$), the model becomes:

$$\begin{cases} \max_{\lambda_k \geq 0} \ \min_{t_k, u_k, U_i} \ \left[\sum_{i=1}^{n} w_i U_i - \sum_{k=2}^{P} \lambda_k \left(t_k - t_{k-1} - p_{\sigma(k-1)} u_{k-1} \right) \right] & & (15) \\ t_k - r_{\sigma(k)} \geq 0 & \forall k & (10) \\ t_k + p_{\sigma(k)} u_k - d_{\sigma(k)} - D_k(1 - u_k) \leq 0 & \forall k & (11) \\ \sum_{\substack{k=1 \\ \sigma(k)=i}}^{P} u_k + U_i = 1 & \forall i & (12) \\ u_k \in \{0,1\} & \forall k & (13) \\ U_i \in \{0,1\} & \forall i & (14) \end{cases}$$

To use Lagrangean relaxation, one needs to solve the previous model for given values of λ_k ($k = 2, .., P$). The objective function can be written:

$$\min_{t_k, u_k, U_i} \ \left[\sum_{i=1}^{n} w_i U_i + \sum_{k=2}^{P} \lambda_k p_{\sigma(k-1)} u_{k-1} + \lambda_2 t_1 + \sum_{k=2}^{P-1} (\lambda_{k+1} - \lambda_k) t_k - \lambda_P t_P \right] \qquad (16)$$

Because Constraint (9) has been relaxed, variables t_k are now independent and bounded through Constraints (10) and (11). Hence, if the coefficient of t_k ($\lambda_{k+1} - \lambda_k$) is positive, t_k will be chosen as small as possible to minimize the cost, *i.e.*, $r_{\sigma(k)}$ (because of (10)), and if the coefficient is negative, t_k will be chosen as large as possible, *i.e.*, $d_{\sigma(k)} + D_k - (p_{\sigma(k)} + D_k)u_k$ (because of (11)). Moreover, using (12), U_i can be replaced by $1 - \sum_{\substack{k=1 \\ \sigma(k)=i}}^{P} u_k$ in the criterion. Hence, (16) becomes:

$$\min_{u_k} \left[\sum_{i=1}^{n} w_i (1 - \sum_{\substack{k=1 \\ \sigma(k)=i}}^{P} u_k) + \sum_{k=2}^{P} \lambda_k p_{\sigma(k-1)} u_{k-1} + \lambda_2 r_{\sigma(1)} + \sum_{\substack{k=2 \\ (\lambda_{k+1}-\lambda_k)\geq 0}}^{P-1} (\lambda_{k+1} - \lambda_k) r_{\sigma(k)} + \right.$$

$$\left. \sum_{\substack{k=2 \\ (\lambda_{k+1}-\lambda_k)<0}}^{P-1} (\lambda_{k+1} - \lambda_k)(d_{\sigma(k)} + D_k - (p_{\sigma(k)} + D_k)u_k) - \lambda_P (d_{\sigma(P)} + D_P - (p_{\sigma(P)} + D_P)u_P) \right]$$

Note that the minimization now only depends on variables u_k. Since r_i and d_i are data, several terms of the previous expression can be ignored in the optimization:

$$\min_{u_k} \sum_{i=1}^{n} \left[\sum_{\substack{k=1;\sigma(k)=i \\ (\lambda_{k+1}-\lambda_k)\geq 0}}^{P} (\lambda_{k+1} p_i - w_i) u_k + \sum_{\substack{k=1;\sigma(k)=i \\ (\lambda_{k+1}-\lambda_k)<0}}^{P} (\lambda_{k+1} p_i - (\lambda_{k+1} - \lambda_k)(p_i + D_k - w_i) u_k \right]$$

or, after simplification,

$$\min_{u_k} \sum_{i=1}^{n} \left[\sum_{\substack{k=1;\sigma(k)=i \\ (\lambda_{k+1}-\lambda_k)\geq 0}}^{P} (\lambda_{k+1} p_i - w_i) u_k + \sum_{\substack{k=1;\sigma(k)=i \\ (\lambda_{k+1}-\lambda_k)<0}}^{P} (\lambda_k p_i - (\lambda_{k+1} - \lambda_k) D_k - w_i) u_k \right] \qquad (17)$$

where λ_1 and λ_{P+1} are parameters such that $\lambda_1 = \lambda_{P+1} = 0$.

