

# Sequencing a Single Machine with Due Dates and Deadlines: an ILP-based Approach to Solve Very Large Instances

Philippe Baptiste

CNRS LIX, Ecole Polytechnique, France, philippe.baptiste@polytechnique.fr

Federico Della Croce, Andrea Grosso

D.A.I., Politecnico di Torino, Italy, {federico.dellacroce, grosso}@polito.it

Vincent T'kindt

Laboratoire d'Informatique, Université de Tours, France, tkindt@univ-tours

We consider the problem of minimizing the weighted number of tardy jobs on a single machine where each job is also subject to a deadline that cannot be violated. We propose an exact method based on a compact integer linear programming formulation of the problem and an effective reduction procedure, that allows to solve to optimality instances with up to 30,000 jobs in size.

*Keywords:* Machine Scheduling, Algorithmics.

## 1 Introduction

This paper deals with the problem of minimizing the weighted number of tardy jobs on a single machine when both due dates and deadlines are specified for jobs. In the classical three-fields notation the problem is denoted  $1|\bar{d}_i|\sum w_i U_i$ : a job set  $N = \{1, 2, \dots, n\}$  is given, with weights, processing times, due dates and deadlines  $w_i$ ,  $p_i$ ,  $d_i$  and  $\bar{d}_i$  respectively, for all  $i \in N$ . The goal is to find a sequence  $\sigma^*$  such that  $C_i \leq \bar{d}_i$  for all the job completion times  $C_i$ ,  $i \in N$ , and  $F(\sigma^*) = \sum \{w_i : C_i > d_i\}$  is minimum — or equivalently,  $f(\sigma^*) = \sum \{w_i : C_i \leq d_i\}$  is maximum. The jobs are executed without idle time or preemption, and all of them are available from time 0 on.

From a complexity point of view, the  $1|\bar{d}_i|\sum w_i U_i$  problem is known to be NP-hard even if  $w_i = 1$ , for all  $i \in N$  [1], or if there are no deadlines [2], but it is still unclear whether it is NP-hard in the strong or in the ordinary sense. On the other hand, the basic  $1|\sum U_i$  problem is well known to be polynomially solvable by Moore's algorithm [3]. If release dates are present, the  $1|r_i|\sum U_i$  problem is already NP-hard in the strong sense [4]. Very little work has appeared in literature for the  $1|\bar{d}_i|\sum w_i U_i$  problem, a state-of-the-art exact algorithm being a branch and bound presented in [6] able to solve instances with up to 300 jobs.

A richer literature is available for the deadline-free variant: Potts and Van Wassenhove [5] gave a branch and bound algorithm for solving instances with up to 1,000 jobs, while M'Hallah and Bulfin [8] propose an exact algorithm capable of handling instances with up to 2,500 jobs. Several papers are also available for the variants with release dates (see [9, 10, 11]). However, the presence of release dates completely changes the structure of the problem.

We note that most of the works on this scheduling problem explicitly avoid to manage the LP relaxation by standard LP tools: in [5] an ILP model is presented, but a specific dynamic programming algorithm is used for solving the relaxation; in [6] no LP is used, the bound being obtained by dynamic programming and state-space relaxation; even in more recent papers like [8] a lagrangian approach is used, and the simplex method is avoided.

In this paper we present a compact integer linear programming formulation and, explicitly relying on an LP/ILP solver with minimum additional procedures, we design a simple exact algorithm

that is able to solve to optimality all instances with up to 30,000 jobs (within 275 seconds on the average) for the  $1|\bar{d}_i|\sum w_iU_i$  problem and up to 50,000 jobs (within 316 seconds on the average) for the special deadline-free case  $1|\sum w_iU_i$ .

Throughout this Conference paper, we omit the proofs for conciseness.

