A Mixed Integer Linear Programming approach to minimize the number of late jobs with and without machine availability constraints

Boris Detienne

Institut de Mathématiques de Bordeaux - Université Bordeaux 1
Team RealOpt - INRIA Bordeaux Sud-Ouest, France

Abstract

This study investigates scheduling problems that occur when the weighted number of late jobs that are subject to deterministic machine availability constraints have to be minimized. These problems can be modeled as a more general job selection problem. Cases with resumable, non-resumable, and semi-resumable jobs as well as cases without availability constraints are investigated. The proposed efficient mixed integer linear programming approach includes possible improvements to the model, notably specialized lifted knapsack cover cuts. The method proves to be competitive compared with existing dedicated methods: numerical experiments on randomly generated instances show that all 350-job instances of the test bed are closed for the well-known problem $1|r_i|\sum w_i U_i$. For all investigated problem types, 98.4% of 500-job instances can be solved to optimality within one hour.

Keywords: Scheduling, Integer programming, Modeling, Availability constraints, Late jobs, Exact method

1. Introduction

Even though most scheduling problems assume that a job can be processed anytime after its release date, an increasing number of research papers investigate scheduling problems that are subject to machine or job availability constraints. In this context, machines or jobs may be unavailable for processing during given time intervals. Such constraints appear [Sch00, KBF$^+$02, MSCK09, HDV12, MKSC13] in the cases of joint production and maintenance management (where preventive maintenance or periodic repair tasks limit the usage time of machines), operating systems for mono and multi-processors (where high priority tasks interfere with low priority programs) or when implementing

Email address: `boris.detienne@u-bordeaux1.fr` (Boris Detienne) rolling time horizon approaches (where decisions taken for a first time horizon are considered as fixed for the consecutive overlapping time horizon)...

The problems investigated in this paper are defined by a set $I = \{J_1, \dots, J_n\}$ 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. Additional input consists of a set of K time intervals $[B_s, F_s]$, $s \in \{1, \dots, K\}$, when the machine is unavailable. Moreover, we add two fictitious unavailability periods 0 and $K+1$ with no loss of generality, such that $B_0 = F_0 = 0$ and $B_{K+1} = F_{K+1} = \infty$. All data are integers and deterministic.

We investigate different job behaviors in terms of unavailability constraints. First, jobs are said to be *non-resumable*, according to the classification of [MCZ10]: if a job has started but has not completed before an unavailability period, it must be restarted from zero after the unavailability period. This problem is denoted by $1, h_k|r_i, nr-a|\sum w_iU_i$ in standard three-field notation [GLL$^+$79]. Conversely, jobs may be *resumable*: a job may start before an unavailability period and be resumed thereafter. 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 $1, h_k|r_i, r-a|\sum w_iU_i$. A third type of problem allows jobs to be resumed after an unavailability period at the expense of an additional setup time. Two variations are possible for this feature, which we refer to as *semi-resumable* jobs: according to [Lee99], interrupted jobs must be partially restarted after an unavailability period, occupying the machine for a time which is proportional to the duration of the job that has already been processed. In this paper, we address a variation of this feature for which the additional setup time is fixed and job-dependent, analogous to the work of [GL99]. More precisely, the setup duration is equal to $\min(\alpha_i, B_{s+1} - F_s)$, where α_i is a fixed setup time related to job J_i. The hypothesis is motivated by the following: The industrial process does not require partial job reprocessing in many real-life situations, especially in the case of preventive and planned maintenance. Instead, the machine requires a setup operation, whose duration only depends on the job to be resumed. An example of this kind of requirement is encountered in the corrugated cardboard industry following the periodic cleaning operations of cutting devices. These setup times are implicitly included in job processing times when no maintenance activities are considered. Also, sometimes a setup operation following F_s will not be completed before B_{s+1}, and another setup time will be performed after F_{s+1}. Therefore, in many practical situations, the job will simply be deferred until F_{s+1}. According to the classification of [ANCK08], these are referred to as *non-batch sequence-independent setup times*. The problem is denoted by $1, h_k|r_i, ST_{si}|\sum w_iU_i$, of which problems $1, h_k|r_i, r-a|\sum w_iU_i$ and $1|r_i|\sum w_iU_i$ are clearly special cases.

This paper aims to provide a generic mixed integer linear programming (MILP) approach to solve problems $1, h_k|r_i, nr-a|\sum w_iU_i$ and $1, h_k|r_i, ST_{si}|\sum w_iU_i$ and their special cases $1, h_k|r_i, r-a|\sum w_iU_i$ and $1|r_i|\sum w_iU_i$. Because problem

$1|r_i|\sum w_iU_i$ is strongly NP-hard [GLL+79], so are the other problems. To the best of our knowledge, to date no published work proposes an exact solution approach for cases with availability constraints, nor an efficient MILP approach for $1|r_i|\sum w_iU_i$.

Our paper is structured as follows: Section 2 contains a brief literature review for related scheduling problems. Section 3 describes the general scheduling problem ($STWP$) of selecting and scheduling jobs that are subject to time windows and group constraints on parallel machines; three MILP models are proposed. Section 4 presents valid inequalities and bound tightening results to improve these models. In particular, we use one of the MILP formulations to derive lifted knapsack cover cuts, which are also valid for the other models. We propose two specialized lifting procedures, embedding a subset of the specific constraints of $STWP$ to yield stronger cuts. Section 5 describes how problems $1, h_k|r_i, nr-a|\sum w_iU_i$ and $1, h_k|r_i, ST_{si}|\sum w_iU_i$ can be converted into $STWP$. Moreover, we show that problem $1, h_k|r_i, r-a|\sum w_iU_i$ is equivalent to $1|r_i|\sum w_iU_i$, which proves some complexity results. Our method proves to be reasonably competitive compared with existing dedicated methods: in Section 6, numerical experiments on randomly generated instances show that most (98.4%) 500-job instances can be solved to optimality within one hour for all investigated problem types. For the well-known problem $1|r_i|\sum w_iU_i$, all 350-job instances are closed and only one (resp. four) 400-job instance (resp. 500-job instances) remains open. Section 7 discusses possible extensions and improvements to this work. Finally, an Appendix provides details concerning the dynamic programming algorithms used by the lifting procedures.

2. Literature review

The problem of minimizing the (weighted) number of late jobs on one machine has been studied extensively. For the general case $1|r_i|\sum w_iU_i$, [DP95, DPS02] present lower bounds and heuristics and [BPP03, PPR03, Sad08] describe algorithms to solve the problem to optimality. To our knowledge, the best known exact algorithms have been contributed by [MB07] for the weighted case and by [KSJH12] for the unweighted case $1|r_i|\sum U_i$. These methods make it possible to solve instances with up to 200 and 300 jobs, respectively. [BS09] present a generic MILP model for total cost scheduling problems with piecewise linear objective functions. According to their numerical experiments, this very general model does not perform as well as existing dedicated approaches.

Concerning only the case of deterministic machine availability constraints (when unavailability periods are perfectly known in advance), [MCZ10] gathers more than 90 research papers. Interestingly, only one paper mentioning the minimization of the number of late jobs is referenced in this survey: [Lee96] shows that $1, h_1|r-a|\sum U_i$ is polynomial and that Moore-Hodgson's algorithm [Moo68] leads to an absolute error of 1 to solve the problem $1, h_1|nr-a|\sum w_iU_i$. Another survey [Sch00] cites [LM89], who studies the minimization of the number of late jobs on two machines whose speeds can vary (and possibly be null) over time if preemption is allowed. [Che09], who studies a special case of

$1, h_k|nr-a|\sum U_i$ in which unavailability intervals correspond to periodic maintenance, develops a branch-and-bound to optimally solve instances with up to 32 jobs. [ZOW14] studies the Order Acceptance and Scheduling problem with such constraints, which is an extension of the problem.

Semi-resumable jobs with non-batch sequence-independent setup times were studied in [GL99], where the authors consider both job and maintenance planning for problems $1|ST_{si}|L_{max}$ and $1|ST_{si}|\sum w_iC_i$. The study provides complexity results and algorithms for some polynomial and ordinary NP-hard special cases. [LC02] and [LC04] investigate problems $1|ST_{si}, pmnt, r_i|D_{max}$ and $1|ST_{si}, pmnt, r_i|\sum w_iC_i$, where a fixed setup time occurs after each job preemption. A special case of STWP is studied in [SW06], where the authors combine MILP and constraint programming techniques to solve $R|r_i, d_i|\sum c_{ij}$, for which jobs have to be assigned to machines so that the assignment cost is minimized and all jobs are processed within their respective time window. The authors report success for instances with up to nine machines and 54 jobs.

3. The problem of scheduling jobs with time windows

This section describes the problem of selecting jobs with time windows on parallel processors (STWP), which is solved with the help of an efficient mixed integer linear program.

3.1. 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, $i \in \{1, \ldots, n_I\}$ 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 $[r_i, \bar{d}_i]$. The objective is to minimize the total processing cost.

This problem clearly generalizes the problem of minimizing the number of tardy jobs on parallel machines ($P||\sum U_i$); it is NP-hard in the strong sense [GJ79] when the number of machines is arbitrary.

3.2. Characterizing feasible job selections

This section introduces a dominance property for optimal schedules of $STWP$. It is a generalization of a result proposed in [DDY11], where the authors present MILP models to minimize a regular step cost function of job completion times on parallel machines. A similar principle was also used in [DPS02] to minimize the number of late jobs on a single machine. Their prior work and our current investigation are both based on the idea of inducing a strict order on the set of processed jobs to derive strong MILP models by greatly simplifying the expression of both conjunctive and disjunctive constraints. However, these two approaches lead to different dominant orders as well as different models resulting in different possible improvements.