To minimize the cost, and to satisfy Constraint (12), one has to determine, for every job J_i, the position k' such that $\sigma(k') = i$ with the smallest coefficient in (17), *i.e.*, $(\lambda_{k+1} p_i - w_i)$ or $(\lambda_k p_i + (\lambda_{k+1} - \lambda_k) D_k - w_i)$ depending on the sign of $(\lambda_{k+1} - \lambda_k)$. If the coefficient is positive, then $u_k = 0 \; \forall k$ such that $\sigma(k) = i$, and $U_i = 1$, and if the coefficient is negative, then $u_{k'} = 1$, $u_k = 0 \; \forall k \neq k'$ such that $\sigma(k) = i$, and $U_i = 0$.

Proposition 3 *Solving the relaxed problem can be done in $\mathcal{O}(P)$ time.*

The solution would be the same, *i.e.*, integral, if Constraints (13) and (14) were to be deleted. Hence, the Lagrangean relaxation bound is identical to the bound obtained by linear relaxation (see Parker and Rardin [10]). However, this bound can be determined faster, because every subproblem can be trivially solved. Actually, before implementing our Lagrangean relaxation algorithm, we performed some preliminary testing using linear relaxation with a standard and efficient LP package. The quality of the bound was better than all other formulations we had tested before (see [4]).

It is relatively easy to interpret the impact of the values of λ_k, p_i, or w_i. Increasing λ_k will force the associated Constraint (9) to be satisfied, *i.e.*, t_k to be chosen as large as possible and equal to $d_{\sigma(k)} + D_k - (p_{\sigma(k)} + D_k)u_k$ (u_k to 0), and t_{k-1} as small as possible and equal to $r_{\sigma(k-1)}$. Intuitively, a job with a large processing time that is set on time might force more jobs to be late than a job with a smaller processing time. Hence, it is natural to favor jobs with small processing times. This is consistent with (17), where the coefficient of u_k will increase with $p_{\sigma(k)}$, and has then more chances to become positive, thus inducing $u_k = 0$, *i.e.*, job $J_{\sigma(k)}$ is not set in position k. The exact opposite can be said about the weight, since the larger its weight, the more you want to sequence a job. Again, this is in accordance with (17), where the coefficient of u_k will decrease with $w_{\sigma(k)}$, and has then more chances to become negative, thus inducing $u_k = 1$, *i.e.*, job $J_{\sigma(k)}$ is set in position k.

The following algorithm is proposed to solve our problem using Lagrangean relaxation and subgradient optimization (see Parker and Rardin [10]).

Step 1 - Initialization of the Lagrangean variables λ_k: $\lambda_k^0 = f \frac{p_{\sigma(k)}}{n * p_{max} * w_{max} * w_{\sigma(k)}}$ $\forall k$, (where p_{max} (resp. w_{max}) is the largest processing time (resp. weight) among all jobs, and f a parameter), and $r = 0$.

Step 2 - Initialize the various parameters: $U_i = 1$, $coef(i) = \infty$ and $pos(i) = -1$ $\forall i$, $u_k = 0$ $\forall k$, $r = r + 1$, and $\lambda_1^r = \lambda_{P+1}^r = 0$.

Step 3 - Solve the relaxed problem:

Step 3.1 - For $k = 1, .., P$, if $\lambda_{k+1}^r - \lambda_k^r \geq 0$ then $coef = \lambda_{k+1}^r p_i - w_i$, else $coef = \lambda_k^r p_i - (\lambda_{k+1}^r - \lambda_k^r) D_k - w_i$.
If $coef < coef(\sigma(k))$, then $coef(\sigma(k)) = coef$ and $pos(\sigma(k)) = k$.

Step 3.2 - For $i = 1, .., n$, if $coef(i) \leq 0$ then $u_{pos(i)} = 1$ and $U_i = 0$.

Step 4 - Compute the **lower bound**:

$$LB = \sum_{i=1}^{n} \left[w_i + \sum_{\substack{k=1;\sigma(k)=i \\ (\lambda_{k+1}^r - \lambda_k^r) \geq 0}}^{P} \Big((\lambda_{k+1}^r - \lambda_k^r) r_i + (\lambda_{k+1}^r p_i - w_i) u_k \Big) \right.$$
$$\left. + \sum_{\substack{k=1;\sigma(k)=i \\ (\lambda_{k+1}^r - \lambda_k^r) < 0}}^{P} \Big((\lambda_{k+1}^r - \lambda_k^r)(d_i + D_k - D_k u_k) + (\lambda_k^r p_i - w_i) u_k \Big) \right]$$

Step 5 - Compute an **upper bound** by sequencing as many jobs as possible among the jobs J_i that are set on time in the solution associated to the lower bound, *i.e.*, such that $U_i = 0$.