## 2 ILP model and problem reduction

### 2.1 Basic properties and model

We first recall that a feasible solution for  $1|\bar{d}_i|\sum w_iU_i$  is immediately defined by selecting an *early set* of jobs  $E \subseteq N$  required to be early: each job  $i \in N$ , is then required to be completed within a maximum completion time

$$D_i = \begin{cases} d_i & \text{if } i \in E, \\ \bar{d}_i & \text{if } i \in N \setminus E. \end{cases}$$

A feasible sequence with early set  $E$  is a sequence where  $C_i \leq D_i$  for all  $i \in N$ . We recall that such feasible sequences exist iff the particular sequence where the jobs appear in nondecreasing order of  $D_i$  is feasible.

Define  $B_t = \{i \in N : \bar{d}_i \leq t\}$ ,  $A_t = \{i \in N : d_i > t\}$ , and let  $T = (t_1, t_2, \dots, t_m)$  be the nondecreasing sequence of the relevant time points in the problem with  $t \in T$  iff  $t = d_i$  or  $t = \bar{d}_i$  for some  $i \in N$ . The optimal set of early jobs can be obtained by solving the following ILP model. Define binary variables  $x_i$ ,  $i \in N$ , such that  $x_i = 1$  iff job  $i$  is early.

$$\text{maximize } z = \sum_{i \in N} w_i x_i \tag{1}$$

subject to

$$\sum_{i \in B_t} p_i + \sum_{i \in N \setminus (B_t \cup A_t)} p_i x_i \leq t \quad t \in T \tag{2}$$

$$x_i \in \{0, 1\}, \quad i \in N \tag{3}$$

With modern ILP solvers and hardware, model (1)–(3) is able to handle fairly large instances: in our experiments, all the considered examples with up to 4,000 jobs within reasonable average CPU times, although the time exceeded 1,000 seconds for the hardest instances. The failures for higher sizes were caused essentially by lack of memory — note that the number of nonzeros in the constraints (2) exhibits a  $\mathcal{O}(n^2)$  growth in the worst case.

In our experience the solver can largely benefit from a problem preprocessing that is able to significantly reduce the number of jobs. Suppose a given job  $j \in N$  is known to be early or tardy in an optimal solution: this defines its  $D_j = d_j$  or  $\bar{d}_j$ . We define a *reduced* problem formulated on the job set  $N' = N \setminus \{j\}$ , with modified data

$$p'_i \equiv p_i, \quad w'_i \equiv w_i \quad i \in N', \tag{4}$$

$$d'_i = \begin{cases} \min \{d_i, D_j - p_j\} & \text{if } d_i \leq D_j \\ d_i - p_j & \text{if } d_i > D_j \end{cases} \quad i \in N', \tag{5}$$

$$\bar{d}'_i = \begin{cases} \min \{\bar{d}_i, D_j - p_j\} & \text{if } \bar{d}_i \leq D_j \\ \bar{d}_i - p_j & \text{if } \bar{d}_i > D_j \end{cases} \quad i \in N'. \tag{6}$$

The following result generalizes the *reduction theorem* presented in [5] for the deadline-free special case.

**Property 1** *There exists a feasible sequence with early set  $E$  iff there exists a feasible sequence with early set  $E' = E \setminus \{j\}$  for the reduced problem.*

Property 1 allows to remove job  $j$  from consideration and solve the reduced problem only, without loss of optimality.

We iteratively identify early or tardy jobs by variable fixing techniques, removing them from the problem by means of Property 1. Components needed for such reduction procedure are a quick method for solving the LP relaxation of (1)–(3) and a heuristic solution.

## 2.2 Solving the LP relaxation

For the LP relaxation, instead of using the dense formulation (1)–(3) we introduce a maximum profit flow problem. Define a graph  $G(N \cup T, A)$  where nodes represent jobs and time points, whereas the arc set  $A$  is defined as

$$A = A_d \cup A_{\bar{d}} \cup A_T,$$

where

$$\begin{aligned} A_d &= \{(i, t_k) : i \in N, t_k \in T, t_k = d_i\}, \\ A_{\bar{d}} &= \{(i, t_k) : i \in N, t_k \in T, t_k = \bar{d}_i\}, \\ A_T &= \{(t_k, t_{k+1}) : t_k, t_{k+1} \in T, k = 1, \dots, m-1\}. \end{aligned}$$