The core idea can be seen as an extension of the earliest deadline first (EDF) rule [Jac55], which is dominant in the special case of jobs with equal release dates. Additional virtual jobs are added to the instance data in order to obtain a similar result when release dates are arbitrary. Formally, for each job $J_i \in I$, the set of associated virtual jobs is defined by

$$H_i = \{J_{h(i,j)} | J_j \in I, m_i = m_j, r_i < r_j, \bar{d}_i > \bar{d}_j, r_i + p_i + p_j \leq \bar{d}_j\}.$$

In this expression, $h(i, j)$ denotes the index of the unique virtual job created to represent the processing of job J_i before job J_j when their time windows are not agreeable. For each virtual job $J_{h(i,j)}$, $J_i \in I$, $J_j \in I - \{i\}$, $J_{h(i,j)} \in H_i$, we define $h^{-1}(h(i, j)) = j$, and for each actual job $J_i \in I$, $h^{-1}(i) = \emptyset$. 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 enhanced set of jobs is denoted by $I' = I \cup (\cup_{i=1}^{n_I} H_i)$, and we set $n_{I'} = |I'|$. The new problem is also characterized by enhanced groups of jobs: $G' = \{G'_1, \ldots, G'_{n_G}\}$, with $G'_g = G_g \cup (\cup_{J_i \in G_g} H_i)$, $g \in \{1, \ldots, n_G\}$. Also, 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.

Proposition 1. *Consider an instance Ω of $STWP$ and the corresponding enhanced set of jobs I'. There is at least one optimal solution to Ω such that, on each machine, selected jobs are scheduled according to a non-decreasing order of their deadlines, with ties being broken according to a non-decreasing order of their release dates.*

PROOF. Consider an optimal schedule S, and focus on a single machine M_m. Let J_i and J_j be two jobs processed consecutively on M_m (J_i precedes J_j). Let C_i (resp. C_j) denote the completion time of J_i (resp. J_j) in S. Observe that active schedules are dominant for $STWP$. We therefore assume that each selected job starts as soon as possible. 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:** $\bar{d}_i < \bar{d}_j$ or ($\bar{d}_i = \bar{d}_j$ and $r_i < r_j$). J_i and J_j are scheduled according to a non-decreasing order of their deadlines.

- **Case 2:** $\bar{d}_i \geq \bar{d}_j$ and $r_i \geq r_j$. Consider a schedule S', built from S by swapping J_i and J_j. Both J_i and J_j clearly meet their deadlines in S'. Moreover, J_i completes no later in S' than J_j in S, so that the latest part of the schedule is not modified. Because the set of selected jobs does not change, the cost of S' equals the cost of S, and S' is optimal. Finally, J_i and J_j are scheduled in the desired order.

- **Case 3:** $\bar{d}_i > \bar{d}_j$ and $r_i < r_j$. First, note that $r_i + p_i + p_j \leq \bar{d}_j$ because S is feasible. Then, by construction, there exists a job J_k with $k = h(i, j)$, such that $m_k = m$, $r_k = r_i$, $\bar{d}_k = \bar{d}_j$, $p_k = p_i$, and $w_k = w_i$. Consider the schedule S' built from S by selecting J_k instead of J_i and placing it at the same position on machine M_m. Because both jobs have the same release date and processing time, the completion time of other jobs does not change; all jobs still meet their deadlines. Because $C'_k = C_i$ and $C_i < C_j \leq \bar{d}_j$, we have $C'_k < \bar{d}_k$. And because J_k and J_i belong to the same group G_g and possess the same processing cost, S' is feasible and also optimal. Finally, J_k and J_j are scheduled in the desired order.

3.3. Generalized existing ILP models

The following two ILP models are generalizations of models described in [DDY11]. In order to build them, 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, with ties being broken according to an arbitrary non-decreasing order of their release dates. Additionally, 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 notation: $r_k^m = r_{\sigma^m(k)}$ is the release date of the k^{th} job on machine M_m. We use a similar notation for durations, deadlines, and weights. Then, problem STWP can be expressed as follows:

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

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

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

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

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

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

In this model, the 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, or else it is equal to 0. Constraints (2) ensure that exactly one job of each group is selected. Constraints (3) state that each job cannot start before its release date, whereas 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, because 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 in the range k', $k' < k$. Expression (1) is the objective function, which minimizes the sum of the costs of selected jobs. Note that Constraints (3), (4), and (5) concern the feasibility of the selection of jobs from a resource point of view. [DDY11] describe another ILP characterization of these constraints for their problem as a set of knapsack constraints, which are used in an iterative constraint generation procedure. The adaptation to STWP is straightforward and leads to a multiple-choice multidimensional knapsack problem formulation ($MMKP$) by replacing (3), (4), and (5) with

$$(r_k^m + p_k^m)x_k^m + \sum_{k'=k+1}^{l} p_{k'}^m x_{k'}^m \leq \bar{d}_l^m \quad m = 1, \ldots, n_M, k = 1, \ldots, |I'^m|, l \geq k.$$

3.4. Alternative model

The main drawback of formulation (A) is that Constraints (3) induce a poor linear relaxation when some jobs have large release dates. The ($MMKP$) formulation provides a better linear relaxation, but cannot be used directly due to its large number of constraints.

This section is devoted to a new model aimed at circumventing those drawbacks by keeping a relatively small size and providing a stronger linear relaxation. This MILP model is inspired by the work of [DPS02], in which the authors establish a related dominant order (represented by a so-called master sequence) for $1|r_i|\sum w_i U_i$. They propose a MILP model that also makes use of large constants, but, in practice, these constants are smaller than those involved in Constraints (3). Unfortunately, their model is based on the master sequence, assuming that selected jobs are sequenced according to a non-decreasing order of their release dates, which is the wrong assumption in our case.

We then arrive at the following key idea: Consider a single machine problem, suppose that a selection of jobs is fixed (and that the sequence of jobs is fixed accordingly), and focus on the timing sub-problem. This sub-problem consists in checking whether all selected jobs can be scheduled according to this sequence while satisfying their time windows. For this purpose, a simple strategy is to schedule selected jobs as soon as possible on each machine, and verify that all jobs complete before their deadline. We can now reverse the direction of time in the timing problem by considering that the time horizon ends at time 0 and starts at a sufficiently late time instant (e.g. $\bar{d}_{max} = \max_k \bar{d}_k^m$). Let us assign reverse time windows to each job, defined by $\hat{r}_k^m = \bar{d}_{max} - d_k$ and $\hat{d}_k^m = \bar{d}_{max} - r_k$. Immediately, the timing problem and its reverse counterpart are equivalent, and when jobs are sorted according to the dominant order used in Proposition 1, they are sorted according to a non-decreasing order of their reverse release dates $\hat{r}$. We can thus write the following model:

$$(B) \qquad \min \sum_{m=1}^{n_M} \sum_{k=1}^{|I'^m|} w_k^m x_k^m \tag{7}$$

$$\sum_{m=1}^{n_M} \sum_{k|J_k \in G_g'^m} x_k^m = 1 \qquad g = 1, \ldots, n_G \tag{8}$$

$$\hat{t}_k^m \geq \hat{r}_k^m \qquad m = 1, \ldots, n_M, k = 1, \ldots, |I'^m| \tag{9}$$

$$\hat{t}_{k-1}^m - \hat{t}_k^m - p_k^m x_k^m \geq 0 \qquad m = 1, \ldots, n_M, k = 2, \ldots, |I'^m| \tag{10}$$

$$\hat{t}_k^m + p_k^m x_k^m - M_k^m (1 - x_k^m) \leq \hat{d}_k^m \qquad m = 1, \ldots, n_M, k = 1, \ldots, |I'^m| \tag{11}$$

$$x_k^m \in \{0, 1\}, \hat{t}_k^m \geq 0 \qquad m = 1, \ldots, n_M, k = 1, \ldots, |I'^m| \tag{12}$$

In this model, we assume that jobs are sorted as for model (A) and $(MMKP)$, and variables (x_k^m) have the exact same meaning as in both previous models. Variable $\hat{t}_k^m$ denotes the starting time in the reverse timing problem of the k^{th} job on machine M_m. Constraints (9) state that no job can start before its release date. Constraints (10) ensure that no job starts before the preceding job has completed. Finally, Constraints (11) force each selected job to complete before its deadline. Constant M_k^m is used not to over constrain the starting time of unselected jobs. The value of constant M_k^m can safely be set to $M_k^m = \max(0, \max_{l>k}\{\hat{d}_l^m - \hat{d}_k^m\})$.

4. Improvements

This section provides several potential improvements to the models presented in the previous section. An interesting feature of these MILP is that they share the set of variables x, which have the exact same meaning in all three of the models. Hence, any relation involving this set of variables can be used in any model. In the sequel, let us assume that, on each machine $M_m \in M$, the enhanced set of jobs I'^m is ordered according to Proposition 1.

4.1. Strengthening the models

The following two propositions exploit the classical selections on disjunction [see e.g. Car82] to strengthen the model.

Proposition 2. *Let $J_i \in I$ and $J_j \in I$, $i \neq j$, $m_i = m_j = m$. If $r_i + p_i + p_j > \bar{d}_j$ and $r_j + p_j + p_i > \bar{d}_i$, 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 simply expresses that J_i and J_j (or one of their corresponding virtual jobs) cannot both be processed in a feasible solution. □

Proposition 3. *Let $M_m \in M$, $k \in \{1, \ldots, |I'^m|\}$, $l \in \{k+1, \ldots, |I'^m|\}$. If $r_k^m + p_k^m + p_l^m > d_l^m$, then the following inequality holds:*

$$x_k^m + x_l^m \leq 1$$

Proof. The inequality simply expresses that $J_{\sigma^m(k)}$ and $J_{\sigma^m(l)}$ (actual or virtual jobs) cannot both be processed in a feasible solution. □

Proposition 4 strengthens the model by removing some equivalent feasible solutions.

Proposition 4. *If at least one feasible solution exists, there exists at least one optimal solution such that the following inequalities hold:*

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

Proof. The relation $\sigma^m(k) = h^{-1}(\sigma^m(k'))$ expresses the fact that virtual job $J_{\sigma^m(k')}$ has been generated to represent a job J_i processed before $J_{\sigma^m(k)}$, as a result of case 3. If $J_{\sigma^m(k)}$ is not processed, then J_i (or another virtual job created to represent J_i) can replace $J_{\sigma^m(k')}$ in another feasible schedule in which jobs are also sequenced according to Proposition 1. Therefore, there is no need to select the virtual job $J_{\sigma^m(k')}$ associated with it. We thus allow the selection of a virtual job only if the real job that generated it is selected. □

The following propositions allow us to tighten the time windows of jobs by showing that their deadlines can be decreased.