Step 6 - Update the lagrangean variables λ_k:

$$\lambda_k^{r+1} = \max\left(0, \lambda_k^r - \rho_r \frac{t_k - t_{k-1} - p_{\sigma(k-1)} u_{k-1}}{||t_k - t_{k-1} - p_{\sigma(k-1)} u_{k-1}||}\right)$$

where $t_k = r_{\sigma(k)}$ if $(\lambda_{k+1}^r - \lambda_k^r) \geq 0$, and $t_k = d_{\sigma(k)} + D_k - (D_k + p_{\sigma(k)}) u_k$ otherwise. Update ρ_{r+1}.

Step 7 - If no stopping conditions are met, go to Step 2.

We use a simple and fast greedy algorithm to determine the upper bound in Step 5. From $k = 1$ to $k = P$, job $J_{\sigma(k)}$ is added to the sequence of early jobs if $u_k = 1$ and $J_{\sigma(k)}$ is on time. The finishing time of the current sequence is updated each time a new job is added.

Various parameters have to be initialized and adjusted to ensure the best convergence of the algorithm for different types of instances. After sd iterations without improvement, the parameter ρ_r is decreased by a factor of $100 \times (1 - red_\rho)\%$. Various stopping conditions are checked: maximum number of iterations $IterMax$, step ρ smaller than or equal to ρ_{min}, and of course if the optimum is found, *i.e.*, the lower and upper bounds are equal. The parameters chosen here could be adjusted to improve the results on some instances, but we decide to use generic parameters instead. After some preliminary testing, we chose the following values: $f = 0.4$, $\rho_1 = 1.6$, $sd = 40$, and $red_\rho = 0.9$. For the stopping conditions, we used $IterMax = 100\ 000$ and $\rho_{min} = 10^{-5}$. Actually, in our numerical experiments, the number of iterations is never larger than 20 000.

As already shown, every relaxed problem in Step 3 are solved very quickly, in $\mathcal{O}(P)$ time where P is not larger than $\frac{n(n+1)}{2}$. Hence, many iterations can be performed, even for large instances.

6 Improving the algorithm

Several improvements are proposed. The first one is based on a rewriting of the formulation. In the model, because of Constraint (9), Constraint (10) can be rewritten

$$t_k - rr_k \geq 0 \quad \forall k$$

where $rr_k = \max_{r=1,..,k}\ r_{\sigma(r)}$ are release dates *per position*. To include this change in the algorithm, it suffices to replace $r_{\sigma(k)}$ by rr_k.

A similar rewriting can be performed for Constraint (11) in the non-weighted case, where $D_k = 0\ \forall k$, as follows

$$t_k + p_{\sigma(k)} u_k - dd_k \leq 0$$

where $dd_k = \min_{r=k,..,P} \; d_{\sigma(r)}$ are due dates *per position.*

Although they do not improve the lower bound obtained by linear relaxation, and thus by Lagrangean relaxation, these changes often considerably speed up the algorithm by better updating the Lagrangean multipliers in Step 6. This is because the positions for a job are better differentiated whereas, in the original formulation, they all have similar Constraints (10). Hence, the algorithm will more quickly choose the best position(s) for a job, and will require less iterations to converge to the lower bound.

Another improvement uses the following property to tighten Constraint (9) in the model.

Proposition 4 *If, in the master sequence, J_i is before and after J_j, then there is an optimal schedule in which either the position k of J_i after (and generated by) J_j is not occupied or is occupied and such that $t_k \geq d_j - p_i$.*

Proof: We want to prove that if, in an optimal schedule S, the position k of J_i after J_j is occupied and $t_k \leq d_j - p_i$ then this schedule can be transformed into an equivalent optimal schedule S' in which J_i is sequenced before J_j (*i.e.*, position k is not occupied).

