J Sched (2010) 13: 39–47
DOI 10.1007/s10951-008-0092-6

Sequencing a single machine with due dates and deadlines: an ILP-based approach to solve very large instances

P. Baptiste · F. Della Croce · A. Grosso · V. T'kindt

Received: 24 October 2007 / Accepted: 19 September 2008 / Published online: 6 November 2008
© Springer Science+Business Media, LLC 2008

Abstract 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, and up to 50,000 jobs in size for the special deadline-free case.

Keywords Sequencing and scheduling · Single machine · Tardy jobs · Deadlines

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 (Graham et al. 1979), the problem is denoted $1|\bar{d}_i|\sum w_i U_i$: a job set $N = \{1, 2, \ldots, n\}$ 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 σ^* 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. We also make the common assumption $d_i \leq \bar{d}_i$ for all $i \in N$.

From the 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 $i \in N$ (Lawler 1983), or if there are no deadlines (Karp 1972), 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 (Moore 1968). If the release dates are present, the $1|r_i|\sum U_i$ problem is already NP-hard in the strong sense (Lenstra et al. 1977). Very little work has appeared in the literature for the $1|\bar{d}_i|\sum w_i U_i$ problem, the state-of-the-art exact algorithm being a branch and bound presented in Hariri and Potts (1994) able to solve instances with up to 300 jobs.

A richer literature is available for the deadline-free variant: Potts and Van Wassenhove (1988) gave a branch and bound algorithm for solving instances with up to 1,000 jobs, while M'Hallah and Bulfin (2003) 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 Dauzère-Pérès and Sevaux 2003, 2004; M'Hallah and Bulfin 2007). However, the presence of release dates completely changes the structure of the problem.

Notice that the considered model is of interest not only to the scheduling community, but also to the wider OR community as it generalizes the $1|d_i = d|\sum w_i U_i$ problem that corresponds to the well known 0/1 Knapsack problem. Hariri and Potts (1994) cite some interesting applications of the $1|\bar{d}_i|\sum w_i U_i$ model, like crop harvesting in agriculture.

P. Baptiste
CNRS LIX, Ecole Polytechnique, Paris, France
e-mail: philippe.baptiste@polytechnique.fr

F. Della Croce
D.A.I., Politecnico di Torino, Torino, Italy
e-mail: federico.dellacroce@polito.it

A. Grosso (✉)
D.I., Universitá di Torino, Torino, Italy
e-mail: grosso@di.unito.it

V. T'kindt
Laboratory of Computer Science, University of Tours, Tours, France
e-mail: tkindt@univ-tours.fr

We note that most of the works on this scheduling problem explicitly avoid to manage the LP relaxation by standard LP tools: in Potts and Van Wassenhove (1988) an ILP model is presented, but a specific dynamic programming algorithm is used for solving the relaxation; in Hariri and Potts (1994) no LP is used, the bound being obtained by dynamic programming and state–space relaxation; even in more recent papers, like M'Hallah and Bulfin (2003), 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 for the $1|\bar{d}_i|\sum w_iU_i$ problem. The algorithm, tested on randomly generated instances as proposed by Hariri and Potts (1994), is able to solve to optimality all randomly generated instances with up to 30,000 jobs (within 275 seconds on average) for the $1|\bar{d}_i|\sum w_iU_i$ problem, and up to 50,000 jobs (within 316 seconds on average) for the special deadline-free case $1||\sum w_iU_i$. Accordingly with the existing literature—particularly with Potts and Van Wassenhove (1988) for the deadline-free case—we further study the performance of the algorithm on considerably harder random instances with *correlated* data; the proposed algorithm suffers a limited performance degradation for the so-called weakly correlated instances, still being able to solve a whole 10,000 jobs batch within 191 seconds on average. Other instances with *strongly* correlated data are apparently the hardest ones, and probably a different approach is needed to efficiently handle such instances for large n.

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 $C_i \le D_i$ 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 $B_t = \{i \in N : \bar{d}_i \le t\}$, $A_t = \{i \in N : d_i > t\}$, and let $T = (t_1, t_2, \ldots, t_m)$ 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.