Proposition 5. *Let S be an optimal schedule such that S is active, and selected jobs are sequenced according to Proposition 1 on each machine. For all $m \in \{1, \ldots, n_M\}$, $k \in \{1, \ldots, |I'^m|\}$, let $D^m(k) = \max\{t | t \in \{0, \ldots, \bar{d}_k^m\}, D^m(k,t) = 1\}$, with*

$$D^m(k,t) = \begin{cases} 1 & \text{if } \big(D^m(k-1, t-p_k^m) = 1 \quad \vee t = r_k^m + p_k^m \\ & \qquad\qquad\qquad\qquad\quad \vee D^m(k-1,t) = 1\big) \\ 0 & \qquad \text{otherwise} \end{cases} \qquad k = 1, \ldots, |I'^m|, t = 0, \ldots, \bar{d}_{|I'^m|}^m$$

And $D^m(0,t) = 0$, $m \in \{1, \ldots, n_M\}$, $t = 0, \ldots, \bar{d}_{|I'^m|}^m$. Then, for all $k \in \{1, \ldots, |I'^m|\}$, $J_{\sigma^m(k)}$ does not complete after $D^m(k)$.

Proof. Clearly, $D^m(k,t) = 1$ if and only if at least one job among the first k that can be processed on M_m can complete at time t. Indeed, $J_{\sigma^m(k)}$ can complete at t only if $t \leq \bar{d}_k^m$ and at least one of the following propositions holds:

- $J_{\sigma^m(k)}$ starts at its release date (i.e. $t = r_k^m + p_k^m$).

- $J_{\sigma^m(k)}$ starts at the completion time of a preceding job. In this case, there exists $J_{\sigma^m(k')}$, $k' < k$, such that $D^m(k', t) = 1$, and then $D^m(k-1, t) = 1$ holds. □

Note that all $D^m(k)$ values can be computed using an $\mathcal{O}(|I'^m| \cdot \bar{d}^m_{|I'^m|})$-time and $\mathcal{O}(\bar{d}^m_{|I'^m|})$-space dynamic programming algorithm.

Proposition 6. *For all $m \in \{1, \ldots, n_M\}$, let $\delta^m = \min_{\{k \in 1, \ldots, |I'^m|\}} r^m_k$. For all $m \in \{1, \ldots, n_M\}$, for all $k \in \{1, \ldots, |I'^m|\}$, replacing the time window $[r^m_k, \bar{d}^m_k]$ with $[\dot{r}^m_k, \dot{d}^m_k]$ leads to the same set of optimal selections of jobs, where $\dot{r}^m_k = r^m_k - \delta^m$, $\dot{d}^m_k = \max(\dot{d}^m_{k-1}, e^m_k) - \delta^m$ and*

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

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, as soon as none of the quantities are negative. We can verify that δ^m is chosen so that all dates are nonnegative and as small as possible. Second, according to Proposition 5, $J_{\sigma^m(k)}$ cannot complete after $D^m(k)$. Moreover, according to Proposition 4, 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 $D^m(k)$. Hence, $J_{\sigma^m(k)}$ must complete before $D^m(k) - p_{h^{-1}(\sigma^m(k))}$. We can therefore decrease the deadline of $J_{\sigma^m(k)}$ to this value if we do not over constrain the preceding jobs, i.e. if we do not prevent that $J_{\sigma^m(k-1)}$ completes at $\dot{d}^m_{k-1}$. □

4.2. Specialized knapsack cover cuts

Lifted knapsack cover inequalities have proved to be a powerful tool for solving (mixed) integer linear programs involving knapsack constraints [Bal75, CJP83]. The concept was substantially improved by introducing lifted generalized upper bound cover inequalities [GNS98] and sequence-independent lifting procedures [Wol77, GNS00, LW07].

This section presents valid inequalities that can be used to further strengthen our models. As already mentioned, any relation involving the set of variables x can be used in any of the three models presented in Section 3. Thus, we use the $(MMKP)$ formulation to identify classical knapsack cover cuts. Then, different lifting procedures are designed to provide cuts that can also be used on models (A) and (B). We rely on sequence-dependent lifting procedures, which are known to be less powerful than sequence-independent methods. However, this simple strategy allows us to design specialized lifting procedures for STWP by introducing specific constraints in the lifting sub-problems.

Given a fractional solution x^* to the linear relaxation of $(MMKP)$, or a partial solution of (A) or (B), the procedure used to derive our valid inequalities follows these steps derived from [GNS98]:

- **Selection** For each knapsack constraint to be treated, denoted by $\sum_{k \in N} a_k x_k \leq b$, perform the following steps:
- **Initial cover** Try to determine a cover $C \subseteq N$, i.e. a subset of indices of variables such that $\sum_{k \in C} a_k > b$ and $\forall k \in C, x_k^* > 0$.
- **Cover partition** Partition set C into the two subsets $C_2 = \{k | k \in C, x_k^* = 1\}$ and $C_1 = C - C_2$.
- **Lifting** Partition the remaining variables into $F = \{k | k \in N - C, x_k^* > 0\}$ and $R = \{k | k \in N - C, x_k^* = 0\}$. Lift up variables in F, lift down variables in C_2, and lift up variables in R.

The lifting phase consists in determining coefficients $(\alpha_k)_{k \in N-C}$ and $(\gamma_k)_{k \in C_2}$ such that the following inequality is valid (satisfied by all integer solutions of the problem):

$$\sum_{k \in C_1} x_k + \sum_{k \in N-C} \alpha_k x_k + \sum_{k \in C_2} \gamma_k x_k \leq \beta + \sum_{k \in C_2} \gamma_k \qquad (13)$$

where β is equal to the maximum number of variables in C_1 that can be set to 1 if all variables in C_2 are set to 1. Up-lifting is used to strengthen the cover inequality: given a valid inequality and an index of variable j, it consists in finding the largest coefficient α_j that can multiply variable x_j while keeping a valid inequality ($\alpha_j = 0$ suffices for validity). Down-lifting is used to ensure validity ($\gamma_j = 0$ does not yield a valid inequality).

As described in Section 4.2.3, the lifting step consists in sequentially solving optimization problems. Classical lifting procedures for knapsack cover inequalities are based on a single knapsack constraint, and lifting one coefficient corresponds to solving one knapsack problem. The first specialized lifting procedure that we propose integrates constraints on time windows of the jobs and can be performed in polynomial time. The second procedure introduces group constraints and constraints from Proposition 4 in order to obtain stronger lifted coefficients. Both lifting problems can be considered scheduling problems.

4.2.1. Lifting problem with time windows

Let us introduce the optimization problem $LIFT_1$, whose instances are characterized by the following:

- A set $\Theta = \{\theta_1, \ldots, \theta_u\}$ of u jobs;
- A subset of mandatory jobs $\Phi \subseteq \Theta$;
- For each job $\theta_i \in \Theta$, the following integers are given: a release date r_i, a deadline $\bar{d}_i$, a weight ω_i, and a processing time p_i. In addition, jobs in Θ are ordered according to Proposition 1.

The lifting problem with time windows, $LIFT_1$, is to determine the maximum weight of a subset of jobs in Θ that can be processed on a single machine with respect to the given order, provided that all jobs in Φ are selected. Appendix A presents an $\mathcal{O}(u \sum_{i=1}^{u} \omega_i)$-time algorithm to solve this problem to optimality.

4.2.2. Lifting problem with time windows, groups of jobs, and dominance of actual occurrences over virtual occurrences

An instance of the problem $LIFT_2$ is defined by the following:

- A set $\Theta = \{\theta_1, \ldots, \theta_u\}$ of u jobs;
- $|\pi|$ disjoint subsets of Θ (groups of jobs) $\{\pi_1, \ldots, \pi_{|\pi|}\}$;
- A subset of mandatory jobs $\Phi \subseteq \Theta$;
- An integer χ related to the considered constraints from Proposition 4 (see below);
- For each job $\theta_i \in \Theta$, the following integers are given: a release date r_i, a deadline $\bar{d}_i$, a weight ω_i, a processing time p_i, a dominance constraint identifier $\lambda_i \in \{-\chi, \ldots, \chi\}$, and a group identifier $g_i = g$ if $\theta_i \in \pi_g$, $g_i = \emptyset$ if $\nexists g | \theta_i \in \pi_g$. In addition, jobs in Θ are ordered according to Proposition 1.

The lifting problem with time windows, groups of jobs, and dominance of actual occurrences over virtual occurrences, $LIFT_2$, is to determine the maximum weight of a subset of jobs in Θ that can be processed on a single machine with respect to the given sequence with the following constraints: all jobs in Φ are selected, and at most one job is selected in each set $\pi_g \in \pi$. In addition for any selected job $\theta_i \in \Theta$ such that $\lambda_i < 0$, the job θ_j such that $j = \min\{k | \lambda_k = -\lambda_i, k > i\}$ is selected. Note that this last constraint allows for the expression of the dominance property from Proposition 4.

Appendix B describes a dynamic program, running in $\mathcal{O}(u \cdot \sum \omega_i \cdot 2^{|\pi|+\chi})$, to solve this problem. Proposition 7 shows that, in some cases, the number χ can be drastically reduced while keeping an equivalent problem.

Proposition 7. *For all $\lambda \in \{1, \ldots, \chi\}$, define*

$$E(\lambda) = \big\{[i,j] | i \in \{1, \ldots, u\}, \lambda_i = -\lambda, j = \min\{k | \lambda_k = \lambda, k > i\}\big\}.$$

Consider two instances, Ω and Ω', of $LIFT_2$. For notational convenience, we add a Ω (resp. Ω') subscript to the data of instance Ω (resp. Ω'). Let $\lambda \in \{1, \ldots, \chi\}$ and $\lambda' \in \{1, \ldots, \chi\} - \{\lambda\}$. Assume that instance Ω' is obtained from Ω by resetting, for all $i \in \{1, \ldots, u\}$, $\lambda_{\Omega',i} = \lambda$ if $\lambda_{\Omega,i} = \lambda'$ and $\lambda_{\Omega',i} = -\lambda$ if $\lambda_{\Omega,i} = -\lambda'$.