We formulate on  $G$  the following flow problem.

$$\text{maximize } z = \sum_{(i, d_i) \in A_d} \frac{w_i}{p_i} y_{i, d_i} \quad (7)$$

subject to

$$y_{i, d_i} + y_{i, \bar{d}_i} = p_i \quad i \in N \quad (8)$$

$$y_{t_1, t_2} - \sum_{(i, t_1) \in A_d \cup A_{\bar{d}}} y_{i, t_1} = 0 \quad (9)$$

$$y_{t_k, t_{k+1}} - y_{t_{k-1}, t_k} - \sum_{(i, t_k) \in A_d \cup A_{\bar{d}}} y_{i, t_k} = 0 \quad k = 2, \dots, m-1 \quad (10)$$

$$-y_{t_{m-1}, t_m} - \sum_{(i, t_m) \in A_d \cup A_{\bar{d}}} y_{i, t_m} = -\sum_{i \in N} p_i \quad (11)$$

$$y_{t_k, t_{k+1}} \leq t_k \quad (t_k, t_{k+1}) \in A_T \quad (12)$$

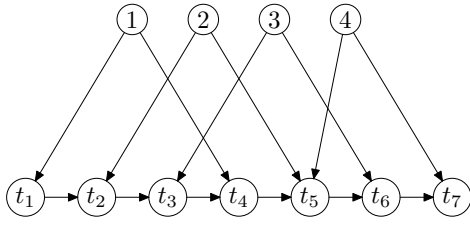
$$y_{ij} \geq 0, (i, j) \in A, \quad k = 1, \dots, m-1. \quad (13)$$

The shape of  $G$  is sketched in Figure 1. Constraints (8)–(11) are flow balance constraints and constraints (12) are capacity constraints. Nodes  $i \in N$  are sources injecting  $p_i$  units of flow each in the network, node  $t_m$  is the unique sink of the network, and the problem is balanced. All arcs have zero cost except the arcs  $(i, d_i) \in A_d$  having cost  $\frac{w_i}{p_i}$ . All arcs have infinite capacity except the arcs  $(t_k, t_{k+1}) \in A_T$  that have finite capacities  $t_k$ .

**Property 2**  $(y_{ij} : (i, j) \in A)$  is a feasible flow of value  $z$  for (7)–(13) iff

$$x_i = \frac{y_{i, d_i}}{p_i} \quad (i \in N)$$

is a feasible solution of value  $z$  for (1)–(3).



$$T = (t_1, \dots, t_7) = (d_1, d_2, d_3, \bar{d}_1, \bar{d}_2 = d_4, \bar{d}_3, \bar{d}_4)$$

Node balances  $b_i$ 's are as follows:

$$b_1 = p_1,$$

$$b_2 = p_2$$

$$b_3 = p_3$$

$$b_4 = p_4$$

$$b_{t_1}, \dots, b_{t_6} = 0$$

$$b_{t_7} = -(p_1 + p_2 + p_3 + p_4)$$

Figure 1: Graph  $G(N \cup T, A)$  for a four-jobs example.

Property 2 establishes a one-to-one correspondence between feasible solutions of the LP relaxation of (1)–(3) and (7)–(13). In our experiments we kept solving the flow model (7)–(13) that requires  $\mathcal{O}(n)$  (instead of quadratic) space, thus overcoming the memory problems experienced with formulation (1)–(3) on large instances.

### 2.3 Heuristic solution and core problem

Once the LP relaxation is solved, we need to determine a heuristic solution of (1)–(3). Preliminary testing with a constructive procedure (first an initial solution having as early all the jobs corresponding to variables having value 1 in the lower bound solution is generated and then all the other jobs are tested one at a time for inclusion in the early set according to a greedy rule) did not reach satisfactory results. Notice that in all tests most of the variables present an integer value in the LP relaxation solution. Based on this consideration, we propose here a heuristic solution corresponding to the optimal solution of a *core problem*.

