MODELS FOR MINIMIZING THE WEIGHTED NUMBER OF LATE JOBS WITH DETERMINISTIC MACHINE AVAILABILITY CONSTRAINTS

Boris Detienne

▶ **To cite this version:**

Boris Detienne. MODELS FOR MINIMIZING THE WEIGHTED NUMBER OF LATE JOBS WITH DETERMINISTIC MACHINE AVAILABILITY CONSTRAINTS. 9th International Conference on Modeling, Optimization & SIMulation, Jun 2012, Bordeaux, France. hal-00728606

HAL Id: hal-00728606

https://hal.science/hal-00728606v1

Submitted on 30 Aug 2012

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

9^{th} *International Conference of Modeling, Optimization and Simulation - MOSIM'12*
June 6-8, 2012 - Bordeaux - France
"Performance, interoperability and safety for sustainable development"

MODELS FOR MINIMIZING THE WEIGHTED NUMBER OF LATE JOBS WITH DETERMINISTIC MACHINE AVAILABILITY CONSTRAINTS

Boris Detienne

LIA / Université d'Avignon et des Pays de Vaucluse
339, chemin des Meinajaries
84911 Avignon cedex 9 - France
boris.detienne@univ-avignon.fr

ABSTRACT: *We study the problem of minimizing the weighted number of tardy jobs on a single machine with deterministic machine availability constraints. We show that the cases with resumable and non-resumable jobs ($1, h_k|r_i, nr - a|\sum w_iU_i$ and $1, h_k|r_i, r - a|\sum w_iU_i$) can both be modeled as a more general scheduling problem, consisting in selecting and scheduling jobs on parallel machines subject to time windows and group constraints. For this generic problem, we design a Mixed Integer Linear Programming (MILP) model as well as several improvements. Based on a set of randomly generated instances, computational results indicate that the direct solution of the improved MILP is a competitive method since it allows solving optimally most instances of $1, h_k|r_i, nr - a|\sum w_iU_i$, $1, h_k|r_i, r - a|\sum w_iU_i$ and $1|r_i|\sum w_iU_i$ with 300 jobs in 1000 seconds. Moreover, it can be favorably compared to previous work published for the special case of periodic maintenance without release date.*

KEYWORDS: *Scheduling, Late jobs, Availability constraints, Integer Linear Programming.*

1 INTRODUCTION

In most scheduling problems, we assume that a job can be processed anytime after its release date. However, more and more research papers study scheduling problems subject to machine or job availability constraints. In this context, machines or jobs can be unavailable for processing during given time intervals. This reflects real constraints that cannot be ignored in many practical situations like, in real production scheduling, maintenance operations or pauses imposed by labor laws.

We study two problems defined by a set $I = \{J_1, \ldots, J_n\}$ of n jobs. Each job J_i is characterized by a release date r_i, a due date d_i, a processing time p_i and a weight w_i. All jobs must be processed on a single machine, so that the weighted number of late jobs is minimized. A job cannot be preempted by another one: if a job starts on the machine, no other job can start until the first one completes. A set of K time intervals $[B_s, F_s]$, $s \in \{1, \ldots, K\}$, when the machine is unavailable, is given as input. Moreover, we add without loss of generality two fictitious unavailability periods 0 and $K + 1$, such that $B_0 = F_0 = 0$ and $B_{K+1} = F_{K+1} = \infty$. All data are integer and deterministic.
We consider the cases of resumable and non-resumable jobs. When jobs are resumable, a job may start before a unavailability period and be resumed after it. In this case, the completion of the job is simply postponed for the duration of the unavailability periods crossed by the job. This problem is denoted by $1, h_k|r_i, r - a|\sum w_iU_i$ in the standard three fields notation (Graham et al., 1979). When jobs are non-resumable, if a job has started but has not completed before an unavailability period, it must be restarted from zero after the unavailability period. Thus, it is equivalent to forbid jobs to be preempted by unavailability periods. This problem is denoted by $1, h_k|r_i, nr - a|\sum w_iU_i$. Although this is not the object of this work, let us note that some research papers also investigate the semi-resumable case, where job can be resumed but must be partially restarted after an availability period (this corresponds in practice to, for example, a setup time required after a maintenance). Some authors (e.g. Mauguière et al., 2005) also consider problems with both resumable and non-resumable jobs, and crossable and non-crossable unavailability periods (even resumable jobs cannot be preempted by non-crossable periods).

Concerning only the case of deterministic machine availability constraints (when unavailability periods are perfectly known in advance), (Ma et al., 2010) gathers more than 90 research papers. Interestingly,

only one paper mentioning the minimization of the number of tardy jobs is referenced in this survey: (Lee, 1996) shows that $1, h_1|r - a|\sum U_i$ is polynomial and that Moore-Hodgson's algorithm (Moore, 1968) lead to an absolute error of 1 to solve the problem $1, h_1|nr - a|\sum w_i U_i$. Another survey (Schmidt, 2000) cites (Lawler and Martel, 1989), which studies the minimization of the number of late jobs on two machines whose speeds can vary (and possibly be null) over time, when preemption is allowed. (Chen, 2009) studies a special case of $1, h_k|nr - a|\sum U_i$ in which unavailability intervals correspond to periodic maintenance. The author develop a heuristic and a branch-and-bound to solve optimally instances with up to 32 jobs.

The aim of this paper is to provide models for solving $1, h_k|r_j, nr-a|\sum w_i U_i$ and $1, h_k|r_i, r-a|\sum w_i U_i$. The idea of these models is to convert the problems into a more general problem, for which we write an efficient Mixed Integer Linear Programming (MILP) model. The paper is structured as follows. Section 2 describes a general scheduling problem ($STWP$) which consists in selecting and scheduling jobs subject to time windows and group constraints on parallel machines. Several improvements are proposed to make the solution of the MILP by a solver easier. Section 3 investigate the case of resumable jobs. We show that the problem is equivalent to $1|r_i|\sum w_i U_i$ and present a way of modeling the latter as ($STWP$). Section 4 gives two different transformations from $1, h_k|r_i, nr-a|\sum w_i U_i$ to ($STWP$). In Section 5, we report computational results obtained for the direct solution of the improved MILP. Finally, we conclude in Section 6.

2 A PROBLEM OF SELECTING AND SCHEDULING JOBS ($STWP$)

This section describes the problem of Selecting jobs with Time Windows on Parallel processors ($STWP$) and provides an efficient Mixed Integer Linear Programming to solve it.