If for all $(e, e') \in E_\Omega(\lambda) \times E_\Omega(\lambda')$, $e \cap e' = \emptyset$, then Ω and Ω' have the same set of feasible solutions.

Proof. First, we observe a one-to-one correspondence for all $l \in \{1, \ldots, \chi\}$ between, on the one hand, the set of pairs of jobs (θ_i, θ_j), such that θ_j is scheduled if θ_i is scheduled and $\lambda_j = l$, and on the other hand, the set of intervals $E_\Omega(l)$. Thus, there is a one-to-one correspondence between the set of all pairs of jobs (θ_i, θ_j) such that θ_j is scheduled if θ_i is scheduled and the set of all intervals $\cup_{l=1}^{\chi} E_\Omega(l)$.

Because $e \cap e' = \emptyset$ for all $(e, e') \in E_\Omega(\lambda) \times E_\Omega(\lambda')$, we have

$$\begin{cases} \forall i \in \{1, \dots, u\} | \lambda_{\Omega,i} = -\lambda, & \min\{k | \lambda_{\Omega,k} = \lambda, k > i\} \\ & \quad = \min\{k | (\lambda_{\Omega,k} = \lambda \vee \lambda_{\Omega,k} = \lambda') \wedge k > i\} \\ \forall i \in \{1, \dots, u\} | \lambda_{\Omega,i} = -\lambda', & \min\{k | \lambda_{\Omega,k} = \lambda', k > i\} \\ & \quad = \min\{k | (\lambda_{\Omega,k} = \lambda \vee \lambda_{\Omega,k} = \lambda') \wedge k > i\} \end{cases}$$

Thus,

$$\begin{cases} \forall i \in \{1, \dots, u\} | \lambda_{\Omega,i} = -\lambda, & \min\{k | \lambda_{\Omega,k} = \lambda, k > i\} = \min\{k | \lambda_{\Omega',k} = \lambda, k > i\} \\ \forall i \in \{1, \dots, u\} | \lambda_{\Omega,i} = -\lambda', & \min\{k | \lambda_{\Omega,k} = \lambda', k > i\} = \min\{k | \lambda_{\Omega',k} = \lambda, k > i\} \end{cases}$$

Therefore, $E_{\Omega'}(\lambda) = E_\Omega(\lambda) \cup E_\Omega(\lambda')$, and $E_{\Omega'}(\lambda') = \emptyset$. Then, the constraints of both instances are the same. □

Proposition 7 is particularly useful when all jobs in STWP have a different deadline: in this case, all virtual jobs that are created to represent other jobs scheduled before job J_k are placed consecutively and immediately before J_k in the order given by Proposition 1. It follows that all dominance from Proposition 4 can be encoded by setting $\lambda_{\sigma^m(k)} = 1$ and setting $\lambda_{\sigma^m(k-l)} = -1$ for all l virtual jobs linked with J_k, since we know that they are dominated by the next actual job. In this case, $\chi = 1$ is sufficient to encode all dominance constraints. Note that, in general, choosing χ as the maximum number of actual jobs that share the same deadline is sufficient, which considerably reduces the memory space requirement to solve the lifting problem.

4.2.3. Overall procedure to derive specialized lifted cover inequalities

The procedure takes as input a fractional solution x^* to the linear relaxation of $(MMKP)$, or a partial solution of (A) or (B), and builds a valid inequality. For the sake of readability, let $J^m_k = J_{\sigma^m(k)}$ denote the k^{th} job that can be processed on machine m according to Proposition 1.

Selection and initial cover. First, choose a knapsack constraint in the $(MMKP)$ model that would be violated if each component of x^* was rounded up. On a machine m, this determines a sequence μ of q jobs $(J^m_{\mu(1)}, \dots, J^m_{\mu(q)})$ that cannot fit in the time window $[r^m_{\mu(1)}, \bar{d}^m_{\mu(q)}]$ (which defines the initial cover $C = \{\mu(i) | i = 1 \dots, q\}$). With $N = [\mu(1), \mu(q)] \cap \mathbb{N}$, the knapsack constraint is

$$r^m_{\mu(1)} x^m_{\mu(1)} + \sum_{k \in N} p^m_k x^m_k \leq \bar{d}^m_{\mu(q)} \tag{14}$$

Cover partition. Set $C_2 = \{k | k \in C, x^*_k = 1\}$ and $C_1 = C - C_2$. Then suppose that the jobs corresponding to C_2 are actually selected. Under this assumption, let us compute β. This requires us to answer the question Q_1: "What maximum number β of jobs in C_1 is it possible to process together with all jobs in C_2?". To this end, we solve an instance of $LIFT_2$ defined by the following:

- $\Theta = \{J_i | i \in C_1 \cup C_2\}$, and release dates, deadlines, and processing times are the same as in the instance of STWP;
- $\Phi = \{J_i | i \in C_2\}$;
- $\omega_i = 1$ if $i \in C_1$, otherwise $\omega_i = 0$.

Moreover, all instances of $LIFT_2$ that we have to solve during the construction of the inequality share the following data:

- $\pi = \{G'_1 \cap \Theta, \dots, G'_{|\pi|} \cap \Theta\}$;
- $\lambda_i = \begin{cases} i & \text{if } J_i \in I \text{ and } \Theta \cap H_i \neq \emptyset \\ -h^{-1}(i) & \text{if } h^{-1}(i) \neq \emptyset \text{ and } J_{h^{-1}(i)} \in \Theta \\ 0 & \text{otherwise} \end{cases}, J_i \in \Theta;$
- $\chi = \max_i \{\lambda_i\}$.

Note that, once this instance is defined, it is possible to modify vector λ according to Proposition 7 to obtain a minimal value of χ. By definition of $LIFT_2$, solving this instance answers Q_1, taking into account a relaxation of constraints from STWP. Supposing that the jobs corresponding to C_2 are actually selected, the following inequality holds for any feasible solution:

$$\sum_{k \in C_1} x_k^m \leq \beta \tag{15}$$

Up-lifting. Still assuming that the jobs corresponding to C_2 are all processed (i.e. $\forall k \in C_2, x_k^m = 1$), let us suppose that coefficients $(\alpha_k)_{k \in F}$ and $(\gamma_k)_{k \in C_2}$ are determined such that the following inequality holds:

$$\sum_{k \in C_1} x_k^m + \sum_{k \in F \cup R} \alpha_k x_k^m + \sum_{k \in C_2} \gamma_k x_k^m \leq \beta + \sum_{k \in C_2} \gamma_k \tag{16}$$

Because (15) holds, valid initial values are $\alpha_k = 0$, $k \in F \cup R$ and $\gamma_k = 0$, $k \in C_2$. Then, given a sequence of variables in F, we sequentially find a (as large as possible) coefficient α_k for each variable x_k^m, $k \in F$, such that (16) holds. For a given variable x_j^m, $j \in F$, this requires answering the question Q_2: "What is the maximum possible value z of the left-hand-side of (16), provided that $\forall k \in C_2, x_k^m = 1$, $\alpha_j = 0$, and $x_j^m = 1$?", which is equivalent to Q_2: "What is the maximum possible weight z of jobs in $C_1 \cup F - \{J_j^m\}$ that can be processed together with J_j^m and all jobs in C_2?", for which the weights of jobs are defined as their coefficient in (16). Clearly, z satisfies $z + \alpha_j x_j^m \leq \beta$ in any feasible solution where J_j^m and jobs in C_2 are processed. Thus, resetting $\alpha_j = \beta - z$ in (16) yields an inequality that is valid in all feasible solutions where all jobs in C_2 are processed. In order to answer Q_2, we solve the following instance of $LIFT_2$: $\Theta = \{J_i | i \in C_1 \cup C_2 \cup F\}$, $\Phi = \{J_i | i \in C_2\} \cup \{J_j\}$, $\omega_i = 1$ if $i \in C_1$, $\omega_i = \alpha_i$ if $i \in F - \{J_j\}$, and otherwise $\omega_i = 0$.

Down-lifting. When all variables in F are lifted up, we obtain an inequality that is valid under the assumption that all jobs corresponding to C_2 are processed. In order to dispense with this hypothesis and make the inequality valid for the whole problem, coefficients of variables in C_2 must be lifted down. Given a sequence of variables in C_2, let L denote the set of indices of variables that have already been down-lifted. At this stage, we have obtained an inequality of the form (16), which is valid under the assumption that all jobs in $C_2 - L$ are processed (i.e. $\forall k \in C_2 - L, x_k^m = 1$). For the next variable x_j^m such that $j \in C_2 - L$, our aim is to find a coefficient γ_j such that (16) holds, under the assumption that $x_j^m = 0$ and $\forall k \in C_2 - L - \{J_j^m\}, x_k^m = 1$. Note that, if $x_j^m = 1$, any value for γ_j is suitable, so that the resulting inequality will be valid whatever the value of x_j^m. The question to be answered here is Q_3: "What is the maximum possible value z of the left-hand-side of (16), given that $x_j^m = 0$ and $\forall k \in C_2 - L - \{J_j^m\}, x_k^m = 1$?", or equivalently Q_3: "Supposing that J_j^m is not processed, what is the maximum weight z of jobs with indices in $C_1 \cup F$ that can be processed together with all jobs with indices in $C_2 - L - \{j\}$?". By construction, choosing $\gamma_j = z - (\beta + \sum_{k\in L} \gamma_k)$ is valid. In order to answer Q_3, we solve the following instance of $LIFT_2$: $\Theta = \{J_i | i \in C_1 \cup C_2 \cup F - \{j\}\}$, $\Phi = \{J_i | i \in C_2 - L - \{j\}\}$, $\omega_i = 1$ if $i \in C_1$, $\omega_i = \alpha_i$ if $i \in F$, $\omega_i = \gamma_i$ if $i \in L$, and $\omega_i = 0$ otherwise. Finally, in order to lift up the coefficients of variables x_j^m, $j \in R$, the following instance of $LIFT_2$ is solved: $\Theta = \{J_i | i \in C_1 \cup C_2 \cup F \cup R\}$, $\Phi = \{J_j\}$, $\omega_i = 1$ if $i \in C_1$, $\omega_i = \alpha_i$ if $i \in F \cup R - \{J_j\}$, $\omega_i = \gamma_i$ if $i \in C_2$, and $\omega_i = 0$ otherwise.