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

subject to

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

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

With modern ILP solvers and hardware, the 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

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

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

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

The following result generalizes the *reduction theorem* presented in Potts and Van Wassenhove (1988) 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.*

Proof Recall that E (respectively, E') defines the maximum completion dates $D_i = d_i$ or $\bar{d}_i$ (resp., $D'_i = d'_i$ or $\bar{d}'_i$) for all jobs $i \in N$ (resp., $i \in N'$).

In the following, let $C_1, \ldots, C_n$ and $C'_1, \ldots, C'_n$ be the completion times of the jobs in the feasible sequence for the original and reduced problem, respectively.

($\Longrightarrow$) Let $\sigma j \omega$ be a feasible sequence and, without loss of generality, assume $D_i \le D_j < D_k$, for all $i \in \sigma$, $k \in \omega$. We prove that $\sigma\omega$ is feasible for the reduced problem.

If $i \in \sigma$, $C_i \le D_i$ for feasibility. Also, since i precedes j and $C_j \le D_j$, $C_i \le C_j - p_j \le D_j - p_j$. Hence, $C'_i = C_i \le \min\{D_i, D_j - p_j\} = D'_i$. If $k \in \omega$ then $C'_k = C_k - p_j \le D_k - p_j = D'_k$.

Hence, $\sigma\omega$ satisfies $C'_i \le D'_i$ for all $i \in N'$.

($\Longleftarrow$) Let $\sigma\omega$ be a feasible sequence for the reduced problem. Without loss of generality, assume $D'_i \le D_j - p_j < D'_k$ for all $i \in \sigma$, $k \in \omega$—note this implies also $D'_k = D_k - p_j$ for all $k \in \omega$. We prove that $\sigma j \omega$ is feasible for the original problem.

For all $i \in \sigma$, $C_i = C'_i \le D'_i \le D_i$. For all $k \in \omega$, $C_k = C'_k + p_j \le D'_k + p_j = D_k$.

For job j, let i be the last job of σ, and note that $C_i \le D'_i \le D_j - p_j$, thus $C_j = C_i + p_j \le D_j$.

Hence, $\sigma j \omega$ satisfies $C_i \le D_i$ for all $i \in N$. □

Property 1 allows removing job j from consideration and solving 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

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

We formulate the following flow problem on G:

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

subject to

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

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

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

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

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

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

The shape of G is sketched in Fig. 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 the LP relaxation of (1)–(3).

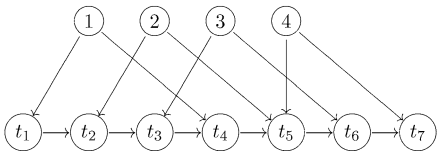
Proof In any feasible flow on G, the ingoing flow in any node $t_k \in T$ is, by flow conservation:

$$\begin{aligned} y_{t_k,t_{k+1}} &= \sum_{i\,:\,\bar{d}_i \le t_k} y_{i,\bar{d}_i} + \sum_{i\,:\,d_i \le t_k} y_{i,d_i} \\ &= \sum_{i\,:\,\bar{d}_i \le t_k} (y_{i,\bar{d}_i} + y_{i,d_i}) + \sum_{i\,:\,d_i \le t_k, \bar{d}_i > t_k} y_{i,d_i} \\ &= \sum_{i\in B_{t_k}} p_i + \sum_{N\setminus(B_{t_k}\cup A_{t_k})} y_{i,d_i} \\ &= \sum_{i\in B_{t_k}} p_i + \sum_{N\setminus(B_{t_k}\cup A_{t_k})} p_i x_i. \end{aligned}$$

The capacity constraints $y_{t_k,t_{k+1}} \le t_k$ and the sink balance correspond one-to-one to constraints (2).

For the objective function, we have $z = \sum_{(i,d_i)\in A_1} \frac{w_i}{p_i} y_{i,d_i} = \sum_{i\in N} w_i x_i$. □

Property 2 establishes the equivalence between model (7)–(13) and the LP relaxation of (1)–(3). In our experiments we kept solving the flow model (7)–(13) that requires $\mathcal{O}(n)$ space instead of $\mathcal{O}(n^2)$, thus overcoming the memory problems experienced with formulation (1)–(3) on large instances. For example, on a 4000-job instance with over 13 million nonzeros for model (1)–(3), the flow model requires only 42548 nonzeros.