2.1 Definition

An instance of ($STWP$) is defined by the following data:

- A set $I = \{J_1, \ldots, J_{n_I}\}$ of n_I jobs;
- A partition G of I into n_G disjoint groups: $G = \{G_1, \ldots, G_{n_G}\}$;
- A set $M = \{M_1, \ldots, M_{n_M}\}$ of n_M machines;
- For each job $J_i \in I$, the following integers are given: a release date r_i, a deadline $\bar{d}_i$, a processing cost w_i, a processing time p_i, and a processing machine m_i

A feasible solution of this instance of ($STWP$) satisfies the following constraints. For each group G_g, $g \in \{1, \ldots, n_G\}$, exactly one job in G_g must be processed. Processing job J_i, $i \in \{1, \ldots, n_J\}$ generates a cost w_i. If it is selected, job J_i must be processed on machine M_{m_i} without preemption, within its processing time window $[r_i, \bar{d}_i]$.

This problem is clearly a generalization of the problem of minimizing of the weighted number of tardy jobs on parallel unrelated machines ($R|r_i|\sum w_i U_i$).

2.2 Characterization of feasible selections of jobs

The MILP characterization of feasible selections of jobs presented in this section is derived from (Detienne et al., 2011), where the authors present MILP models for minimizing a regular step cost function of jobs completion times on parallel machines. The same principle was also used in (Dauzère-Pérès and Sevaux, 2002) for minimizing the number of late jobs on a single machine. The current work differs by focusing on the feasibility of a selection of jobs, while the two others focus on the optimality of a sequence of jobs and lead to slightly different MILP models. Moreover, we propose original improvements for the derived model. Finally, in current work we address a more general problem containing problems of both previous papers.

Let us focus on a single machine, the result being easily adapted to multiple machines. The core idea can be seen as an extension of the Earliest Due Date rule. Suppose that all release dates are null, and consider a set $J \subseteq I$ of jobs that are selected for processing, such that all constraints on groups are satisfied. Then all jobs $J_i \in J$ must be scheduled in $[0, \bar{d}_i]$. We know that it is possible (the selection is feasible) if and only if, when jobs are sequenced with no idle time and in a non decreasing order of their deadlines, they all meet their deadline (Jackson, 1955). However, this result does not hold in the presence of non equal release dates. In this case, we show that, by inserting virtual jobs with appropriate time windows, we can establish a similar dominant order.

For the sake of conciseness, we do not give a formal proof of the result. This proof would be similar to the one described in (Detienne et al., 2011). Let us intuitively show how we build the set of virtual jobs. Let us consider a feasible selection and schedule of jobs. Let J_i and J_j be two jobs processed consecutively in this solution (J_i precedes J_j). Concerning the positions of J_i and J_j with respect to their release dates and deadlines, the three following cases can occur.

- Case 1: $\bar{d}_i \leq \bar{d}_j$. Jobs J_i and J_j are scheduled according to a non-decreasing order of their dead-

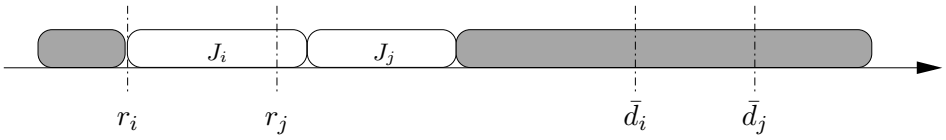

Figure 1: Case 1: $\bar{d}_i \leq \bar{d}_j$. Jobs i and j are scheduled according to a non-decreasing order of their deadlines.

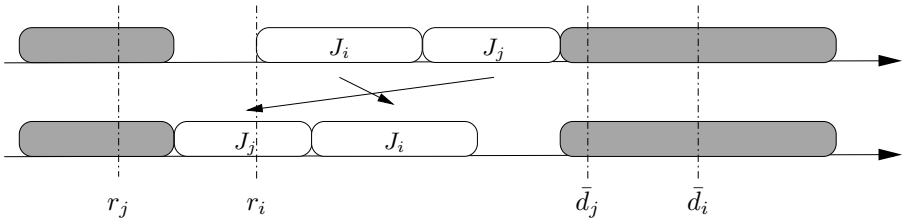

Figure 2: Case 2: $r_i \geq r_j$ and $\bar{d}_i > \bar{d}_j$. Jobs i and j can be swapped and scheduled according to a non-decreasing order of their deadlines, and the resulting schedule is feasible.

lines (see Figure 1).

- Case 2: $r_i \geq r_j$ and $\bar{d}_i > \bar{d}_j$. It can be seen from Figure 2 that J_i and J_j can be swapped, so that the schedule before and after them does not change and J_i and J_j still meet their deadlines. Moreover, in the resulting schedule, J_i and J_j are scheduled according to a non-decreasing order of their deadlines.
- Case 3: $r_i < r_j$ and $\bar{d}_i > \bar{d}_j$. Jobs J_i and J_j are not scheduled according to a non-decreasing order of their deadlines (see Figure 3). Moreover, interchanging their processing order may result in delaying the jobs processed after J_i and J_j, or simply the completion time of job J_i exceeding its deadline. We propose to insert a virtual job $J_{h(i,j)}$ in the group of job J_i, so that the current feasible solution can be obtained by selecting $J_{h(i,j)}$, equivalent to selecting J_i, and schedule it before J_j while keeping a sequence of jobs satisfying a non-decreasing order of their deadlines. We can show that choosing the same characteristics for $J_{h(i,j)}$ as for J_i except $\bar{d}_{h(i,j)} = \bar{d}_j$ is valid if, in the order considered, ties are broken according to an arbitrary non-decreasing order of the release dates of the jobs.

Thus, in case 1, the feasible solution is obtained by scheduling J_i and J_j in the order of their deadlines. In case 2, interchanging J_i and J_j leads to another feasible schedule, in which they are scheduled in the order of their deadlines. In case 3, once we have added the adapted set of virtual jobs, the feasible solution can be obtained by selecting the virtual job $J_{h(i,j)}$ corresponding to scheduling J_i before J_j, so that $J_{h(i,j)}$ and J_j are scheduled in a non-decreasing order of their deadlines, ties being broken according to a non-decreasing order of their release dates. The intuitive insight provided above can be applied separately on each machine of the problem, to generate a suitable set of virtual jobs. We use the fact that, one each machine, there exists at least one feasible schedule in which the processed job are scheduled according to this order, to write an MILP model that determines the optimal selection of jobs.