Note that, because the constraints of $LIFT_2$ consist of a relaxation of the constraints of STWP, β and z may be over-estimated. In this case, coefficients α_j, $j \in F \cup R$, may be under-estimated and coefficients γ_j, $j \in C_2$ may be over-estimated, so that the resulting inequality is also valid. Moreover, $LIFT_1$ is clearly either a relaxation of $LIFT_2$, so that it is possible to solve instances of this simpler problem to compute (16), or it is a relaxation of $LIFT_2$ containing only a subset of group and dominance constraints.

Observe that $\forall \theta_k \in F$, $\alpha_k \leq \beta \leq n_{I'}$. It follows that $\forall k \in C_2$, $\gamma_k \leq n_{I'}^2$, and $\forall k \in R$, $\alpha_k \leq n_{I'}^2$. Therefore, for each instance of the lifting problem, $\sum_{i=1}^{u} \omega_i \leq n_{I'}^3$. Finally, computing all lifted coefficients for a given inequality takes $\mathcal{O}(n_{I'}^5)$ operations when using $LIFT_1$ and $\mathcal{O}(n_{I'}^5 \cdot 2^{|\pi|+\chi})$ operations when using $LIFT_2$.

5. Application to the problem of minimizing the number of late jobs

In this section, we show how instances of late job minimization problems can be converted into instances of STWP.

5.1. $1, h_k | r_i, nr - a | \sum w_i U_i$

Let us consider an instance Ω of $1, h_k | r_i, nr - a | \sum w_i U_i$ and let us build an equivalent instance $\tilde{\Omega}$ of STWP. In the remainder of this paper, for notational convenience, we use a tilde (˜) sign to refer to data of the instance denoted by

$\tilde{\Omega}$. The idea behind 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 considering that each availability period corresponds to an independent machine on which jobs can be processed in parallel. Formally, to build an equivalent instance $\tilde{\Omega}$ to Ω, we create $K+2$ machines. For each job J_i in Ω, we create a group of jobs composed of one job $\tilde{J}_{i,s}$ for each unavailability period s where J_i can be processed, plus one fictitious late job: $\tilde{G}_i = \{\tilde{J}_{i,s} | 1 \leq s \leq K+1, B_s \geq r_i + p_i, F_{s-1} + p_i \leq d_i\} \cup \{\tilde{J}_{i,*}\}$. The late job $\tilde{J}_{i,*}$ is such that $\tilde{r}_{i,*} = 0$, $\tilde{d}_{i,*} = 1$, $\tilde{p}_{i,*} = 0$, $\tilde{w}_{i,*} = w_i$, and $\tilde{m}_{i,*} = K+2$. The release and due dates of other jobs are defined as $\tilde{r}_{i,s} = \max(r_i, F_{s-1}) - F_{s-1}$ and $\tilde{d}_{i,s} = \min(d_i, B_s) - F_{s-1}$. Finally, $\tilde{p}_{i,s} = p_i$, $\tilde{w}_{i,s} = 0$, and $\tilde{m}_{i,s} = s$.

5.2. $1, h_k | r_i, ST_{si} | \sum w_i U_i$

The transformation proposed in this section is based on the idea that unavailability periods have two consequences for jobs: First, no job can start during one of them. Second, the completion of a job that crosses one or several unavailability periods is postponed for a deterministic, foreseeable amount of time. Let us introduce the following remarks.

Remark 1. For any job $J_i \in I$, for any time instant $t \geq r_i$, the completion time of J_i starting at t is

$$C_i(t) = \begin{cases} \infty \text{ if } \exists s \in \{1, \dots, K\}, B_s \leq t < F_s \\ \min\{\beta(t,i,s) | s \leq K, \beta(t,i,s) \leq B_{s+1}\} \text{ otherwise} \end{cases}$$

where

$$\beta(t,i,s) = t + p_i + \sum_{s'=s(t)+1}^{s} \big(F_{s'} - B_{s'} + \min(B_{s'+1} - F_{s'}, \alpha_i)\big)$$

and $s(t) = \arg\max\{s | B_s < t\}$.

We can verify that $\beta(t,i,s)$ is the completion time of a job J_i starting at t if we take into account the unavailability periods up to period s. The actual completion time of J_i is related to the earliest unavailability period that does not postpone the end of J_i. Hence, the time during which the machine will be unavailable for processing another job if J_i starts at t is $\bar{f}_i(t) = C_i(t) - t$. Taking this variable duration as the processing time of J_i allows us to implicitly account for the machine's unavailability periods. Indeed, it ensures that, in any feasible solution, no job starts during an unavailability period (or there is exactly one such job, which is late and can be removed without changing the cost of the schedule). Moreover, the completion time of each job will be correctly postponed. The following dominance property reduces the set of possible starting times for each job.

Remark 2. There is at least one optimal solution such that, for all $J_i \in I$ and all $s \in \{1, \dots, K\}$, J_i does not start in $[B_s - \min\{\alpha_i, B_{s+1} - F_s, p_i - 1\}, F_s]$.

Intuitively, if one such job exists in an optimal solution, it starts before B_s and causes a setup time after F_s. Delaying the job to start at F_s removes the setup time and does not result in the job completing later. The following remark states that the duration for which the machine is occupied by each job is a stepwise function of the job's starting time.

Remark 3. Let $J_i \in I$ and $t \geq 0$ such that $F_{s_1-1} \leq t < B_{s_1} - \min\{\alpha_i, B_{s_1+1} - F_{s_1}, p_i - 1\}$ and $F_{s_2-1} \leq C_i(t) \leq B_{s_2}$. Then, for all t' such that $F_{s_1-1} \leq t' < B_{s_1} - \min\{\alpha_i, B_{s_1+1} - F_{s_1}, p_i - 1\}$ and $F_{s_2-1} \leq C_i(t') \leq B_{s_2}$, $\bar{f}_i(t) = \bar{f}_i(t')$.

A formal proof would include showing that J_i crosses the same unavailability periods when it starts at t as at t' and therefore is subject to the same setup times. The following remark allows us to consider a problem without unavailability constraints, for the reasons given above. Note that the value of $\bar{f}_i(t)$ changes when either t or $C(t)$ jumps to another availability period. Moreover, when t jumps to a later availability period, cases where $C(t)$ jumps to an earlier period can be discarded thanks to Remark 2. Thus, at most $2K + 1$ intervals are required to define $\bar{f}_i(t)$.

Remark 4. For each instance Ω of $1, h_k | r_i, ST_{si} | \sum w_i U_i$, we can build an equivalent instance Ω' of $1 | r_i, p_i(t) = \bar{f}_i(t) | \sum w_i U_i$, such that $|I'| = n$ and for all $J_i \in I$, we create a job $J'_i \in I'$ such that $r'_i = r_i$, $d'_i = d_i$, $w'_i = w_i$, and with a variable processing time $p'_i(t) = \bar{f}_i(t) = C_i(t) - t$.

Note that the only events that can occur during an unavailability period are release or due dates, which can be shifted to the beginning of the corresponding period without consequences. Therefore, contracting the horizon during unavailability periods, i.e. considering that no time elapses during them, yields an equivalent problem. Let us define the following functions:

$$\begin{aligned} \Delta(t) = & \begin{cases} t - \sum_{s=1}^{s(t)} (F_s - B_s) & \text{if } F_{s(t)} < t \\ B_{s(t)} - \sum_{s=1}^{s(t)} (F_s - B_s) & \text{otherwise} \end{cases} \\ \Delta^+(t) = & \min \left\{ t + \sum_{s'=1}^{s} (F_{s'} - B_{s'}) | s \leq K, t + \sum_{s'=1}^{s} (F_{s'} - B_{s'}) \leq B_{s+1} \right\} \end{aligned}$$

Function Δ aims to shift a time instant from a problem with unavailability periods to a problem without unavailability periods by removing the unavailability periods and considering that no time elapses during them. In its definition, the first case applies when t lies between two unavailability periods, whereas the second case concerns situations where t falls during an unavailability period. Function Δ^+ can be considered as the inverse function of Δ, defined for t values that lie outside of the unavailability periods: $\Delta^+(\Delta(t)) = t$ if $F_{s(t)} < t$.

Proposition 8. *Let Ω'' be an instance of $1 | r_i, p_i(t) = \bar{f}_i(t) | \sum w_i U_i$ obtained from Ω' by resetting, for all jobs $J''_i \in I''$, $r''_i = \Delta(r'_i)$, $d''_i = \Delta(d'_i)$, and $p''_i(t) = \Delta(C_i(\Delta^+(t))) - t$. Then, there exists a feasible schedule for Ω' if and only if there exists a feasible schedule for Ω'' with the same cost.*

PROOF. Consider a feasible schedule S' of instance Ω', and let t'_i denote the starting time of $J'_i \in I'$. Let t''_i denote the starting time of $J''_i \in I''$ in a schedule S'' defined by $t''_i = \Delta(t'_i)$ for all $J'_i \in I'$. We show that S'' is feasible for Ω''. First, Δ is a non-decreasing function, so $t'_i \geq r'_i \leftrightarrow t''_i \geq r''_i$. Therefore, S'' satisfies the release date constraints.

Moreover, note that for all $J'_i \in I'$, $F_{s(t'_i)} < t'_i$. So $p''_i(\Delta(t'_i)) = \Delta(C_i(\Delta^+(\Delta(t'_i)))) - \Delta(t'_i) = \Delta(C_i(t'_i)) - \Delta(t'_i)$. It follows that $\Delta(C_i(t'_i)) = t''_i + p''_i(t''_i)$.

Because S' is feasible, $C_i(t'_i) \leq d'_i$, equivalent to $\Delta(C_i(t'_i)) \leq d''_i$, or $t''_i + p''_i(t''_i) \leq d''_i$. Therefore, S'' satisfies the deadline constraints.