$T = (t_1, \ldots, t_7) = (d_1, d_2, d_3, \bar{d}_1, \bar{d}_2 = d_4, \bar{d}_3, \bar{d}_4)$
Node balances b_i's are as follows:
$b_1 = p_1$,
$b_2 = p_2$
$b_3 = p_3$
$b_4 = p_4$
$b_{t_1}, \ldots, b_{t_6} = 0$
$b_{t_7} = -(p_1 + p_2 + p_3 + p_4)$

Fig. 1 Graph $G(N \cup T, A)$ for a four-job example

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 model (1)–(3) applied to a *core problem*.

To this extent we introduce the following dominance property.

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.*

Proof (By interchange argument) Suppose by contradiction that an optimal sequence $S = \sigma j \pi i \rho$ with j early preceding i tardy exists where σ, π and ρ are possibly empty subsequences of jobs. We show that the sequence $S' = \sigma i \pi j \rho$ has cost function value $f(S') \leq f(S)$ and is feasible. For the feasibility, as $p_j \geq p_i$, we have that job j and all jobs in σ and π complete not later in S' than in S. Further, all jobs in ρ keep the same completion time they have in S. Hence, all these jobs do not violate their deadline in S' as in S. Finally, job j completes in S' at time $C_j(S') = C_i(S)$. But then we have $C_j(S') = C_i(S) \leq \bar{d}_i \leq \bar{d}_j$, namely, job j completion time does not violate the deadline $\bar{d}_j$. For the optimality, the cost function value contribution of the jobs in σ and ρ for S' is lower than or equal to the corresponding cost function value contribution in S as they all do not increase their completion times. With respect to the cost function value contribution of jobs i and j, it is equal to w_i in S as job i is tardy and job j is early, while it reduces to at most w_j (assuming j tardy) as we have $C_i(S') \leq C_j(S) \leq d_j \leq d_i$, namely, job i is early in S'. □

Let $X^* = [x_1^*, \ldots, x_n^*]$ be the optimal solution of the linear programming continuous relaxation of the problem. 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^*,

that is, by means of the following procedure:

CoreProblem($x_1^*, \ldots, x_n^*$ **: solution of LP relaxation**)
1: Let $C := \emptyset$;
2: **for all** $j \in N$ **do**
3: **if** $(0 < x_j^* < 1)$
or ($x_j^* = 0$ and $\nexists i : x_i^* = 0$ dominating j according to Property 3)
or ($x_j^* = 1$ and $\nexists i : x_i^* = 1$ dominated by j according to Property 3) **then**
4: $C := C \cup j$;
5: **else**
6: Set j early if $x_j^* = 1$ and set j tardy if $x_j^* = 0$;
7: **end if**
8: **end for**
9: **return** C.

A first heuristic solution is then obtained by applying model (1)–(3) to the core jobset C. The size of C has in all tests been always less than 5% of the original problem size, and this core problem is therefore solvable to optimality in a very short time by the ILP solver. In order to further improve the quality of the heuristic solution, we then perform a local search phase that uses the optimal solution of the core problem as initial solution. Two feasible solutions $(\bar{x}_1, \ldots, \bar{x}_n)$ and $(\tilde{x}_1, \ldots, \tilde{x}_n)$ are neighbors 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 neighborhood 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 $\nexists i : x_i^* = 0$ ($x_i^* = 1$) dominating j (dominated by j) according to Property 3. In this way, in practice, always much less than 1% of the neighborhood is explored. 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

$$z = z^* + \sum_{x_i \notin B^*} \bar{c}_i x_i,$$

$$x_j = x_j^* + \sum_{x_i \notin B^*} \alpha_{ji} x_i \quad (x_j \in B^*),$$

$$x_i \geq 0, \quad i \in N,$$

be the corresponding reformulation with $\bar{c}_i$ being the reduced cost of variable x_i, and α_{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, Linderoth and Savelsbergh 1999):

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

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

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 Baptiste et al. 1998; T'kindt et al. 2007 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, otherwise 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 the 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 growth of the constraint matrix size as n increases. We then implemented a depth-first enumeration scheme which incorporates the reduction procedure sketched in Sect. 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 the 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 Hariri and Potts (1994), we considered ten classes of instances structured as follows:

- The values for processing times $p_1, \ldots, p_n$ and weights $w_1, \ldots, w_n$ are integers drawn randomly from the uniform distribution $[1, 100]$. We note that in Hariri and Potts (1994) the range was $[1, 10]$.
- The due dates $d_1, \ldots, d_n$ 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 $\bar{d}_1, \ldots, \bar{d}_n$ are integers drawn for each job i from the uniform distribution $[d_i, 1.1P]$.