Formally, for each job $J_i \in I$, we define the set of associated virtual jobs by:

$$H_i = \{J_{h(i,j)} / J_j \in I, m_i = m_j, r_i < r_j, \bar{d}_i > \bar{d}_j\}$$

where $h(i,j)$ denotes the index of the virtual job, satisfying $h(i,j) \neq h(i',j')$ if $i \neq i'$ or $j \neq j'$, and $h(i,j) \in \{n_I + 1, \ldots, n_{I'}\}$ with $n_{I'} = |I| + |\cup_{k=1}^{n_I} H_k|$. Moreover, let us denote $h^{-1}(l)$ the index of job J_j such that $\exists i / h(i,j) = l$, i.e. the index of the job that generated J_l. If $J_l \in I$ (J_l is not a virtual job), we set $h^{-1}(l) = \emptyset$. Each job $J_l \in H_i$, $i \in \{1, \ldots, n_I\}$ has the following characteristics: $r_l = r_i$, $\bar{d}_l = \bar{d}_{h^{-1}(l)}$, $p_l = p_i$, $w_l = w_i$ and $m_l = m_i$.

The set of jobs of the enhanced problem is $I' = \{J_1, \ldots, J_{n_{I'}}\}$ and the new groups of jobs are $G' = \{G'_1, \ldots, G'_{n_G}\}$, with $G'_g = G_g \cup \cup_{J_i \in G_g} H_i$, $g \in \{1, \ldots, n_G\}$. Besides, let $I'^m = \{J_i \in I' / m_i = m\}$, $m \in \{1, \ldots, n_M\}$ be the set of jobs that can be processed on machine M_m, and let $G'^m_g = G'_g \cap I'^m$ be the set of jobs from group G'_g that can be processed on machine M_m.

2.3 MILP model

In order to build our Mixed Integer Linear Programming model, let us assume that, for each machine $M_m \in M$, (virtual and real) jobs that can be processed on M_m are sorted according to a non-decreasing order of their deadlines, ties being broken according to an arbitrary non-decreasing order of their release dates. Moreover, let $\sigma^m(k)$ be the index of the k^{th} job that can be processed on M_m. For the sake of readability, let us introduce the following notations:

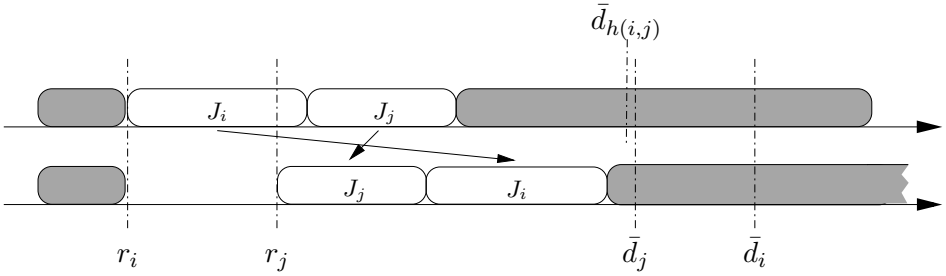

Figure 3: Case 3: $r_i < r_j$ and $\bar{d}_i > \bar{d}_j$. Swapping i and j may lead to an infeasible schedule. So we cannot derive an other optimal schedule satisfying the usual non-decreasing order of the deadlines. A virtual job $J_{h(i,j)}$ is added to the group of job J_i, with the same characteristics as J_i except its deadline $\bar{d}_{h(i,j)} = \bar{d}_j$. Scheduling i before j comes to selecting $J_{h(i,j)}$ and J_j, which can be scheduled in a non-decreasing order of their deadlines.

- $r_k^m = r_{\sigma^m(k)}$ is the release date of the k^{th} job on machine M_m
- $p_k^m = p_{\sigma^m(k)}$ is the processing time of the k^{th} job on machine M_m
- $\bar{d}_k^m = \bar{d}_{\sigma^m(k)}$ is the deadline of the k^{th} job on machine M_m
- $w_k^m = w_{\sigma^m(k)}$ is the processing cost of the k^{th} job on machine M_m

Then, the problem $(STWP)$ can be expressed as:

$$(A) \quad \min \sum_{m=1}^{n_M} \sum_{k=1}^{|I'^m|} w_k^m x_k^m \tag{1}$$

$$\sum_{m=1}^{n_M} \sum_{k/J_k \in G_g'^m} x_k^m = 1 \qquad g = 1, \ldots, n_G \tag{2}$$

$$t_k^m - (r_k^m + p_k^m) x_k^m \geq 0 \qquad m = 1, \ldots, n_M, \; k = 1, \ldots, |I'^m| \tag{3}$$

$$t_k^m - t_{k-1}^m - p_k^m x_k^m \geq 0 \qquad m = 1, \ldots, n_M, \; k = 2, \ldots, |I'^m| \tag{4}$$

$$0 \leq t_k^m \leq \bar{d}_k^m \qquad m = 1, \ldots, n_M, \; k = 1, \ldots, |I'^m| \tag{5}$$

$$x_k^m \in \{0, 1\} \qquad m = 1, \ldots, n_M, \; k = 1, \ldots, |I'^m| \tag{6}$$

In this model, variable t_k^m denotes the completion time of the k^{th} job on machine M_m, if it is processed. Otherwise, it is a lower bound of the starting time of the next processed job. Variable x_k^m is equal to 1 if and only if the k^{th} job on machine M_m is processed, and is equal to 0 otherwise. Constraints (2) ensure that exactly one job of each group is selected. Constraints (3) state that each job cannot start before its release date, while Constraints (4) express the resource and conjunctive constraints (each machine can handle at most one job at a time, and selected jobs are processed in the correct order). Constraints (5) force each job to complete before its deadline. Note that, as jobs are sorted according to a non-decreasing order of their deadlines, Constraint (5) at rank k does not over-constrain the completion times of jobs of ranks k', $k' < k$. Expression (1) is the objective function, which minimizes the sum of the costs of selected jobs.

2.4 Improvements

Proposition 1 and 2 exploit the classical selections on disjunction (see e.g. Carlier, 1982) to remove some virtual jobs and to strengthen the model. Similar propositions are also used in (Dauzère-Pérès and Sevaux, 2002) or (Detienne et al., 2011).

Proposition 1. *Let $J_i \in I$ and $J_j \in I$, $i \neq j$, $m_i = m_j$. If $r_i < r_j$, $d_i > d_j$ and $r_i + p_i + p_j > d_j$, then the virtual job $J_{h(i,j)}$ is useless and can be removed from the model.*

Proof. Processing J_i before J_j is not feasible. Hence, if both J_i and J_j are scheduled, J_j precedes J_i and case 3 (Figure 3) cannot occur. □

Proposition 2. *Let $J_i \in I$ and $J_j \in I$, $i \neq j$, $m_i = m_j = m$. If $r_i + p_i + p_j > d_j$ and $r_j + p_j + p_i > d_i$, then the following inequality holds:*

$$\sum_{k/J_k \in H_i \cup H_j \cup \{J_i, J_j\}} x_k^m \leq 1$$

Proof. The inequality just expresses that J_i and J_j (or one of their corresponding virtual jobs) cannot be both processed in a feasible solution. □

Proposition 3 strengthens the model by removing some equivalent feasible solutions.