Finally, for all $J'_j \in I' - \{J'_i\}$ such that J'_j is scheduled after J'_i in S', we have $t'_j \geq C_i(t'_i)$. This is equivalent to $\Delta(t'_j) \geq \Delta(C_i(t'_i))$ or $t''_j \geq t''_i + p''_i(t''_i)$. Thus, S'' satisfies the resource constraints, and S'' is feasible and evidently has the same cost as S'. □

The principle of the transformation from an instance Ω' of $1|r_i, p_i(t) = \bar{f}_i(t)|\sum w_i U_i$ to an instance $(\tilde{\Omega})$ of $STWP$ is to create one job in $(\tilde{\Omega})$ for each job in Ω' and each interval for which the processing time of the job is constant. Jobs of $(\tilde{\Omega})$ associated with the same job of Ω' are placed in the same group together. Formally, for each job $J'_i \in I'$, we denote

$$\begin{aligned}\Xi_i = &\{(t_1, t_2, s_1, s_2) | t_1 \in \{r'_i, \ldots, d'_i\}\\ &s_1 \in \{1, \ldots, K+1\}, s_2 \in \{1, \ldots, K+1\},\\ &F_{s_1-1} \leq t_1 \leq B_{s_1} - \min\{\alpha_i, B_{s_1+1} - F_{s_1}, p_i - 1\},\\ &F_{s_2-1} \leq t_2 \leq \min(B_{s_2}, d'_i)\}\end{aligned}$$

For each job $J'_i \in I'$, we define a group of jobs $\tilde{G}_i = \{\tilde{J}_{i,s_1,s_2} | \exists t \in \{r'_i, \ldots, d'_i\}, (t, C_i(t), s_1, s_2) \in \Xi_i\}$. Job $\tilde{J}_{i,s_1,s_2}$ corresponds to the execution of J_i such that it starts between unavailability periods $s_1 - 1$ and s_1 and completes between unavailability periods $s_2 - 1$ and s_2. This job has the following characteristics: $\tilde{r}_{i,s_1,s_2} = \min\{t | t \in \{r'_i, \ldots, d'_i\}, (t, C_i(t), s_1, s_2) \in \Xi_i\}$, $\tilde{d}_{i,s_1,s_2} = \max\{C_i(t) | t \in \{r'_i, \ldots, d'_i\}, (t, C_i(t), s_1, s_2) \in \Xi_i\}$, $\tilde{p}_{i,s_1,s_2} = \Delta(C_i(\Delta^+(\tilde{r}_{i,s_1,s_2}))) - \tilde{r}_{i,s_1,s_2}$, and $\tilde{w}_i = w_i$ for all $i \in \{1, \ldots, n\}$. Note that it is possible to determine the set of jobs $\tilde{I}$ in $\mathcal{O}(nK)$ operations thanks to Remarks 2 and 3.

Remark 5. Problem $1|r_i, r{-}a|\sum w_i U_i$ is a special case of $1, h_k|r_i, ST_{si}|\sum w_i U_i$, where $\forall J_i \in I$, $\alpha_i = 0$. By intuition, it is sufficient to remove all unavailability periods and shift all dates accordingly to obtain an equivalent instance of problem $1|r_i|\sum w_i U_i$. Moreover, this transformation can be performed in polynomial time and preserves any non-strict order on release and due dates, which therefore implies the following complexity results: $1, h_k|r - a|\sum U_i$ is polynomial [Moo68], $1, h_k|r - a|\sum w_i U_i$ is NP-hard in the ordinary sense [Kar72], $1, h_k|r_i, r - a|\sum U_i$ is polynomial when release and due dates are similarly ordered [KIM78], $1, h_k|r - a|\sum w_i U_i$ is polynomial when processing times and job weights are oppositely ordered [Law94]...

6. Numerical experiments

This section reports the computational results we obtained by solving the problem through the transformations described in this paper with the help of the commercial MILP solver IBM ILOG Cplex v12.4 using default settings on a personal computer (PC) equipped with a 3.2 GHz quad-core processor and 3 GB RAM and running the Windows Seven 64-bit operating system.

6.1. Test bed

Based on Remark 5, problem $1, h_k|r_i, r{-}a|\sum w_i U_i$ is equivalent to $1|r_i|\sum w_i U_i$. Thus, we test our solving method by generating instances for these problems as described in [DPS02]. 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, whereas parameter D controls the size of the job execution windows. The parameters used to build our test bed are the combinations of $n \in \{200, 250, 300, 350, 400, 450, 500\}$, $R \in \{1, 5, 10, 20\}$, and $D \in \{1, 5, 10, 20\}$. Ten instances are generated for each combination of these parameters, leading to a total of 1120 instances. In order to generate a test bed for $1, h_k|r_i, nr - a|\sum w_i U_i$, we introduce 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 $n \in \{200, 300, 400, 500\}$, $R \in \{1, 5, 10, 20\}$, $D \in \{1, 5, 10, 20\}$, $K \in \{1, 3, 5\}$, and $UR \in \{1, 5, 10\}$. Five instances are generated for each combination of parameters, leading to a total of 2880 instances. This test bed is used for problem $1, h_k|r_i, ST_{si}|\sum w_i U_i$, and we define for all jobs $J_i \in I$, $\alpha_i = \lceil \alpha p_i \rceil$, with $\alpha \in \{0.1, 0.25\}$, deriving 5760 instances.

6.2. Implementation details

It is computationally expensive to calculate the specialized lifted cover cuts (SLCC) described in Section 4.2. The procedure therefore uses several parameters that we fixed according to preliminary tests. In our experiments, (SLCC) are applied at root node only and through several passes. At each pass, only cuts based on an initial cover $\{\mu(i)|i \in \{1, \ldots, q\}\}$ with less than q_{max} variables are generated. We vary q_{max} in $\{3, 5, 7, 9, 14, \infty\}$ in increasing order to generate cuts with few variables first. Only knapsack constraints such that $\mu(q) - \mu(1) \leq 100$ are examined. Each pass consists of a loop that ends when no cut is added to the model. Each turn of the loop starts by computing the linear relaxation of the current MILP model (B), which yields a fractional solution, and then determines an initial cover by scanning the variables in increasing order of their machines and indices. Then, the set N of indices of variables is extended to $[\mu(1) - 50, \mu(q) + 50] \cap \mathbb{N}$. Although some of these variables do not belong to the chosen knapsack constraint, they can be integrated to the lifted cover inequality because the lifting problems do not rely on this constraint only. In order to elaborate the lifting sequence, variables in F are sorted according to two criteria: variables in $[\mu(1), \mu(q)] \cap F$ are lifted first, and then variables in $([\mu(1) - 50, \mu(1)[\cup]\mu(q), \mu(q) + 50]) \cap F$ are processed. Inside each group, variables with the highest value in the current solution are lifted first. Variables in C_2 are lifted according to a non-decreasing order of their reduced cost. A similar rule is used for variables in R, except that variables in $[\mu(1), \mu(q)] \cap R$ are lifted first. If the current solution violates the valid inequality, it is added to the model and the loop is restarted. Otherwise, we look for the next knapsack constraint.

When $LIFT_2$ is used to compute the lifting coefficients, only a subset of $|\pi|$ group constraints is taken into account. At the beginning of the lifting process for a given valid inequality and for each group of jobs of the current solution, we sum the values of variables that are in the group and in N. Throughout the lifting procedure, we retain only the $|\pi|$ groups with the largest cumulated values. Also, the value of χ is limited. For each inequality being constructed, we first assign a suitable value of parameter $\lambda_i \in \{-\chi, \dots, \chi\}$ for all jobs $\theta_i \in \Theta$. For this purpose, we use a simple greedy algorithm, which proceeds from the first variable to the last, storing the set of unused possible values. Each time it encounters a dominated variable, it assigns it the first unused value, if there is one available, and sets this value as used. If no value is available, it assigns 0, which means that the corresponding constraint will not be taken into account for this valid inequality. When it encounters a dominating variable such that at least one of the linked dominated variables received a non-zero value, its opposite is assigned, and the value is reset as unused. In order to control memory requirements, we abort the search for an optimal solution of $LIFT_2$ if the current maximal value exceeds $n_I/8$, i.e. one eighth of the overall number of jobs. The value of χ is passed to the method as a fixed parameter. The value $|\pi|$ is calculated from n_I and χ so that the size of the dynamic program cannot exceed 3 GB. A time limit of 360 seconds is imposed for the generation of all SLCC. Because there can be many of them, the number of constraints issued from Proposition 3 is limited to 10000.

6.3. Results

Table 1 reports results we obtained when solving different types of problems by applying the MILP solver to Model (B) through the transformation proposed in this paper and using all described improvements except SLCC. This approach proves to be very efficient in spite of its genericity, because all 250-job instances of $1|r_i|\sum w_iU_i$ are solved to optimality within one hour. Also, more than 96% of 500-job instances are solved within this time limit for the three types of problems we have considered. Model (A), used with the same improvements and under the same benchmarking conditions, leaves open 28 instances of $1|r_i|\sum w_iU_i$. In addition, this model is very sensitive to parameter R due to the structure of Constraints (3): 27 of them are generated with the combination of parameters $R = 20$ and $D = 1$. For the problem $1, h_k|r_i, nr-a|\sum w_iU_i$, 49 instances are left open using this first model. For the case of semi-resumable jobs, 55 300-job instances cannot be solved within one hour, which is five more times than for Model (B). This is explained by the large release dates generated for the jobs in (STWP) for this problem. Because of these premiliminary results, we did not evaluate Model (A) for the whole test bed of $1|r_i, ST_{si}|\sum w_iU_i$. As expected, Model (B) yields a stronger linear relaxation, which also requires about half the computing time. Increasing parameter K makes the two other problems harder to solve: 99.3% vs. 97.1% (resp. 99.2% vs. 97.7%) of the instances of $1|r_i, ST_{si}|\sum w_iU_i$ (resp. of $1|r_i, nr-a|\sum w_iU_i$) are closed when K is increased from $K = 1$ to $K = 5$. This is easily explained by the increase in the number of jobs in STWP. Increasing parameter UR has a slighter, opposite effect, which can be explained by the fact that the more the machine is unavailable, the fewer feasible solutions exist.