Unfeasible examples (that can occur but are trivially detected whenever the earliest deadline sequence violates at least one deadline) were discarded by the generation scheme. We generated 20 feasible examples for each u, v class—thus creating batches of 200 examples—with size n ranging from 1000 to 30,000.

All the testing ran on a Pentium IV PC with 3 GHz clock and 1 Gb memory. The ILP solver used is XPRESS-MP 18.1 by Dash Optimization.

4.2 Results

As previously remarked, the 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 jobs instance with distribution $u = 0.1$, $v = 0.5$ that required 3982 seconds. The results are summarized in Table 2.

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 $\frac{\text{UB}-\text{OPT}}{\text{OPT}}$ and $\frac{\text{OPT}-\text{LB}}{\text{OPT}}$ 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.

Indeed, as indicated in Table 3 dedicated to the 30,000-job instances, the high quality of both upper and lower bounds allows strongly decreasing the problem size by means of the reduction procedure already at the root node (see the related "Red. probl. size" column). From Table 3, we evince also that the hardest distributions are those with $u = 0.1$ and $v = 0.3, 0.5$.

Table 1 Performances of the model (1)–(3)

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

Table 2 Performances of the enumerative algorithm

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 3 Enumerative algorithm: details of performances for the case $n = 30{,}000$

u	v	Red. probl. size		CPU Time (s)		NODES		UB-GAP%		LB-GAP%	
		avg	max	avg	max	avg	max	avg	max	avg	max
0.1	0.3	3486.2	10406	420.3	1644.5	6.3	81	0.001	0.002	$< 10^{-3}$	0.001
0.1	0.5	4974.4	10883	581.2	3491.1	6.3	47	0.001	0.001	$< 10^{-3}$	0.001
0.1	0.7	2620.6	5643	262.7	416.8	1.0	1	$< 10^{-3}$	0.001	$< 10^{-3}$	0.001
0.1	0.9	876.7	4452	225.9	272.8	1.0	1	$< 10^{-3}$	$< 10^{-3}$	$< 10^{-3}$	$< 10^{-3}$
0.3	0.5	1596.8	10047	263.0	335.8	1.1	3	$< 10^{-3}$	0.001	$< 10^{-3}$	0.001
0.3	0.7	1352.8	7538	239.3	278.9	1.0	1	$< 10^{-3}$	0.001	$< 10^{-3}$	0.001
0.3	0.9	254.6	654	205.4	276.8	1.0	1	$< 10^{-3}$	$< 10^{-3}$	$< 10^{-3}$	$< 10^{-3}$
0.5	0.7	163.8	373	203.3	222.8	1.0	1	$< 10^{-3}$	$< 10^{-3}$	$< 10^{-3}$	$< 10^{-3}$
0.5	0.9	310.4	500	181.9	214.9	1.0	1	$< 10^{-3}$	$< 10^{-3}$	$< 10^{-3}$	$< 10^{-3}$
0.7	0.9	233.6	546	165.1	185.7	1.0	1	$< 10^{-3}$	$< 10^{-3}$	$< 10^{-3}$	$< 10^{-3}$

Additionally, we performed some tests on instances with a larger $[1, 10000]$ distribution for p_i's and w_i's. Apparently, the algorithm is not sensitive to such enlargement; for example, a whole batch of 10,000 jobs instances was handled within 321 seconds in the worst case (average 45), with all instances solved at 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 4.

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 M'Hallah and Bulfin 2003).

Correlated instances

In Potts and Van Wassenhove (1988), also *correlated* instances (linking processing times and weights) are studied for the deadline-free problem. Such instances are likely to be harder than purely random instances; random *weakly correlated* instances are generated by drawing each w_i from the uniform distribution $[p_i, p_i + 20]$, while *strongly correlated* instances are obtained by setting $w_i = p_i + 20$ for all $i \in N$. We note that no correlated instances were considered in Hariri and Potts (1994).