Since J_i is before and after J_j, we know that $r_i < r_j$ and $d_i > d_j$. Hence, moving J_i before J_j will just translate J_j and the jobs between J_j and J_i in S by p_i and, because $t_k \leq d_j - p_i$ in S, the completion time of the translated jobs will not be larger than d_j. By definition of the master sequence, and because position k is generated by J_j, the due dates of the jobs between J_j and J_i in S are larger than or equal to d_j. Thus, the schedule S' is feasible. □

Following Proposition 4, Constraints (10) can be tightened (the added term is positive) as follows:

$$t_k - rr_k - RR_k u_k \geq 0 \qquad \forall k$$

where $RR_k = \max(0, \min_{r=1,..,k-1} \; d_{\sigma(r)} - p_{\sigma(k)} - rr_k)$.

The relaxed problem in the Lagrangian relaxation changes accordingly by adding the new term in the objective function, and by considering the coefficient $(\lambda_{k+1} p_i + (\lambda_{k+1} - \lambda_k) RR_k - w_i)$ when $(\lambda_{k+1} - \lambda_k)$ is positive. Strengthening the constraints helps to improve the quality of the lower bound. Moreover, it also accelerates the algorithm by again better differentiating the positions.

7 The non-weighted case

The mixed-integer linear programming model defined in Section 3 can be enhanced for the non-weighted case, *i.e.*, $w_i = 1 \; \forall i$ following Theorem 2 in Section 3. The new model is given below:

$$
\begin{cases}
c^* = \min \ c = \sum_{i=1}^{n} U_i & & (18) \\
t_k - t_{k-1} - p_{\sigma(k-1)} u_{k-1} \geq 0 & k = 2, .., P & (19) \\
t_k - r_{\sigma(k)} \geq 0 & \forall k & (20) \\
t_k + p_{\sigma(k)} u_k - d_{\sigma(k)} \leq 0 & \forall k & (21) \\
\sum_{\substack{k=1 \\ \sigma(k)=i}}^{P} u_k + U_i = 1 & \forall i & (22) \\
u_k \in \{0, 1\} & \forall k & (23) \\
U_i \in \{0, 1\} & \forall i & (24)
\end{cases}
$$

Because $w_i = 1 \ \forall J_i$ and $D_k = 0$, $\forall k$, the objective function (17) can equivalently be written:

$$
\min_{u_k} \sum_{i=1}^{n} \sum_{\substack{k=1; \\ \sigma(k)=i}}^{P} \left(\max(\lambda_k, \lambda_{k+1}) p_i - 1\right) u_k \qquad (25)
$$

Remark 1 *In the non-weighted case, for a given job J_i, finding the position k', $\sigma(k') = i$, with the smallest coefficient in (17) is equivalent to finding the position with the smallest coefficient λ_{k+1} or λ_k, depending on the sign of $(\lambda_{k+1} - \lambda_k)$.*

In the Lagrangean relaxation algorithm described in Section 5, the following steps are modified:

Step 3.1 - For $k = 1, .., P$, if $\lambda^r_{k+1} - \lambda^r_k \geq 0$ then $coef = \lambda^r_{k+1} p_i - 1$, else $coef = \lambda^r_k p_i - 1$. If $coef < coef(\sigma(k))$, then $coef(\sigma(k)) = coef$ and $pos(\sigma(k)) = k$.

Step 4 - Compute the lower bound:

$$
\begin{aligned}
LB \;\; = \;\; n + \sum_{i=1}^{n} \Bigg[& \sum_{\substack{k=1; \sigma(k)=i \\ (\lambda^r_{k+1} - \lambda^r_k) \geq 0}}^{P} \Big((\lambda^r_{k+1} - \lambda^r_k) r_i + (\lambda^r_{k+1} p_i - 1) u_k \Big) \\
+ & \sum_{\substack{k=1; \sigma(k)=i \\ (\lambda^r_{k+1} - \lambda^r_k) < 0}}^{P} \Big((\lambda^r_{k+1} - \lambda^r_k)(d_i - p_i u_k) + (\lambda^r_k p_i - 1) u_k \Big) \Bigg]
\end{aligned}
$$

Moreover, the Kise *et al.*'s algorithm [6] can be used to compute the upper bound associated to the current value of the multipliers λ^r in Step 6. This is because, when the sequence in which jobs can be sequenced is fixed, *i.e.*, for a given permutation of the jobs, the optimal sequence of early jobs can be found using the Kise *et al.*'s algorithm. In our case, the set of jobs from which jobs have to be sequenced is the set of jobs J_i such that $U_i = 1$, and the fixed sequence is given by the positions k such that $u_k = 1$.