For the sake of comparison, we tested the method described in [TF12] on our set of instances of problem $1|r_i|\sum w_iU_i$. This algorithm solves to optimality 100 out of 160 200-job instances. The average computing time for instances solved to optimality is 415.7 seconds, compared with only 10.8 seconds using our method. Let us recall that the work of [TF12] applies to generic total cost single machine problems, which explains these modest results on our specific problem. To the best of our knowledge, the best dedicated method for solving $1|r_i|\sum w_iU_i$ has been published in [MB07], where the authors report success for instances with up to 200 jobs. Although their algorithm is likely to perform better with the computer power available today, it is based on a MILP formulation that cannot handle machine availability constraints. Because our method is theoretically suitable for multiple machine problems, we applied it to solving the multi-machine assignment and scheduling problem [SW06]. The test bed proposed by the authors is composed of 36 instances counting up to 54 jobs to be assigned to nine machines, and their method, based on constraint programming and integer programming cooperation, solves all of them in less than one hour. The use of Model (B) is not as efficient, given that only 15 instances are solved on our faster computer. By using $(SLCC)$, only 4 more instances can be solved. These degraded results suggest that our model is good at sequencing jobs once they are assigned to machines, but it is not very adapted to the assignment component of the problem.

In order to evaluate the pertinence of the specialized lifting cover cuts, we apply the procedures to solving the hardest instances of our test bed. We select instances which cannot be solved within 1000 seconds (corresponding to the column "#unsolved 1000 sec." in Table 1) using model (B) alone to test $LIFT_1$ and $LIFT_2$ with $\chi \in \{0, 1, 2\}$. The polynomial-time procedure $LIFT_1$ allows us to close all 200-job instances of problem $1|r_i, ST_{si}|\sum w_iU_i$ and, compared with other methods, seems to perform well for 450 and 500-job instances of $1|r_i|\sum w_iU_i$. (Although we remain cautiously optimistic concerning these results due to the small number of instances concerned.) Procedure $LIFT_2$

Problem type		Model (B)					Model (A)	
#jobs	#instances	#unsolved 1000 sec.	#unsolved 1 h.	Avg. time (sec.)	Avg. LR time (sec.)	Avg. LR gap	Avg. LR time (sec.)	Avg. LR gap
$1\|r_i\|\sum w_iU_i$								
200	160	0	0	11.6	1.9	1.40%	2.3	1.64%
250	160	2	0	50.7	3.3	1.11%	4.3	1.24%
300	160	1	1	46.9	5.2	0.85%	7.1	0.99%
350	160	2	1	69.1	7.9	0.67%	11.5	0.75%
400	160	7	5	89.9	12.0	0.59%	18.0	0.69%
450	160	4	3	143.9	17.1	0.48%	25.6	0.54%
500	160	7	5	181.8	23.0	0.45%	38.6	0.50%
$1\|r_i, nr-a\|\sum w_iU_i$								
200	720	0	0	7.7	0.9	1.24%	1.2	1.36%
300	720	7	5	32.7	2.8	0.78%	5.0	0.86%
400	720	16	10	66.7	5.9	0.52%	11.4	0.56%
500	720	34	21	124.5	11.6	0.38%	24.9	0.41%
$1\|r_i, ST_{si}\|\sum w_iU_i$								
200	1440	5	8	11.9	2.3	1.31%	3.0	1.56%
300	1440	11	24	51.7	7.5	0.81%	14.7	0.94%
400	1440	27	51	92.3	17.6	0.53%	38.9	0.63%
500	1440	55	75	120.1	34.8	0.40%	74.7	0.45%

Table 1: Results without (SLCC). Computing time for linear relaxation (LR) and average gap between LR and the best known integer solution are reported for Models (A) and (B). The numbers of instances solved to optimality within 1000 seconds and within one hour are summarized for Model (B). The average time is calculated only on instances solved to optimality within 1 hour.

proves to be powerful, because it allows us to close all instances of our test bed with up to 350 jobs for problem $1|r_i|\sum w_iU_i$. Only one (resp. four) 400-job instance (resp. 500-job instances) is left open with parameter $\chi = 0$, and only three 500-job instances remain open with $\chi = 1$. This setting permits us to solve all 400-job instances of problem $1|r_i, nr-a|\sum w_iU_i$; one 300-job instance is left open. It seems preferable to use parameter $\chi = 1$ only for problem $1|r_i, nr-a|\sum w_iU_i$, and larger values do not seem to improve the efficiency of the method.

$LIFT_2$ is decidedly better for instances with large release dates: for $1|r_i, ST_{si}|\sum w_iU_i$, the gap improvement at root node is 26.5% on average, 3.0% when $R = 1$, and 33.9% when $R = 20$. It is calculated as $\frac{\text{Root lower bound after cuts - Root lower bound}}{\text{Best known upper bound - Root lower bound}}$. This result is not surprising because the root gap, calculated as $2\frac{\text{Best known upper bound - Root lower bound}}{\text{Best known upper bound + Root lower bound}}$, is 1.7% on average, 0.3% for $R = 1$, and 2.2% for $R = 20$, so that there is little room for improvement when release dates are small. In contrast, the root gap improvement is 35.7% for $D = 1$ and 2.1% for $D = 20$, with the same possible explanation. Other parameters do not seem to clearly affect the performance of SLCC.

Out of the 9760 instances of our test bed, composed of 200 jobs or more, 149 are not solved using model (B) alone. Using $LIFT_2$ with $\chi = 0$ or $\chi = 1$

allows us to close 95 more instances. Finally, 99.4% of the test bed is closed, and 98.4% of the 500-job instances are closed.

7. Conclusion and perspectives

This paper shows that the generic problem STWP can serve to model several scheduling problems, and assist in solving these problems efficiently through the use of a MILP approach. Indeed, the performance of the developed method matches that of dedicated methods for the most well-known problem investigated in this study. Furthermore, the method can be directly adapted to parallel machine problems and cases where different unavailability periods are defined for each job. There is still room for improvement: although it seems to us that integrating dominance properties into the lifting problem is of theoretical interest, it does not appear to be of great help in practice. This might be caused by the naive assignment of the constraint identifiers employed in our study, which could be refined. Additionally, it might be possible to improve the theoretical and practical time-complexity of the lifting procedures. Considerable progress might be achieved by investigating the possibility of sequence-independent lifting procedures that integrate specific STWP constraints. In addition, it is likely that our methods cannot solve some instances, not because the size of the MILP model is too large or because of a poor linear relaxation, but because of the existence of a large number of symmetrical solutions generated by similar machines in the constructed STWP problem. Thus, another means of improvement lies in the development of symmetry breaking techniques for this problem. We hope that it will be possible to extend this work to solve other scheduling problems by embedding more realistic constraints, such as sequence-dependent setup times, precedence constraints, batching features and the like.

References

[ANCK08] Ali Allahverdi, C.T. Ng, T.C.E. Cheng, and Mikhail Y. Kovalyov. A survey of scheduling problems with setup times or costs. *European Journal of Operational Research*, 187(3):985–1032, June 2008.

[Bal75] Egon Balas. Facets of the knapsack polytope. *Mathematical Programming*, 8(1):146–164, 1975.

[BPP03] Philippe Baptiste, Laurent Peridy, and Eric Pinson. A branch and bound to minimize the number of late jobs on a single machine with release time constraints. *European Journal of Operational Research*, 144(1):1–11, January 2003.

[BS09] Philippe Baptiste and Ruslan Sadykov. On scheduling a single machine to minimize a piecewise linear objective function: A compact MIP formulation. *Naval Research Logistics*, 56(6):487–502, 2009.

[Car82] J. Carlier. The one-machine sequencing problem. *European Journal of Operational Research*, 11:42–47, 1982.

[Che09] Wen-Jinn Chen. Minimizing number of tardy jobs on a single machine subject to periodic maintenance. *Omega*, 37(3):591–599, June 2009.

[CJP83] Harlan Crowder, Ellis L. Johnson, and Manfred Padberg. Solving large-scale zero-one linear programming problems. *Operations Research*, 31(5):803–834, January 1983.

[DDY11] Boris Detienne, Stéphane Dauzère-Pérès, and Claude Yugma. Scheduling jobs on parallel machines to minimize a regular step total cost function. *Journal of Scheduling*, 14(6):523–538, November 2011.

[DP95] Stephane Dauzère-Pérès. Minimizing late jobs in the general one machine scheduling problem. *European Journal of Operational Research*, 81(1):134–142, February 1995.

[DPS02] Stephane Dauzère-Pérès and Marc Sevaux. Using lagrangean relaxation to minimize the weighted number of late jobs on a single machine. *Naval Research Logistics*, 50(3):273–288, 2002.

[GJ79] M. R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, 1979.

[GL99] Gregory H. Graves and Chung-Yee Lee. Scheduling maintenance and semiresumable jobs on a single machine. *Naval Research Logistics (NRL)*, 46(7):845–863, 1999.

[GLL$^+$79] R.L. Graham, E.L. Lawler, J.K. Lenstra, A.H.G.Rinnooy Kan, E.L. Johnson P.L. Hammer, and B.H. Korte. Optimization and approximation in deterministic sequencing and scheduling: a survey. In *Proceedings of the Discrete Optimization Symposium*, volume Volume 5, pages 287–326. Elsevier, 1979.

[GNS98] Zonghao Gu, George L. Nemhauser, and Martin W. P. Savelsbergh. Lifted cover inequalities for 0-1 integer programs: Computation. *INFORMS Journal on Computing*, 10(4):427 –437, 1998.

[GNS00] Zonghao Gu, George L. Nemhauser, and Martin W.P. Savelsbergh. Sequence independent lifting in mixed integer programming. *Journal of Combinatorial Optimization*, 4(1):109–129, March 2000.

[HDV12] Navid Hashemian, Claver Diallo, and Béla Vizvàri. Makespan minimization for parallel machines scheduling with multiple availability constraints. *Annals of Operations Research*, pages 1–14, 2012.