In Table 5 we report the computational experience with $n = 10{,}000$ for the weakly correlated instances of the $1|\bar{d}_i|\sum w_i U_i$ problem; such instances are considerably harder than the uncorrelated ones, but not dramatically harder for our algorithm. With respect to the corresponding uncorrelated instances, we get a reduced problem size at the root node that is larger by a factor of approximately 2.3 (2.6 for the $u = 0.1, v = 0.3$ class). The average CPU times are inflated by a factor of 6 (but 32 for $u = 0.1, v = 0.3$). The worst figures are due to the fact that the reduction devices

Table 4 Performances of the enumerative algorithm on instances of the deadline-free $1||\sum w_i U_i$ problem

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 5 Weakly correlated instances, $n = 10{,}000$

u	v	Red. probl. size		CPU Time (s)		Nodes		UB-GAP%		LB-GAP%	
		avg	max	avg	max	avg	max	avg	max	avg	max
0.1	0.3	3460.9	9027	1077.3	10094.7	35.4	293	0.002	0.009	0.001	0.005
0.1	0.5	2949.9	7689	501.2	3775.9	10.2	137	0.001	0.002	0.001	0.004
0.1	0.7	1685.0	4935	95.5	276.3	1.0	1	$< 10^{-3}$	0.002	$< 10^{-3}$	0.001
0.1	0.9	495.1	1182	41.1	76.6	1.0	1	$< 10^{-3}$	$< 10^{-3}$	$< 10^{-3}$	$< 10^{-3}$
0.3	0.5	1106.4	5335	47.3	99.3	1.0	1	$< 10^{-3}$	0.002	$< 10^{-3}$	$< 10^{-3}$
0.3	0.7	926.1	7120	41.4	70.1	1.1	3	$< 10^{-3}$	0.003	$< 10^{-3}$	$< 10^{-3}$
0.3	0.9	306.1	627	32.2	38.0	1.0	1	$< 10^{-3}$	$< 10^{-3}$	$< 10^{-3}$	$< 10^{-3}$
0.5	0.7	608.9	2575	31.9	37.8	1.0	1	$< 10^{-3}$	0.001	$< 10^{-3}$	$< 10^{-3}$
0.5	0.9	355.1	717	27.9	32.3	1.0	1	$< 10^{-3}$	$< 10^{-3}$	$< 10^{-3}$	$< 10^{-3}$
0.7	0.9	407.6	822	23.3	25.0	1.0	1	$< 10^{-3}$	$< 10^{-3}$	$< 10^{-3}$	$< 10^{-3}$
Overall		1230.1	9027	191.9	10094.7	5.4	293	$< 10^{-3}$	0.009	$< 10^{-3}$	0.005

embedded in the algorithm are less effective with correlated data. Interestingly enough, the 10,094 seconds instance in class $(0.1, 0.3)$ does not produce the largest 9,027 jobs reduced problem at the root. Instead it gives a smaller—but quite difficult—problem with 7,719 jobs, that is solved in 233 nodes, whereas the 9,027 jobs reduced problem only takes 26 nodes and 867 seconds. The second-worst time in the batch is 5,254 seconds.

The strongly correlated case is by far the hardest one. Here the dominance conditions no longer hold and all reduction devices are apparently ineffective even though the gap between UB and LB remains very limited (less than 0.2% on average): the enumerative algorithm basically exhibits the performances of the underlying ILP solver and is just able to solve the whole $n = 100$ batch within a few seconds on average, but already with a bottleneck instance requiring approximately 60 seconds. For the $n = 200$ batch, only 191 out of the 200 considered instances could be solved within a time limit of 3600 seconds per instance. In order to assess the rationale of such behavior, we worked on some of the unsolved instances, and came up with a family of hard examples having two due dates only and no deadlines; the (1)–(3) model for such examples is thus a particular 2-constraint knapsack problem. In the appendix, we provide data for one of such examples that turned out to be (with others of the same family), in our experience, extremely difficult: all the solvers we tried were not able to solve it to optimality within 3600 seconds. The solvers were XPRESS-MP 18.10, CPLEX 10.11, and the $2 - KP$ exact algorithm *MT* of Martello and Toth (2003) (applied on an equivalent instance without zeros in the constraints matrix to meet the input assumptions of MT). This suggests that the presence of correlated processing times and weights is already sufficient enough to make our problem very hard, similarly to any (multi)knapsack-like problem with strong correlation between items sizes and profits. Particularly, even if the upper bound quality does not break down (less than 0.26% from optimum for the instance in the appendix) the UB-LB gap narrows very slowly.