Proposition 3. *If at least one feasible solution exists, there exists at least one feasible solution such that the following inequalities hold:*

$$x_k^m \geq x_{k'}^m \qquad m = 1, ..., n_M, k = 1, ..., |I'^m|, \; k' = 1, ..., |I'^m|, \sigma^m(k) = h^{-1}(\sigma^m(k'))$$

Proof. The relation $\sigma^m(k) = h^{-1}(\sigma^m(k'))$ expresses the fact that the virtual job $J_{\sigma^m(k')}$ has been generated to represent a job J_i processed before $J_{\sigma^m(k)}$, because of case 3. If $J_{\sigma^m(k)}$ is not processed, then there is no need to consider the virtual job $J_{\sigma^m(k')}$ associated with it. Thus, we allow selecting a virtual job only if the real job that generated it is selected. □

Proposition 4 tightens the upper bound of variables t_k^m and reduces the constants used in Constraints (3).

Proposition 4. *For all $m \in \{1,\dots,n_M\}$, let $\delta^m = \min_{\{k\in 1,\dots,|I'^m|\}} r_k^m$. Constraints (3) and (5) can be replaced by:*

$$t_k^m - (r'^m_k + p_k^m)x_k^m \geq 0 \quad m = 1,\dots,n_M,\ k = 1,\dots,|I'^m| \quad (7)$$

$$0 \leq t_k^m \leq \bar{d}'^m_k \quad m = 1,\dots,n_M,\ k = 1,\dots,|I'^m| \quad (8)$$

where $r'^m_k = r_k^m - \delta^m$ and $\bar{d}'^m_k = \max(\bar{d}'^m_{k-1}, \bar{d}'^m_k) - \delta^m$ and:

$$\bar{d}''^m_k = \begin{cases} \bar{d}_k^m - p_{h^{-1}(\sigma^m(k))} & \text{if } h^{-1}(\sigma^m(k)) \neq \emptyset \\ \bar{d}_k^m & \text{if } h^{-1}(\sigma^m(k)) = \emptyset \end{cases}$$

Proof. First, it is easy to see that all dates can be shifted together from any number δ^m of time slots without changing the set of integer solutions. One can verify that δ^m is chosen so that the constants of Constraints (3) are non-negative and as small as possible. Second, according to Proposition 3, a virtual job can be selected only if the real job that generated it is selected. Thus, if $J_{\sigma^m(k)}$ is a virtual job, then it must complete before $J_{h^{-1}(\sigma^m(k))}$, which must itself complete before $\bar{d}_k^m$. Hence, $J_{\sigma^m(k)}$ must complete before $\bar{d}_k^m - p_{h^{-1}(\sigma^m(k))}$. So, we can decrease the upper bound of t_k^m down to this value, if we do not to overconstrain the preceding jobs, i.e. if we do not prevent that $t_{k-1}^m = \bar{d}'^m_{k-1}$. □

Proposition 5 allows replacing some conjunctive constraints by knapsack constraints, which are usually better dealt with by MILP solvers (for example by generating cover cuts).

Proposition 5. *Let $m \in \{1,\dots,n_M\}$ and $k_*^m = \arg\max\{k \in \{1,\dots,|I'^m|\}/r'^m_k = 0\}$. Constraints (4) and (8) at ranks from 1 to k_*^m can be replaced by the following constraints:*

$$\sum_{l=1}^{k} p_l^m x_l^m \leq \bar{d}'^m_k \quad k \in 1,\dots,k_*^m \quad (9)$$

$$t^m_{k^m_*} \geq \sum_{l=1}^{k^m_*} p_l^m x_l^m \quad (10)$$

Proof. Since there is no release date for jobs before position k_*^m, $\sum_{l=1}^{k} p_l^m x_l^m$ is the completion time of job at position k on M_m if it is processed. Inequality (10) just defines $t^m_{k^m_*}$ for Constraint (4) to be valid at rank $k_*^m + 1$. □

Propositions 6 and 7 introduce Zero-Half cuts which can be added to the model to tighten its linear relaxation.

Proposition 6. *Let $m \in \{1,\dots,n_M\}$, $k \in \{2,\dots,|I'^m| - 1\}$, $u \in \{k,\dots,|I'^m| - 1\}$ and $v \in \{u+1,\dots,|I'^m|\}$. Let $\lfloor z \rfloor$, $z \in \mathbb{R}$ denote the largest integer not larger than z. If, for all $l \in \{u+1,\dots,v\}$, p_l^m is even, then the following inequality is valid:*

$$t_{k-1}^m + \sum_{l=k}^{u} p_l^m x_l^m + \sum_{l=u+1}^{v} \frac{p_l^m}{2} x_l^m \leq \left\lfloor \frac{\bar{d}'^m_u + \bar{d}'^m_v}{2} \right\rfloor$$

Proof. First note that Constraints (4) and (8) imply:

$$t_{k-1}^m + \sum_{l=k}^{q} p_l^m x_l^m \leq \bar{d}'^m_q \quad q \in \{k,\dots,|I'^m|\} \quad (11)$$

Let us sum up inequalities (11) at ranks u and v. We get:

$$2t_{k-1}^m + \sum_{l=k}^{u} 2p_l^m x_l^m + \sum_{l=u+1}^{v} p_l^m x_l^m \leq \bar{d}'^m_u + \bar{d}'^m_v \quad (12)$$

Let us divide (12) by 2:

$$t_{k-1}^m + \sum_{l=k}^{u} p_l^m x_l^m + \sum_{l=u+1}^{v} \frac{p_l^m}{2} x_l^m \leq \frac{\bar{d}'^m_u + \bar{d}'^m_v}{2} \quad (13)$$

On the left side of this inequality, all terms are integer. Thus, the value of the left term cannot be fractional, and we can tighten the inequality to obtain the claimed result. □

Proposition 7. *Let $m \in \{1,\dots,n_M\}$, $k \in \{1,\dots,|I'^m| - 1\}$, $u \in \{k,\dots,|I'^m| - 1\}$ and $v \in \{u+1,\dots,|I'^m|\}$. Let $\lfloor z \rfloor$, $z \in \mathbb{R}$ denote the largest integer not larger than z. If, for all $l \in \{u+1,\dots,v\}$, p_l^m is even, then the following inequality is valid:*

$$r'^m_k x_k^m + \sum_{l=k}^{u} p_l^m x_l^m + \sum_{l=u+1}^{v} \frac{p_l^m}{2} x_l^m \leq \left\lfloor \frac{\bar{d}'^m_u + \bar{d}'^m_v}{2} \right\rfloor$$