To this extent we introduce the following dominance property (easily provable by interchange argument).

**Property 3** Let  $p_i \leq p_j, d_i \geq d_j, \bar{d}_i \leq \bar{d}_j$  and  $w_i \geq w_j$  with at least one strict inequality. Then:

- if  $i$  is tardy also  $j$  must be tardy
- if  $j$  is early also  $i$  must be early.

Let  $X^* = [x_1^*, \dots, x_n^*]$  be the optimal solution of the linear programming continuous relaxation of the problem. The set  $C$  of variables (jobs) in the core problem is determined by selecting

- all variables presenting non integer value in  $X^*$ ,
- all variables  $x_j$  set to 0 in  $X^*$  and that are non dominated according to Property 3 by any other variable  $x_i$  set to 0 in  $X^*$ ,
- all variables  $x_j$  set to 1 in  $X^*$  and that are dominated according to Property 3 by every other variable  $x_i$  set to 1 in  $X^*$ ,

All the jobs  $j \notin C$  are fixed early (if  $x_j = 1$ ) or tardy (if  $x_j = 0$ ) and removed via Property 1.

The core problem size has in all tests been always less than 5% of the original problem size and is therefore solvable to optimality in very short time by the ILP solver. In order to further improve the quality of the heuristic solution, we perform then a local search phase that uses the

optimal solution of the core problem as initial solution. Two feasible solutions  $(\bar{x}_1, \dots, \bar{x}_n)$  and  $(\tilde{x}_1, \dots, \tilde{x}_n)$  are neighbours if  $\sum_{i \in N} |\bar{x}_i - \tilde{x}_i| = 2$ , i.e. a job tardy and a job early are swapped. The corresponding neighbourhood can be generated and evaluated in  $\mathcal{O}(n^3)$  time, and a best-improve exploration is adopted. Once again, to save time, a job  $j$  tardy (early) is swapped iff  $\exists i : x_i^* = 0$  ( $x_i^* = 1$ ) dominating  $j$  (dominated by  $j$ ) according to Property 3. In this way, in practice, always much less than 1% of the neighborhood is explored. Let denote by  $\bar{z}$  the solution value provided by the local search phase.

## 2.4 Fixing jobs

Job fixing is performed by applying variable-fixing techniques from Integer Linear Programming. Given an optimal basis  $B^*$  for (1)–(3), let

$$\begin{aligned} z &= z^* + \sum_{x_i \notin B^*} \bar{c}_i x_i \\ x_j &= x_j^* + \sum_{x_i \notin B^*} \alpha_{ji} x_i \quad (x_j \in B^*) \\ x_i &\geq 0, \quad i \in N \end{aligned}$$

be the corresponding reformulation with  $\bar{c}_i$  being the reduced cost of variable  $x_i$  and  $\alpha_{ji}$  being the updated coefficient of variable  $x_i$  in the constraint related to the in-base variable  $x_j$ . We apply the following fixing rules.

For nonbasic variables:

(R1) fix  $x_i = 0$  (job  $i$  tardy) if  $\bar{c}_i < 0$  and  $z^* + \bar{c}_i \leq \bar{z}$ ,

(R2) fix  $x_i = 1$  (job  $i$  early) if  $\bar{c}_i > 0$  and  $z^* - \bar{c}_i \leq \bar{z}$ .

For basic variables  $x_j \in B^*$ , we compute the branching penalties  $u_j, l_j$  for branching at  $x_j = 1$  and at  $x_j = 0$  respectively (often indicated as *pseudo-costs* — see, for instance, [12]):

$$u_j = (x_j^* - 1) \min \left\{ -\frac{\bar{c}_i}{\alpha_{ji}} : x_i \notin B^*, \bar{c}_i \alpha_{ji} \leq 0, \alpha_{ji} \neq 0 \right\}; \quad (14)$$

$$l_j = -x_j^* \min \left\{ \frac{\bar{c}_i}{\alpha_{ji}} : x_i \notin B^*, \bar{c}_i \alpha_{ji} \geq 0, \alpha_{ji} \neq 0 \right\}; \quad (15)$$

then we apply

(R3) fix  $x_j = 0$  if  $z^* + u_j \leq \bar{z}$ ,

(R4) fix  $x_j = 1$  if  $z^* + l_j \leq \bar{z}$ .