It is better to adjust the parameters for the algorithm when $w_i = 1, \forall i$. After multiple trials, we decided to use the following for all tested instances: $f = 0.4$, $\rho_1 = 0.05$, $sd = 60$, and $red_\rho = 0.92$. The same parameters are kept for the stopping conditions ($IterMax =$ 100 000 and $\rho_{Min} = 10^{-5}$).

8 Computational Results

Many test problems have been generated to evaluate our algorithm. For each value of n, the number of jobs, 160 instances have been randomly generated. The test program, written in C, is running on a SUN UltraSparc workstation.

Random generator For each job J_i, a processing time p_i is randomly generated in the interval $[1, 100]$ and a weight w_i is generated in the interval $[1, 10]$. As in [3], two parameters K_1 and K_2 are used, and taken in the set $\{1, 5, 10, 20\}$. Because we want data to depend on the number of jobs n, the release date r_i is randomly generated in the interval $[0, K_1 n]$, and the due date in the interval $[r_i + p_i, r_i + p_i + K_2 n]$. The algorithm was tested for $n \in \{20, 40, 60, 80, 100, 120, 140\}$. For each combination of n, K_1, and K_2, 10 instances are generated, *i.e.*, 160 instances for each value of n.

Results on the non-weighted case The Lagrangean relaxation algorithm was first ran on the $1|r_j|\sum U_j$ problem. In Table 2, results are reported for each value of n. The optimum is considered to be found when lower and upper bounds are equal. For $n = 60$, 66 out of 160 instances are optimally solved, *i.e.*, 41.3%. The CPU time necessary to find the best bounds is also reported. For $n = 80$, the mean CPU time is about 1 minute. To evaluate the efficiency of both bounds, the gap between the upper and lower bounds is also measured and reported in the last three columns of the table. This gap is expressed in number of jobs. For $n = 100$, the average gap is close to 2 jobs. The standard deviation and maximum gap are also given in the table.

The results are good, although the average duality gap increases quickly when n is larger than 100. This is mostly because it is large for specific sets of instances, as attested by the large standard deviation. Remember that we decided to use the same parameters for our algorithm for every test instance, independently of n, K_1, or K_2. The algorithm does not perform so well when the master sequence is long. Looking at Proposition 2, this happens when there are many pairs of jobs (J_i, J_j) such that $r_i < r_j$ and $d_i > d_j$. This is the case when K_2 is large, and even more when K_1 is also small. The same analysis holds for the CPU time, since the time to solve the relaxed problem at every iteration directly

Nb of jobs	Optimum Nb	(%)	CPU Time (sec) Mean	StDev	Max	Gap Mean	StDev	Max	Gap (%) Mean
$n = 20$	85	53.1%	3.06	2.45	15.62	0.54	0.67	3	2.70
$n = 40$	75	46.9%	13.18	9.70	49.16	0.65	0.71	3	1.63
$n = 60$	66	41.3%	33.48	23.41	98.88	0.85	0.99	5	1.42
$n = 80$	55	34.4%	66.63	49.11	216.78	1.07	1.22	6	1.34
$n = 100$	26	16.3%	138.57	107.73	432.17	2.34	2.99	18	2.34
$n = 120$	11	6.9%	226.33	182.25	663.83	6.39	7.66	37	5.33
$n = 140$	8	5.0%	359.49	275.53	938.01	11.57	12.96	44	8.26

Table 2: Results on the non-weighted case.

depends on the length of the master sequence P. This is why the CPU time average and standard deviation increase with the number of jobs. Table 3 reports the results and the length of the master sequence for $n \in \{100, 120, 140\}$ and $K_2 \in \{1, 5, 10, 20\}$. Note that, for $K_2 = 20$, the mean CPU time and the mean gap are approximatively two times larger than in Table 2.

Nb of jobs	Value of K_2	Length of σ Mean	StDev	CPU Time (sec) Mean	StDev	Gap Mean	StDev
$n = 100$	1	147.95	53.11	29.38	18.32	2.67	1.67
	5	991.80	530.18	84.27	30.61	1.00	1.11
	10	1466.38	545.92	153.14	51.86	1.43	2.45
	20	1877.05	429.40	287.48	71.67	4.28	4.45
$n = 120$	1	236.05	111.05	36.30	16.53	2.62	2.10
	5	1502.25	843.39	129.89	55.72	2.00	2.74
	10	2190.62	844.37	253.41	89.16	5.25	5.86
	20	2793.60	654.81	485.73	88.01	15.70	8.27
$n = 140$	1	350.27	186.30	55.32	29.94	2.50	1.93
	5	2077.15	1154.75	202.33	68.82	3.85	4.89
	10	2980.35	1132.38	437.93	130.68	12.57	12.61
	20	3791.62	891.07	742.39	99.50	27.38	9.75

Table 3: Sensitivity of the results to parameter K_2.