Proof. Similar to the proof of Proposition 6, except that it also uses Constraints (7). □

Note that the cuts provided by Propositions 6 and 7 are not dominated by the set of constraints defining the linear relaxation of (P) when $\bar{d}_u^m + \bar{d}_v^m$ is odd. In practice, adding all these cuts makes the MILP a lot larger and slows down its solution by a

solver. Moreover, we did not develop an efficient separation algorithm for these particular cuts, so that checking whether one of them is active given a fractional solution at each node of a search tree is not practically efficient. Therefore, we add a subset of the cuts directly in the model before its solution. To define this subset, we solve a family of $|I'|$ Linear Programs based on the linear relaxation of (A), in each of which we fix a single variable x_k^m to 1. Note that the procedure is fast since only a few dual-simplex iterations are usually required to solve an LP incrementally from the optimal basis of the previous problem. We add the cuts that are active in at least one of the solutions of the LPs and such that $u = k+1$, $v \in \{u+1, \ldots, \arg\max\{v' \geq u+1/p_q$ is even for $q \in \{u+1, \ldots, v'\}\}\}$, and $\bar{d}'^m_u + \bar{d}'^m_v$ is odd.

3 RESUMABLE JOBS

(Lee, 1996) shows that the problem $1, h_1|r-a|\sum U_i$ can be solved using Moore-Hodgson's algorithm initially designed to solve the problem $1||\sum U_i$ (Moore, 1968). Indeed, within the procedure, it is sufficient to add the duration $F_1 - B_1$ of the unavailability period to the completion time of the jobs completing after B_1. However, remark that considering a problem without availability constraints and adding unavailability periods, even with resumable jobs, sometimes changes the complexity of the problem: the author shows that $1, h_1|r-a|\sum w_i C_i$ is NP-hard in the ordinary sense, although $1||\sum w_i C_i$ is polynomial.

3.1 Conversion into $1|r_i|\sum w_i U_i$

In this section, we show that any instance of $1, h_k|r_i, r-a|\sum w_i U_i$ can be transformed into an equivalent instance of $1|r_i|\sum w_i U_i$. Let us denote by $s(t) = \arg\max\{s/B_s < t\}$ the index of the last unavailability period starting before t. Let us define the following function, which aims at shifting a time instant from a problem with unavailability periods to a problem without unavailability periods, by removing them and considering that no time elapse during them.

$$\Delta(t) = \begin{cases} t - \sum_{s=1}^{s(t)}(F_s - B_s) & \text{if } F_{s(t)} < t \\ B_{s(t)} - \sum_{s=1}^{s(t)}(F_s - B_s) & \text{otherwise} \end{cases} \quad (14)$$

The first case applies when t lies between two unavailability periods, while the second is for situations where t falls during an unavailability period. Any feasible solution Ω of an instance (P) of $1, h_k|r_i, r-a|\sum w_i U_i$ can be characterized by, for each job $J_i \in I$, a starting time $S(i)$ and a completion time $C(i)$. We claim that, to any feasible solution of (P), corresponds a feasible solution Ω' of an instance (P') of the problem $1|r_i|\sum w_i U_i$, with the same cost. (P') is defined by a set of jobs I' such that $|I'| = |I|$, and, for each job $J'_i \in I'$, a release date $r'_i = \Delta(r_i)$, a processing time $p'_i = p_i$, a due date $d'_i = \Delta(d_i)$ and a weight $w'_i = w_i$. The corresponding solution of (P') is defined by $S'(i) = \Delta(S(i))$ and $C'(i) = \Delta(C(i))$. In order to prove this, let us first observe that Δ is a non-decreasing function (we do not give the proof here for the sake of conciseness).

Proposition 8. *If $t \geq t'$, then $\Delta(t) \geq \Delta(t')$.*

Now, let us prove that the values of $S'(i)$ and $C'(i)$ are consistent.

Proposition 9. *For all job $J'_i \in I'$, $C'(i) = S'(i) + p_i$.*

Proof. The completion time of job J_i can be expressed as $C(i) = \beta(S(i), p_i, \bar{s})$, with $\bar{s} = \arg\min\{s \leq k/\beta(S(i), p_i, s) \leq B_{s+1}\}$ and $\beta(t,p,s) = t + p + \sum_{s'=s(t)+1}^{s}(F_{s'} - B_{s'})$. Clearly, $\beta(t,p,s)$ is the completion time of a job starting at t and whose processing time is p, if we take into account unavailability periods up to period s. We then seek for the first unavailability period $\bar{s}$ that does not postpone the completion of the job. Besides, notice that $s(C(i)) = \bar{s}$. After (14) and by remarking that a job never completes during an unavailability period, we have:

$$\begin{aligned} \Delta(C(i)) &= C(i) - \sum_{s=1}^{s(C(i))}(F_s - B_s) \\ &= S(i) + p_i + \sum_{s=s(S(i))+1}^{\bar{s}}(F_s - B_s) \\ &\quad - \sum_{s=1}^{\bar{s}}(F_s - B_s) \\ &= S(i) + p_i - \sum_{s=1}^{S(i)}(F_s - B_s) \end{aligned}$$

Besides, after (14), $\Delta(S(i)) = S(i) - \sum_{s=1}^{s(S(i))}(F_s - B_s)$. Hence, we have $\Delta(C(i)) - \Delta(S(i)) = p_i$. □

Proposition 10. *Ω' is a feasible solution of (P').*

Proof. Ω is feasible. So, $S(i) \geq r_i$ for all $J_i \in I$. Hence, after Proposition 8, $S'(i) = \Delta(S(i)) \geq \Delta(r_i) = r'_i$ and all release date constraints are satisfied by Ω'.
Moreover, let us consider any pair of jobs J_i and J_j processed consecutively in Ω. Then $S(j) \geq C(i)$. So, $S'(j) = \Delta(S(j)) \geq \Delta(C(i)) = C'(i)$. Hence and after Proposition 9, Ω' satisfies all conjunctive and disjunctive constraints.
Besides, let J_i be a late job in Ω. Then $C(i) > d_i$, and $C'(i) = \Delta(C(i)) > \Delta(d_i) = d'_i$ (we can show

that the latter inequality is strict because $C(i)$ cannot stand during an unavailability period). Consequently, J'_i is late in Ω' and a cost $w'_i = w_i$ is incurred. Let J_i be a on-time job in Ω. Then $C(i) \leq d_i$, and $C'(i) = \Delta(C(i)) \leq \Delta(d_i) = d'_i$ and J'_i is on-time. Hence, Ω and Ω' have the same cost. □

Based on a function $\Delta^+(t) = \min\{\beta(0,t,s)/\beta(0,t,s) \leq B_{s+1}\}$, we can show in a similar way that a feasible solution of (P) corresponds to each feasible solution of (P'), with the same cost. So, from (P), we can build an equivalent instance (P') of the problem $1|r_i|\sum w_iU_i$. This reduction from $1,h_k|r_i,r-a|\sum w_iU_i$ to $1|r_i|\sum w_iU_i$ can be done in polynomial time.