Although sometimes overlooked in textbooks, this technique is known in integer programming as well as in constraint programming (see [7, 13] for an application to scheduling problems). Also, note that  $u_j, l_j$  are easily computed on the maximum profit flow problem — computing penalties for setting  $y_{j,d_j} = p_j, y_{j,d_j} = 0$ .

The complete reduction procedure works as follows — fixed jobs in steps (2) and (3) below are removed by means of Property 1.

1. the LP relaxation is solved via model (7)–(13),
2. jobs are fixed if possible, by rules (R1) and (R2),

3. jobs are fixed if possible, by rules (R3) and (R4),
4. if (R3), (R4) were successful in fixing jobs, steps (1)–(3) are reiterated on the reduced problem, else the procedure stops.

### 3 An exact algorithm

The solution algorithm we considered is a depth-first branch and bound procedure relying on problem reduction and model (1)–(3). We observed that the ILP model already reaches good performances for fairly large  $n$ : the limit seems to be memory consumption, due to the quadratic growing of the constraint matrix size as  $n$  increases. We then implemented a depth-first enumeration scheme which incorporates the reduction procedure sketched in Section 2.4: at each node, the LP relaxation of model (1)–(3) is solved via the equivalent network flow model, and the reduction procedure is applied. If the reduced problem allows to build an instance of (1)–(3) with no more than  $1.4 \cdot 10^7$  nonzeros, such integer program is solved directly by calling the ILP solver. Otherwise the fractional variable  $x_i$  with largest max-min pseudo-cost, namely the one with  $\max_{i:0 < x_i^* < 1} \{\min\{|l_i|, |u_i|\}\}$  value, is selected and binary branching is performed by setting  $x_i = 0$  (job  $i$  tardy) and  $x_i = 1$  (job  $i$  early) in the descendant nodes. The resulting algorithm is quite simple, and adds a minimal machinery on top of the ILP solver. The performances of the algorithm are further enhanced by incorporating Property 3 in the branching phase.

## 4 Computational results

### 4.1 Test instances

Following [6], we considered ten classes of instances structured as follows.

- The values for processing times  $p_1, \dots, p_n$  and weights  $w_1, \dots, w_n$  are integers drawn randomly from the uniform distribution  $[1, 100]$ . We note that in [6] the range was  $[1, 10]$ .
- The due dates  $d_1, \dots, d_n$  are integers drawn randomly from the uniform distribution  $[Pu, Pv]$ , with  $P = \sum_{i \in N} p_i$ . Ten pairs of values for  $u, v$  were considered:  $u \in \{0.1, 0.3, 0.5, 0.7\}$ ,  $v \in \{0.3, 0.5, 0.7, 0.9\}$ ,  $u < v$ .
- The deadlines  $\bar{d}_1, \dots, \bar{d}_n$  are integers drawn for each job  $i$  from the uniform distribution  $[d_i, 1.1P]$ .

We generated 20 examples for each  $u, v$  class — thus creating batches of 200 examples — with size  $n$  ranging from 1000 to 15,000. All the testing ran on a Pentium IV PC with 3 GHz clock and 1GB memory. The ILP solver used is XPRESS-MP by Dash Optimization.

### 4.2 Results

As previously remarked, model (1)–(3) already reaches fairly good performances for  $n \leq 4000$  — see Table 1.

The solver did not manage to solve the whole  $n = 5000$  batch because of memory problems. The enumerative algorithm was able to solve all the batches up to  $n = 30000$ , the worst case being a 25,000- job instance with distribution  $u = 0.1$ ,  $v = 0.5$  that required 3982 seconds. The results are summarized in Table 2.