In [5], we propose a branch-and-bound procedure which is only valid for the non-weighted problem. This exact method also uses the notion of master sequence, and has been tested on the same set of instances. In a maximum running time of one hour, more than 95% of 140-job instances are solved to optimality. Hence, it is possible to compare the bounds given by our Lagrangean relaxation algorithm to the optimal solution for test instances that are optimally solved by our exact procedure. In Table 4, we compare the two bounds for instances of more than 80 jobs with the optimal solution.

Nb of jobs	Lagrangean Lower Bound Opt. found	Gap with optimum Mean	StDev	Max	Lagrangean Upper Bound Opt. found	Gap with optimum Mean	StDev	Max
$n = 80$	43.3%	0.87	1.07	5	84.1%	0.20	0.50	3
$n = 100$	25.5%	1.85	2.40	16	68.2%	0.52	1.09	7
$n = 120$	15.2%	4.46	5.60	27	35.5%	2.38	3.06	12
$n = 140$	14.9%	9.19	10.10	38	26.9%	3.88	4.41	20

Table 4: Comparing with the optimal solution

For both the lower and upper bounds, the results are reported as follows: the first column gives the percentage of cases where the bound and the optimal solution are equal, and the next three columns give the mean, the standard deviation and the maximum of the gap between the bound and the optimal solution. These figures are expressed in number of jobs. Even for the largest instances ($n = 140$), the upper bound is very good on average, about 4 jobs more than the optimal solution (which corresponds to an error of less than 3%). However, the standard deviation becomes rather large, which emphasizes again the large variance observed on the CPU time and the duality gap.

Better results could be obtained, when the gap is very large, by adjusting the parameters of the Lagrangean algorithm. We did it for $n = 140$, $K_1 = 1$ and $K_2 = 20$, where the largest gaps are observed. Using the generic parameters ($f = 0.4$, $\rho_1 = 0.05$, $sd = 60$, and $red_\rho = 0.92$), the average difference between the lower and upper bounds for the 10 instances is 38.4. By modifying only ρ_1 ($\rho_1 = 0.5$), the mean gap is reduced to 3.4 (more than 10 times smaller!).

Results on the weighted case Weights are randomly generated in the interval $[1, 10]$. Results are reported in Table 5. The Lagrangean relaxation algorithm seems to be more efficient than in the non-weighted case. When n is large, the bounds are obtained faster (184.66 seconds on average *vs* 359.49 for $n = 140$), and the average gap between the two bounds is also reduced. The last column of Table 5 give the gap between the two bounds expressed in %. This gap can be compared to the one given in Table 2.

Results on instances of small size are better in the non-weighted case than in the weighted case. However, it becomes the opposite when the number of jobs increases ($n = 120$ and $n = 140$). For $n = 140$, the gap in the weighted case is less than 4%, whereas it is more than 8% in the non-weighted case. Moreover, in nearly all the cases, the CPU time is smaller in the weighted case, and the difference amplifies when n increases. We do not give a table equivalent to Table 3 for the weighted case, since it would be very similar and would not bring much.

Nb of jobs	CPU Time (sec) Mean	StDev	Max	Gap Mean	StDev	Max	Gap (%) Mean
$n = 20$	4.18	2.28	12.46	4.46	3.54	20	4.07
$n = 40$	13.20	8.58	37.88	7.65	7.65	31	3.36
$n = 60$	30.66	20.39	83.28	10.26	6.37	41	3.09
$n = 80$	56.49	38.79	184.43	11.40	7.31	36	2.56
$n = 100$	86.21	58.83	231.57	14.82	8.78	47	2.70
$n = 120$	130.96	90.47	378.59	18.17	11.49	72	2.74
$n = 140$	184.66	130.34	496.85	29.18	21.46	127	3.82

Table 5: Results on the weighted case.