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 problem structure can greatly boost performances. The special strongly-correlated instances apparently stand as the real challenge for future research on the considered problem.

Appendix

We provide data for a 200-jobs 2-due dates difficult instance of $1||\sum w_i U_i$. Note that p_i and w_i are strongly correlated ($w_i = p_i + 20$), and that the due dates $d_1, \ldots, d_{100}$ and $d_{101}, \ldots, d_{200}$ are set approximately to $1/4$ and $1/2$ of $\sum_i p_i$.

$n = 200$

$w_i = p_i + 20, \quad i = 1, \ldots, 200.$

$p_1, \ldots, p_{200} =$ 72 48 6 28 24 85 50 42 93 16 31 39 82 100 94 49 13 95 49 40 81 17 10 73 31 83 41 73 33 13 93 55 6 88 27 43 56 48 33 31 84 6 70 11 69 100 43 85 68 14 52 100 65 66 44 49 96 77 87 2 43 52 34 26 44 85 94 69 36 73 75 65 1 44 38 47 93 59 43 44 65 17 44 78 34 54 53 80 74 4 68 78 90 13 35 54 27 41 28 17 14 11 89 29 52 13 83 49 27 2 76 97 96 13 79 54 29 80 76 87 61 63 72 76 13 86 44 52 93 37 6 63 77 30 45 30 52 20 15 56 25 12 78 81 93 70 52 40 68 2 48 80 47 52 11 18 1 96 61 3 49 69 75 70 47 96 62 57 46 75 72 87 28 81 100 8 29 66 4 74 75 29 92 2 97 82 93 72 63 51 49 69 74 47 42 62 52 89 91 62

$d_1, \ldots, d_{100} = 2547, \qquad d_{101}, \ldots, d_{200} = 5094.$

References

Baptiste, Ph., Le Pape, C., & Péridy, L. (1998). Global constraints for partial CSPs: a case-study of resource and due date constraint. In *LNCS (Proc. of CP)* (Vol. 1520, pp. 87–101).

Dauzère-Pérès, S., & Sevaux, M. (2003). Using Lagrangean relaxation to minimize the weighted number of late jobs on a single machine. *Naval Research Logistics*, *50*, 273–288.

Dauzère-Pérès, S., & Sevaux, M. (2004). An exact method to minimize the number of tardy jobs in single machine scheduling. *Journal of Scheduling*, *7*, 405–420.

Graham, R. L., Lawler, E. L., Lenstra, J. K., & Rinnooy Kan, A. H. G. (1979). Optimization and approximation in deterministic sequencing and scheduling: a survey. *Annuals of Discrete Mathematics*, *5*, 287–326.

Hariri, A. M. A., & Potts, C. N. (1994). Single machine scheduling with deadlines to minimize the weighted number of tardy jobs. *Management Science*, *40*(12), 1712–1719.

Karp, R. M. (1972). Reducibility among combinatorial problems. In R. E. Miller & J. W. Thatcher (Eds.), *Complexity of Computations* (pp. 85–103). New York: Plenum.

Lawler, E. L. (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.

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

Linderoth, J. T., & Savelsbergh, M. W. P. (1999). A computational study of search strategies for mixed integer programming. *INFORMS Journal on Computing*, *11*(2), 173–187.

Martello, S., & Toth, P. (2003). An exact algorithm for the two-constraint 0–1 knapsack problem. *Operations Research*, *51*, 826–835.

M'Hallah, R., & Bulfin, R. L. (2003). Minimizing the weighted number of tardy jobs on a single machine. *European Journal of Operational Research*, *145*(1), 45–56.

M'Hallah, R., & Bulfin, R. L. (2007). Minimizing the weighted number of tardy jobs on a single machine with release dates. *European Journal of Operational Research*, *176*, 727–744.

Moore, J. M. (1968). An *n* job, one machine sequencing algorithm for minimizing the number of late jobs. *Management Science*, *15*, 102–109.

Potts, C. N., & Van Wassenhove, L. M. (1988). Algorithms for scheduling a single machine to minimize the weighted number of late jobs. *Management Science*, *34*(7), 843–858.

T'kindt, V., Della Croce, F., & Bouquard, J.-L. (2007). Enumeration of Pareto optima for a flowshop scheduling problem with two criteria. *INFORMS Journal on Computing*, *19*(1), 64–72.