CPU Time (s)		
$n$	avg	max
1000	7.2	21.3
2000	44.8	170.9
3000	135.3	577.2
4000	289.7	1278.5

Table 1: Performances of model (1)–(3).

$n$	CPU Time (s)		NODES		UB-GAP%		LB-GAP%	
	avg	max	avg	max	avg	max	avg	max
1000	0.9	4.7	1.0	1	0.008	0.084	0.001	0.037
2000	2.2	22.6	1.0	1	0.003	0.041	0.001	0.042
3000	3.6	12.4	1.0	1	0.002	0.026	$< 10^{-3}$	0.016
4000	5.9	20.6	1.0	1	0.002	0.027	$< 10^{-3}$	0.009
5000	8.4	42.2	1.0	1	0.001	0.013	$< 10^{-3}$	0.006
6000	12.6	71.6	1.0	1	0.001	0.013	$< 10^{-3}$	0.011
7000	16.4	116.2	1.0	1	0.001	0.011	$< 10^{-3}$	0.007
8000	20.2	65.3	1.0	1	0.001	0.010	$< 10^{-3}$	0.003
9000	26.5	238.4	1.0	1	0.001	0.007	$< 10^{-3}$	0.006
10000	32.1	197.2	1.0	3	0.001	0.007	$< 10^{-3}$	0.007
15000	79.5	1858.2	1.2	21	$< 10^{-3}$	0.005	$< 10^{-3}$	0.008
20000	139.0	2957.9	1.5	55	$< 10^{-3}$	0.004	$< 10^{-3}$	0.004
25000	204.5	3981.7	1.8	57	$< 10^{-3}$	0.003	$< 10^{-3}$	0.001
30000	274.8	3491.1	2.1	81	$< 10^{-3}$	0.002	$< 10^{-3}$	0.001

Table 2: Performances of the enumerative algorithm.

The columns NODES refer to nodes of the enumeration scheme, i.e. nodes where the reduction procedure was applied, then either branch or ILP solution occurred (the related instance of (1)–(3) presented no more than  $1.4 \cdot 10^7$  nonzeros). Columns  $UB - GAP$  and  $LB - GAP$  indicate the percentage relative deviations  $\frac{UB-OPT}{OPT}$  and  $\frac{OPT-LB}{OPT}$  of upper and lower bounds at the root node from the optimal solution value. For  $n = 1000, 2000, 3000, 4000$ , the CPU times (both average and worst-case) were drastically reduced with respect to those in Table 1. Also, we note that for  $n$  up to 9000 jobs, the enumeration scheme only needed to handle the root node.

The proposed algorithm exhibits extremely good performances also when applied to the deadline-free special case  $1 | \sum w_i U_i$  with the results summarized in Table 3. For this case, the algorithm solves to optimality all instances with up to 50,000 jobs, strongly outperforming the state of the art algorithms (see [8]).

## 5 Conclusions

We have proposed for the  $1 | \bar{d}_i | \sum w_i U_i$  problem an exact procedure that is able to solve to optimality very large size instances by exploiting a compact ILP formulation of the problem. As an outcome of this work, we remark that, also for scheduling problems, whenever structural properties allow to derive “good” LP/ILP formulations, then already commercial solvers standalone can handle reasonably large size instances. Moreover, adding minimal additional procedures that rely on some