Note that equal release dates in (P) will yield equal release dates in (P'). Thus, this implies the following complexity results: $1,h_k|r-a|\sum U_i$ is polynomial and $1,h_k|r-a|\sum w_iU_i$ is NP-hard in the ordinary sense. Other results can be derived, for example that $1,h_k|r_i,r-a|\sum U_i$ is polynomial when release and due dates are similarly ordered, or that $1,h_k|r-a|\sum w_iU_i$ is polynomial when processing times and job weights are oppositely ordered...

3.2 Conversion into $(STWP)$

In the remainder of the paper, data names followed by a "′" (resp. "″") sign, like "r'_i" (resp. "r''_i") refer to data of instance denoted by (P') (resp. by (P'')). From an instance (P) of $1,h_k|r_i,r-a|\sum w_iU_i$, we first build the instance (P') of $1|r_i|\sum w_iU_i$ as explained in the previous section. Then, the conversion to an instance (P'') of $(STWP)$ is straightforward. Instance (P'') counts $n''_M = 2$ machines: the first one is for scheduling on-time jobs, while the second is a virtual machine which receives late jobs. For each job J'_i of (P'), create two jobs J''_i and J''_{n+i+1} in (P''), such that $r''_i = r'_i$, $\bar{d}''_i = d'_i$, $w''_i = 0$, $p''_i = p'_i$, $m''_i = 1$ and $r''_{n+i+1} = 0$, $d''_{n+i+1} = 1$, $w''_{n+i+1} = w'_i$, $p''_{n+i+1} = 0$ and $m''_{n+i+1} = 2$. Finally, create a group $G''_i = \{J''_i, J''_{n+i+1}\}$ for each job J'_i of (P').

4 NON RESUMABLE JOBS

The problem $1,h_k|nr-a|\sum U_i$ is NP-hard in the strong sense (this can be proved, for example, by reduction of 3-partition as suggested in (Lee, 1996) for $1,h_k|nr-a|C_{max}$). In this section, we propose two ways of converting this problem to $(STWP)$. The first one is similar to the transformation used in Section 3. The second one decomposes the time horizon according to the disjoint availability periods. In Section 5, we report significant differences in the practical solution of the problem depending on the transformation chosen.

4.1 Conversion into a two-machine $(STWP)$

Let us consider an instance (P) of $1,h_k|r_i,nr-a|\sum w_iU_i$ and let us build an equivalent instance (P') of $(STWP)$. For each job J_i of (P), we create a group of jobs composed of one job $J'_{i,s}$ for each unavailability period s where J_i can be processed, plus one fictitious late job: $G'_i = \{J'_{i,s}/1 \leq s \leq K+1, B_s \geq r_i + p_i, F_{s-1} + p_i \leq d_i\} \cup \{J'_{i,*}\}$. The late job $J'_{i,*}$ is such that $r'_{i,*} = 0$, $d'_{i,*} = 1$, $p'_{i,*} = 0$, $w'_{i,*} = w_i$, $m'_{i,*} = 2$. The time window of $J'_{i,s}$ is the intersection of the time window of J_i and the availability period $[F_{s-1}, B_s]$: $r'_{i,s} = \max(r_i, F_{s-1})$ and $d'_{i,s} = \min(d_i, B_s)$. Hence, the job representing J_i is forced to be processed out of the unavailability periods of the machine, and within the time window of J_i. The other characteristics of $J'_{i,s}$ are: $p'_{i,s} = p_i$, $w'_{i,s} = 0$ and $m'_{i,s} = 1$.

4.2 Conversion into a $K+1$-machine $(STWP)$

The idea of this conversion is that jobs that are not processed during the same availability period cannot interfere with each other. More precisely, the starting time of the first job processed in a period $[F_s, B_{s+1}]$ (which does not start before F_s) is not influenced by the completion time of the last job processed in $[F_{s-1}, B_s]$ (which does not complete after $B_s < F_s$). Therefore, we can shift down all dates of events occurring during $[F_s, B_{s+1}]$ by F_s time units. This is equivalent to consider that each availability period corresponds to an independent machine, on which jobs can be processed in parallel. Formally, to build an equivalent instance (P'') to (P) and (P'), we create $k+1$ machines. For each job in (P), we create a group of jobs composed of one job $J''_{i,s}$ for each unavailability period s where J_i can be processed, plus one fictitious late job: $G''_i = \{J''_{i,s}/1 \leq s \leq k+1, B_s \geq r_i + p_i, F_{s-1} + p_i \leq d_i\} \cup \{J''_{i,*}\}$. The late job $J''_{i,*}$ is such that $r''_{i,*} = 0$, $d''_{i,*} = 1$, $p''_{i,*} = 0$, $w''_{i,*} = w_i$, $m''_{i,*} = K+1$. The release and due dates of other jobs are defined as $r''_{i,s} = \max(r_i, F_{s-1}) - F_{s-1}$ and $d''_{i,s} = \min(d_i, B_s) - F_{s-1}$. Finally, $p''_{i,s} = p_i$, $w''_{i,s} = 0$ and $m''_{i,s} = s$. Compared to the two-machine transformation, this allows using smaller constants in Constraints (7) and (8).

5 COMPUTATIONAL RESULTS

This section reports computational results obtained by solving the problems through the transformations described in the paper, with help of the commercial MILP solver IBM ILOG Cplex v12.3, on a Personal Computer equipped with a 3.2 GHz quad-core processor and 3 GB RAM. The OS used is Windows Seven 32 bits, and so the RAM available for the program is limited to 2 GB.

In the results presented here, we applied all improvements described in Section 2.4, except Propositions 6 and 7. Indeed, they globally degrade the performance of the solver on our models. This can be explained by the fact that they make the MILP model larger, without sometimes bringing relevant information for its solution. However, as we show in the sequel, they sometimes help to solve difficult instances or to reduce the gap between the best known bounds for unsolved instances.