Let us give a tentative explanation of the better efficiency of the algorithm in the weighted case. Weights help to differentiate between two jobs that could be both sequenced, but not together, in an optimal solution in the non-weighted case. Hence, the objective function will be less "flat", *i.e.*, there will be less identical solutions associated to the same value of the objective function. The Lagrangean relaxation algorithm reaches more quickly its lower bound, whose quality is improved.

As in the non-weighted case, better results could be obtained by adjusting the parameters of the Lagrangean algorithm. We did it again for $n = 140$, $K_1 = 1$ and $K_2 = 20$. The average duality gap for the 10 instances reduces from 66.4, when using the generic parameters ($f = 0.4$, $\rho_1 = 1.6$, $sd = 40$, and $red_\rho = 0.9$), to 15.8 by modifying only ρ_1 and sd ($\rho_1 = 2.6$ and $sd = 80$).

9 Conclusion

This paper considers a single-machine scheduling problem in which the objective is to minimize the weighted number of late jobs. Based on the definition of the *master sequence*, a new and efficient mixed-integer linear programming formulation is derived. By relaxing some coupling constraints using Lagrangean multipliers, the resulting problem becomes easily solvable. A Lagrangean relaxation algorithm is proposed and improved. Numerical experiments have been performed on an extended set of test instances for the non-weighted case, and for the weighted case, and the algorithm performs well for problems with more than 100 jobs.

To our knowledge, our Lagrangean relaxation algorithm is the first method proposed to solve the problem $1|r_j|\sum w_j U_j$. We would like to improve the algorithm, in particular the number of iterations required to obtain the lower bound, by for instance using dual ascent instead of subgradient optimization when updating the Lagrangean multipliers.

The master sequence has also been used in a branch-and-bound method to solve the

$1|r_j|\sum U_j$ problem *i.e.*, the non-weighted case [5]. It would be interesting to investigate other problems where the notion of master sequence could be applied. For instance, we believe it can be used to tackle the case where jobs can be processed in batches (although not with families, see Crauwels *et al.* [2]).

References

[1] Baptiste, P., Le Pape, C. and Péridy, L. (1998), Global Constraints for Partial CSPs: A Case Study of Resource and Due-Date Constraints. *4th International Conference on Principles and Practices of Constraint Programming*, Pisa, Italy, To appear in Lecture Notes in Computer Science.

[2] Crauwels H.A.J., Potts C.N. and Van Wassenhove L.N. (1996). Local Search Heuristics for Single Machine Scheduling with Batching to Minimize the Number of Late Jobs. *European Journal of Operational Research* **90**, 200-213.

[3] Dauzère-Pérès, S. (1995). Minimizing late jobs in the general one machine scheduling problem. *European Journal of Operational Research* **81**, 134-142.

[4] Dauzère-Pérès, S. and Sevaux, M. (1998). Various Mathematical Programming Formulations for a General One Machine Sequencing Problem. Research report 98/3/AUTO, Ecole des Mines de Nantes, France.

[5] Dauzère-Pérès, S. and Sevaux, M. (1999). An Exact Method to Minimize the Number of Tardy Jobs in Single Machine Scheduling. Research report 99/6/AUTO, Ecole des Mines de Nantes, France.

[6] Kise, H., Ibaraki, T. and Mine, H. (1978). A solvable case of the one-machine scheduling problem with ready and due times. *Operations Research* **26**(1), 121-126.

[7] Lawler, E.L. (1994). Knapsack-Like Scheduling Problems, the Moore-Hodgson Algorithm and the 'Tower of Sets' Property. *Mathematical Computer Modelling* **20**(2), pp 91-106.

[8] Lenstra, J.K., Rinnooy Kan, A.H.G. and Brucker, P. (1977). Complexity of Machine Scheduling Problems. *Annals of Discrete Mathematics*, **1** 343-362.

[9] Moore, J.M. (1968). A n job, one machine sequencing algorithm for minimizing the number of late jobs. *Management Science* **15**(1), 102-109.

[10] Parker R.G. and Rardin R.L. (1988). Discrete Optimization, *Academic Press*.

[11] Potts, C.N., and Van Wassenhove, L.N. (1988). Algorithms for Scheduling a Single Machine to Minimize the Weighted Number of Late Jobs. *Management Science* **34**(7), 834-858.

[12] Villarreal, F.J. and Bulfin, R.L. (1983). Scheduling a single machine to minimize the weighted number of tardy jobs. *IIE Transactions* **15**, pp 337-343.