$n$	CPU Time (s)		NODES		UB-GAP%		LB-GAP%	
	avg	max	avg	max	avg	max	avg	max
1000	0.6	4.1	1.0	1	0.003	0.069	0.001	0.037
2000	1.2	4.5	1.0	1	0.001	0.028	$< 10^{-3}$	0.008
3000	2.0	5.8	1.0	1	0.001	0.022	$< 10^{-3}$	0.005
4000	3.3	8.8	1.0	1	0.001	0.022	$< 10^{-3}$	0.005
5000	4.9	11.1	1.0	1	$< 10^{-3}$	0.009	$< 10^{-3}$	0.004
10000	19.3	211.7	1.0	1	$< 10^{-3}$	0.005	$< 10^{-3}$	0.007
15000	37.3	84.4	1.0	1	$< 10^{-3}$	0.003	$< 10^{-3}$	0.001
20000	61.2	233.3	1.0	7	$< 10^{-3}$	0.003	$< 10^{-3}$	0.005
25000	93.9	459.7	1.1	11	$< 10^{-3}$	0.002	$< 10^{-3}$	0.003
30000	128.4	193.8	1.0	1	$< 10^{-3}$	0.001	$< 10^{-3}$	0.001
35000	177.1	1671.6	1.4	59	$< 10^{-3}$	0.001	$< 10^{-3}$	0.002
40000	215.6	966.0	1.2	43	$< 10^{-3}$	0.002	$< 10^{-3}$	0.001
45000	281.1	537.0	1.0	7	$< 10^{-3}$	0.001	$< 10^{-3}$	0.002
50000	315.3	736.0	1.2	17	$< 10^{-3}$	0.002	$< 10^{-3}$	0.001

Table 3: Performances of the enumerative algorithm on instances of the deadline-free  $1 | \sum w_i U_i$  problem.

problem structure can greatly boost performances. These results emphasize the need of research on novel ILP formulations for other more complex scheduling models.

## References

- [1] E.L. Lawler (1983), Scheduling a single machine to minimize the number of late jobs Report CSD-83-139, EECS Department, University of California, Berkeley. Available from <http://techreports.lib.berkeley.edu>.
- [2] R.M. Karp (1972), Reducibility among Combinatorial Problems, in *Complexity of Computations*, R.E. Miller and J.W. Thatcher (Eds.), Plenum Press, New York, 85 – 103.
- [3] J.M. Moore (1968), An  $n$  job, one machine sequencing algorithm for minimizing the number of late jobs, *Management Science* **15**, 102 – 109.
- [4] J.K. Lenstra and A.H.G. Rinnooy Kan and P. Brucker (1977), Complexity of machine scheduling problems, *Annals of Discrete Mathematics* **1**, 343 – 362.
- [5] C.N. Potts and L.M. Van Wassenhove (1988), Algorithms for scheduling a single machine to minimize the weighted number of late jobs, *Management Science* **34**, 7, 843 – 858.
- [6] A.M.A. Hariri and C.N. Potts (1994), Single machine scheduling with deadlines to minimize the weighted number of tardy jobs *Management Science* **40**, 12, 1712 – 1719.
- [7] Ph. Baptiste, C. Le Pape and L. Péridy (1998), Global Constraints for Partial CSPs: A Case-Study of Resource and Due Date Constraint *LNCS (Proc. of CP)* **1520**, 87 – 101.
- [8] R. M’Hallah and R.L. Bulfin (2003), Minimizing the weighted number of tardy jobs on a single machine, *European Journal of Operational Research* **145**, 1, 45 – 56.

- [9] S. Dauzère-Pérès and M. Sevaux (2003), Using Lagrangean Relaxation to Minimize the Weighted Number of Late Jobs on a Single Machine, *Naval Research Logistics* **50**, 273 – 288.
- [10] S. Dauzère-Pérès and M. Sevaux (2004), An exact method to minimize the number of tardy jobs in single machine scheduling, *Journal of Scheduling* **7**, 405 – 420.
- [11] R. M’Hallah and R.L. Bulfin (2007), Minimizing the weighted number of tardy jobs on a single machine with release dates, *European Journal of Operational Research* **176**, 727 – 744.
- [12] J.T. Linderoth and M.W.P. Savelsbergh (1999), A computational study of search strategies for mixed integer programming, *INFORMS Journal on Computing* **11**, 2, 173 – 187.
- [13] V. T’kindt and F. Della Croce and J.-L. Bouquard (2007), J.-L. Enumeration of Pareto optima for a flowshop scheduling problem with two criteria, *INFORMS Journal on Computing* **19**, 1, 64 – 72.