5.1 Resumable jobs

Section 3 shows that the problem $1, h_k|r_i, r - a|\sum w_iU_i$ is equivalent to $1|r_i|\sum w_iU_i$. So, to test our solving method, we generated instances for the latter as described in (Dauzère-Pérès and Sevaux, 2002). More precisely, the generator takes the following parameters as input: The number n of jobs, a release date factor R and a due date factor D. For each job J_i, a processing time p_i is drawn from a uniform distribution $\{1, \ldots, 100\}$. To each job i is assigned a release date r_i drawn from a uniform distribution $\{0, \ldots, nR\}$, and a due date d_i drawn from a uniform distribution $\{r_i + p_i, \ldots, r_i + p_i + nD\}$. Parameter R controls the dispersion of the release dates, while parameter D controls the size of the job execution windows. The parameters used to build our test bed are the combinations of $n \in \{100, 200, 250, 300\}$, $R \in \{1, 5, 10, 20\}$ and $D \in \{1, 5, 10, 20\}$. Ten instances are generated for each combination of these parameters, leading to a total of 640 instances.

	Time limit			
n	10 sec.	100 sec.	1000 sec.	3600 sec.
100	99.4%	100.0%	100.0%	100.0%
200	55.6%	96.9%	99.4%	100.0%
250	21.3%	96.3%	97.5%	98.8%
300	9.4%	90.6%	94.4%	96.3%

Table 1: Part of instances of $1, h_k|r_i, r - a|\sum w_iU_i$ and $1|r_i|\sum w_iU_i$ proved to be solved optimally according to different time limits.

Table 1 reports the part of instances solved to optimality (such that the optimality of the solution found is proved), according to the computing time required. All instances with 200 jobs are solved within one hour (applying Propositions 6 and 7 allows solving the only 200-job instance unsolved within 1000 seconds in 491 seconds), and 90.4% of the 300-job instances are solved within 100 seconds. These results are very good for the type of generic approach we use: to the best of our knowledge, the best method for solving $1|r_i|\sum w_iU_i$ has been published in (M'Hallah and Bulfin, 2007), where the authors describe a dedicated branch-and-bound and report success with up to 200 jobs (which is likely to perform better with today's computer power). It is interesting to note that all the failures to solve instances within one hour were caused by an out-of-memory status of the MILP solver. Moreover, the unsolved instances are all generated with the same set of parameters $R = 20$ and $D = 1$. A possible explanation is that the parameter $R = 20$ generates large release dates, which make the linear relaxation of the model weaker because of Constraints (7). Besides, parameter $T = 1$ generates narrow time windows and, thus, more constrained problems compared to other parameter combinations.

5.2 Non-resumable jobs

In order to generate a test bed for $1, h_k|r_i, nr - a|\sum w_iU_i$, we based on the generator used for resumable jobs and introduced two additional parameters. K is the number of unavailability periods in the instance, and UR is the ratio of unavailability of the machine over the planning horizon. We then generate K unavailability intervals $[B_s, F_s]$ such that B_s is drawn from a uniform distribution $\{\min_i p_i, \ldots, \max_i d_i - \min_i p_i\}$ and $F_s = B_s + 1 + \frac{UR \times \max_i d_i}{100 \times K}$. If two generated periods overlap or are adjacent, the instance is rejected. We report results for all combinations of $n \in \{30, 50, 60, 100, 150, 200, 250, 300\}$, $R \in \{1, 5, 10, 20\}$, $D \in \{1, 5, 10, 20\}$, $K \in \{1, 3, 5\}$ and $UR \in \{1, 5, 10\}$. Five instances are generated for each combination of parameters, leading to a total of 5760 instances.

5.2.1 Two-machine transformation

Table 2 gathers the part of instances solved using the two-machine transformation of $1, h_k|r_i, nr - a|\sum w_iU_i$. One 60-job instance is left open by this method and almost 10% of the 100-job instances are not solved. Therefore, we do not report results for larger instances in this section.

5.2.2 $K + 1$-machine transformation

Table 3 summarizes the results obtained using the $K + 1$-machine transformation of $1, h_k|r_i, nr - a|\sum w_iU_i$. This method performs much better than the previous one: while the two-machine transformation leaves on 60-job instance open in 1000 seconds, the maximum computing time for 60-job instances is only 1.77 seconds for the $K + 1$-machine transformation. Moreover, only one 100-job instance is solved in more than ten seconds, only one 200-jobs instance is not solved and more than 98% of the 300-job instances are solved.

Let us focus on the set of 300-job instances to analyze the sensitivity of the method with respect to generation parameters. Instances with a larger ratio of unavailability are slightly easier to solve: 98.8% when $UR = 5$ against 96.7% when $UR = 1$. It is

	Time limit			
n	1 sec.	10 sec.	100 sec.	1000 sec.
30	100.0%	100.0%	100.0%	100.0%
50	87.1%	98.9%	99.9%	100.0%
60	71.8%	95.8%	99.6%	99.9%
100	28.6%	69.9%	83.6%	90.1%

Table 2: Results for the two-machine transformation of $1, h_k|r_i, nr-a|\sum w_iU_i$: Part of instances proved to be solved optimally according to different time limits.

	Time limit			
n	1 sec.	10 sec.	100 sec.	1000 sec.
30	100.0%	100.0%	100.0%	100.0%
50	100.0%	100.0%	100.0%	100.0%
60	97.8%	100.0%	100.0%	100.0%
100	69.6%	99.9%	100.0%	100.0%
150	25.8%	98.6%	100.0%	100.0%
200	7.1%	81.3%	99.2%	99.9%
250	1.9%	64.4%	96.9%	99.0%
300	0.6%	43.6%	93.9%	98.1%

Table 3: Results for the $K+1$-machine transformation of $1, h_k|r_i, nr-a|\sum w_iU_i$: Part of instances proved to be solved optimally according to different time limits.

easily explained because the less the machine is available, the smaller the number of possible positions for each job is. Concerning the impact of the number of unavailability period, we can hypothesize that instances with one single unavailability period are more difficult than those with three because their structure is relatively close to the case without availability constraints. Thus, when $K = 1$ (97.9% solved), the combinatorial difficulty comes from the sequencing of the jobs, enhanced by a second level of decision consisting in assigning the suitable part of the horizon for each job. When $K = 3$ (99.6% solved), this source of hardness is lessen because the sequencing of jobs is easier. Indeed, the time windows of jobs are likely to me modified, so that the number of jobs with strictly nested time windows (Figure 3) is relatively small. Hence, the number of virtual jobs is relatively small. When the number of unavailability periods grows ($K = 5$, 96.7% solved), then the assignment component of the problem becomes more difficult to solve, and this is no more balanced by the, although even more important, simplification of the sequencing component. This hypothesis is confirmed by the apparently higher difficulty of the special case of periodic maintenance discussed below, which practically generates ($STWP$) problems with many machines.
Regarding the release date and due date factors, the combination $R = 20$, $D = 1$ yields once again the most difficult instances for our method, since only 75.6% of these instances are solved.

Impact of Propositions 6 and 7 When applying Proposition 6 and 7 as described in Section 2.4 on the set of unsolved non-resumable instances, the only 200-job instance left open is solved in 398 seconds. One (resp. four) of the seven 250-job (resp. fourteen 300-job) instances is solved. The average relative gap for the still open instances, calculated as (best upper bound-best lower bound)/best lower bound, is reduced from 0.5% to 0.37%. However, as stated before, the results obtained are globally degraded. This is explained because difficult instances (with $R = 20$ and $D = 1$) have a poorer linear relaxation because of large coefficients used in Constraints (7). The cuts added improve the relaxation and improve the results on these instances. However, for other instances, the difficulty seems to come from the larger number of variables and constraints while the linear relaxation is better, so that making the MILP larger just make it more difficult to solve.

Periodic maintenance constraints In order to compare our methods with other work of the literature, we interest in the problem of minimizing the number of tardy jobs on a single machine subject to periodic maintenance, investigated by (Chen, 2009). The author proposes a branch-and-bound to solve optimally instances with up to 32 jobs, in about 500 seconds on average (these are the results reported in the paper and we did not run the program on our machine). In this particular problem, jobs are non-resumable, and unavailability periods are placed according to a regular pattern over the planning horizon: the machine is available for T time slots, then under maintenance during t time slots, and this pattern is repeated until the end of the planning period. This is clearly a special case of $1, h_k|nr-a|\sum U_i$.
We generated a set of instances as described in (Chen, 2009): For each of the n jobs, a processing time is randomly generated from a discrete uniform distribution (DU) over $[1, 10]$. The due dates are selected from another DU over $[(1-C-Q/2)\sum_i p_i, (1-C+Q/2)\sum_i p_i]$ with restriction $d_i \geq 0$, where Q and C are the due date and tardiness factors, respectively. The parameters chosen are all combinations of $n \in \{32, 50, 100\}$, $C \in \{0.2, 0.6\}$, $Q \in \{0.2, 0.6\}$, $T \in \{10, 15, 20\}$ and $t \in \{3, 6\}$. For each combination, we generated 30 instances, leading to a total of 1440 instances.

Table 4 indicates that most instances with 100 jobs can be solved in less than 10 seconds. From Table 5, we see that our method outperforms the previously published approach, even if we take into account the evolution of computer processing power: only 0.2 seconds are needed in average to solve optimally 32-job instances. However, the table also point out a weakness of our method: even if a few seconds are sufficient to solve most instances, it is sometimes un-

	Time limit			
n	1 sec.	10 sec.	100 sec.	1000 sec.
32	98.8%	99.7%	100.0%	100.0%
50	95.1%	99.2%	99.4%	99.7%
100	66.1%	93.2%	95.0%	96.0%

Table 4: Special case with periodic maintenance and equal job weights: Part of instances proved to be optimally solved using the $K+1$-machine transformation according to different time limits.

able to prove the optimality of solutions for instances of the exact same size and generation parameters in less than 1000 seconds. This is certainly partially explained by the high degeneracy of these instances. More precisely, the availability periods are relatively short compared to the planning horizon. That is why, in practice, there are many unavailability periods. Moreover, there is no release date constraint. Therefore, the problem, in the way we model it, is close to an assignment problem with many identical resources (machines of $(STWP)$ with an equivalent set of jobs that can be assigned). The difficulty comes from the fact that, if there are q identical machines, each feasible assignment of jobs to machines counts $q! - 1$ other feasible equivalent assignments obtained by permuting the set of jobs over the identical machines. In order to improve the performance of our method on this problem, we could investigate ways of breaking these symmetries. Note that, for unsolved instances, absolute gaps are very small on average.

n	Avg time in sec.	Max time in sec.	Avg unsolved abs. Gap
32	0.2	37.5	-
50	0.9	>1000	1
100	4.9	>1000	1.53

Table 5: Special case with periodic maintenance and equal job weights: average and maximum solving time, average absolute gap after 1000 seconds for unsolved instances.

6 CONCLUSION

This paper proposes different ways of transforming scheduling problems with machine availability constraints into a more general problem. We show that designing a suitable transformation leads to a competitive exact solution method based on a MILP solver for this type of problems. Indeed, although we lack of points of comparison for the problem $1, h_k|r_i, nr-a|\sum w_iU_i$, the results obtained are comparable to the state-of-the-art for the special case of the problem $1|r_i|\sum w_iU_i$, and outperform a previously published solution approach for the special case of periodic maintenance. Further work could consist in developing a dedicated approach for solving the problem $(STWP)$.

REFERENCES

Carlier, J., 1982. The one-machine sequencing problem. *European Journal of Operational Research*, 11(1), p. 42-47.

Chen, W.-J, 2009. Minimizing number of tardy jobs on a single machine subject to periodic maintenance. *Omega* 37(3), p. 591-599.

Dauzère-Pérès, S., and Sevaux, M., 2002. Using Lagrangean relaxation to minimize the weighted number of late jobs on a single machine. *Naval Research Logistics*, 50, p. 273-288.

Detienne, B., Dauzère-Pérès, S., and Yugma, C., 2011. Scheduling jobs on parallel machines to minimize a regular step total cost function. *Journal of Scheduling*, 14(6), p. 523-538.

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

Jackson, J. R., 1955. Scheduling a production line to minimize maximum tardiness (Research Report 43). *Management Science Research Project*, University of California, Los Angeles.

Lawler, E.L., Martel, C.U., 1989. Preemptive scheduling of two uniform machines to minimize the number of late jobs. *Operations Research*, 37, p. 314-318.

Lee, C.-Y., 1996. Machine scheduling with an availability constraint. *Journal of Global Optimization*, 9(3), p. 395-416.

M'Hallah, R., and Bulfin, R.L., 2007. Minimizing the weighted number of tardy jobs on a single machine with release dates. *European Journal of Operational Research*, 176(2), p. 727-744.

Ma, Y., Chu, C. and Zuo, C., 2010. A survey of scheduling with deterministic machine availability constraints. *Computers & Industrial Engineering*, 58(2), p. 199-211.

Mauguière, Ph., Billaut, J.-C., and Bouquard, J.-L., 2005. New Single Machine and Job-Shop Scheduling Problems with Availability Constraints. *Journal of Scheduling*, 8(3), p. 211-231.

Moore, J.M., 1968. A n job one machine sequencing algorithm for minimizing the number of late jobs. *Management science*, 15, p. 102-109.

Schmidt, G., 2000. Scheduling with limited machine availability. *European Journal of Operational Research*, 121(1), p. 1-15.