[Jac55] J.R. Jackson. Scheduling a production line to minimize maximum tardiness (research report 43). *Management Science Research Project*, University of California, Los Angeles, 1955.

[Kar72] R. M. Karp. Reducibility among combinatorial problems. In R. E. Miller and J. W. Thatcher, editors, *Complexity of Computer Computations*, pages 85–103. Plenum Press, 1972.

[KBF$^+$02] Wieslaw Kubiak, Jacek Blazewicz, Piotr Formanowicz, Joachim Breit, and Gunter Schmidt. Two-machine flow shops with limited machine availability. *European Journal of Operational Research*, 136(3):528–540, February 2002.

[KIM78] Hiroshi Kise, Toshihide Ibaraki, and Hisashi Mine. A solvable case of the One-Machine scheduling problem with ready and due times. *Operations Research*, 26(1):121–126, January 1978.

[KSJH12] Gio Kao, Edward Sewell, Sheldon Jacobson, and Shane Hall. New dominance rules and exploration strategies for the $1|r_i|\sum U_i$ scheduling problem. *Computational Optimization and Applications*, 51(3):1253–1274, 2012.

[Law94] E. L. Lawler. Knapsack-like scheduling problems, the Moore-Hodgson algorithm and the [‘]tower of sets’ property. *Mathematical and Computer Modelling*, 20(2):91–106, 1994.

[LC02] Zhaohui Liu and T.C.Edwin Cheng. Scheduling with job release dates, delivery times and preemption penalties. *Information Processing Letters*, 82(2):107–111, April 2002.

[LC04] Zhaohui Liu and T.C.Edwin Cheng. Minimizing total completion time subject to job release dates and preemption penalties. *Journal of Scheduling*, 7(4):313–327, 2004.

[Lee96] Chung-Yee Lee. Machine scheduling with an availability constraint. *Journal of Global Optimization*, 9(3):395–416, December 1996.

[Lee99] Chung-Yee Lee. Two-machine flowshop scheduling with availability constraints. *European Journal of Operational Research*, 114(2):420–429, April 1999.

[LM89] E. L. Lawler and C. U. Martel. Preemptive scheduling of two uniform machines to minimize the number of late jobs. *Operations Research*, 37(2):314–318, March 1989.

[LW07] Quentin Louveaux and Laurence Wolsey. Lifting, superadditivity, mixed integer rounding and single node flow sets revisited. *Annals of Operations Research*, 153(1):47–77, 2007.

[MB07] Rym M'Hallah and R.L. Bulfin. Minimizing the weighted number of tardy jobs on a single machine with release dates. *European Journal of Operational Research*, 176(2):727–744, January 2007.

[MCZ10] Ying Ma, Chengbin Chu, and Chunrong Zuo. A survey of scheduling with deterministic machine availability constraints. *Computers & Industrial Engineering*, 58(2):199–211, March 2010.

[MKSC13] Racem Mellouli, Imed Kacem, Chérif Sadfi, and Chengbin Chu. Lagrangian relaxation and column generation-based lower bounds for the $p_m, h_{j1}|| \sum w_i c_i$ scheduling problem. *Applied Mathematics and Computation*, 219(22):10783–10805, July 2013.

[Moo68] J. M. Moore. A n job one machine algorithm for minimizing the number of late jobs. *Management Science*, 15:102–109, 1968.

[MSCK09] Racem Mellouli, Chérif Sadfi, Chengbin Chu, and Imed Kacem. Identical parallel-machine scheduling under availability constraints to minimize the sum of completion times. *European Journal of Operational Research*, 197(3):1150–1165, September 2009.

[PPR03] Laurent Péridy, Eric Pinson, and David Rivreau. Using short-term memory to minimize the weighted number of late jobs on a single machine. *European Journal of Operational Research*, 148(3):591–603, August 2003.

[Sad08] Ruslan Sadykov. A branch-and-check algorithm for minimizing the weighted number of late jobs on a single machine with release dates. *European Journal of Operational Research*, 189(3):1284–1304, Sep 2008.

[Sch00] Gunter Schmidt. Scheduling with limited machine availability. *European Journal of Operational Research*, 121(1):1–15, February 2000.

[SW06] Ruslan Sadykov and Laurence A. Wolsey. Integer programming and constraint programming in solving a multimachine assignment scheduling problem with deadlines and release dates. *INFORMS JOURNAL ON COMPUTING*, 18(2):209–217, January 2006.

[TF12] Shunji Tanaka and Shuji Fujikuma. A dynamic-programming-based exact algorithm for general single-machine scheduling with machine idle time. *Journal of Scheduling*, 15(3):347–361, 2012.

[Wol77] L. A. Wolsey. Valid inequalities and superadditivity for 0-1 integer programs. *Mathematics of Operations Research*, 2(1):66–77, January 1977.

[ZOW14] Xueling Zhong, Jinwen Ou, and Guoqing Wang. Order acceptance and scheduling with machine availability constraints. *European Journal of Operational Research*, 232(3):435–441, February 2014.

Appendix A. Dynamic program for lifting with time windows

To solve the optimization problem $LIFT_1$, we can use the following recursive functions:

- $F(w)$, $w \in \{0, \dots, \sum_{i=1}^{u} \omega_i\}$ is the earliest possible completion time of a schedule composed of a set of jobs whose total weight is exactly w. If no such set of jobs exists, then $F(w) = \infty$.
- $f(w,i)$, $w \in \{0, \dots, \sum_{j=1}^{u} \omega_j\}$, $i \in \{1, \dots, u\}$ is the earliest possible completion time of a schedule composed of a subset of jobs in $\{\theta_1, \dots, \theta_i\}$ whose total weight is exactly w. If no such subset of jobs exists, then $f(w,i) = \infty$.
- $f_1(w,i)$ (resp. $f_0(w,i)$), $w \in \{0, \dots, \sum_{i=1}^{u} \omega_i\}$, $i \in \{1, \dots, u\}$ is similar to $f(w,i)$, with the additional constraint that θ_i belongs (resp. does not belong) to the subset of processed jobs.
- $ct(w,i)$ denotes the earliest possible completion time of θ_i when it is processed together with a subset of jobs in $\{\theta_1, \dots, \theta_{i-1}\}$ whose total weight is $w - \omega_i$.

These functions are linked with the help of the following relations:

$$\left\{\begin{array}{ll}
F(w) = f(w,u) & w \in \{0, \dots, \sum_{i=1}^{u} \omega_i\} \\
f(w,0) = \left\{\begin{array}{ll} 0 & \text{if } w = 0 \\ \infty & \text{otherwise} \end{array}\right. & w \in \{0, \dots, \sum_{j=1}^{u} \omega_j\} \\
f(w,i) = \left\{\begin{array}{ll} \min\big(f_0(w,i), f_1(w,i)\big) & \text{if } \theta_i \notin \Phi \\ f_1(w,i) & \text{if } \theta_i \in \Phi \end{array}\right. & w \in \{0, \dots, \sum_{j=1}^{u} \omega_j\}, i \in \{1, \dots, u\} \\
f_0(w,i) = f(w,i-1) & w \in \{0, \dots, \sum_{j=1}^{u} \omega_j\}, i \in \{1, \dots, u\} \\
f_1(w,i) = \left\{\begin{array}{ll} ct(w,i) & \text{if } ct(w,i) \leq \bar{d}_i \\ \infty & \text{otherwise} \end{array}\right. & w \in \{0, \dots, \sum_{j=1}^{u} \omega_j\}, i \in \{1, \dots, u\} \\
ct(w,i) = \left\{\begin{array}{l} \max\big(r_i, f(w-\omega_i, i-1)\big) + p_i \\ \infty \end{array}\right. & \begin{array}{l} w \in \{\omega_i, \dots, \sum_{j=1}^{u} \omega_j\}, i \in \{1, \dots, u\} \\ w \in \{0, \dots, \omega_i - 1\}, i \in \{1, \dots, u\} \end{array}
\end{array}\right. \tag{A.1}$$

Finally, the optimal value of the lifting problem is $z = \max\{w | F(w) \neq \infty\}$. The dynamic program counts $u(\sum_{i=1}^{u} \omega_i + 1)$ states, each of which requires $\mathcal{O}(1)$ operation to compute the corresponding optimal value. Therefore, the optimal value z can be computed in $\mathcal{O}(u \sum_{i=1}^{u} \omega_i)$. Because it is relatively lengthy, we provide the proof of validity for these equations as supplementary material.

Appendix B. Dynamic program for lifting with time windows, groups of jobs, and dominance of actual occurrences over virtual occurrences

To solve $LIFT_2$, we use recursive functions, which we define with the help of the following notations: $A = \{0,1\}^u$, $B = \{0,1\}^\chi$, $\bar{W} = \sum_{i=1}^{u} \omega_u$, $W =$

$\{0, \ldots, \bar{W}\}$, $U = \{1, \ldots, u\}$. For ease of reading, when the domain of 4-tuple (w, i, Π, Λ) is not specified, we assume that $i \in U$, $w \in W$, $\Pi \in A$, and $\Lambda \in B$.

$$
\left\{
\begin{array}{ll}
f(w,0,\Pi,\Lambda) = & \begin{cases} 0 & \text{if } w=0, \Pi = \mathbf{0}, \Lambda = \mathbf{0} \\ \infty & \text{otherwise} \end{cases} \\
f(w,i,\Pi,\Lambda) = & \begin{cases} f_1(w,i,\Pi,\Lambda) & \text{if } \lambda_i = 0 \\ f_2(w,i,\Pi,\Lambda) & \text{if } \lambda_i > 0 \\ f_3(w,i,\Pi,\Lambda) & \text{if } \lambda_i < 0 \end{cases} \\
f_1(w,i,\Pi,\Lambda) = & \begin{cases} f_4(w,i,\Pi,\Lambda) & \text{if } g_i \neq \emptyset \\ f_5(w,i,\Pi,\Lambda) & \text{if } g_i = \emptyset \end{cases} \\
f_2(w,i,\Pi,\Lambda) = & \begin{cases} \infty & \text{if } \Lambda_{\lambda_i} = 1 \\ f_8(w,i,\Pi,\Lambda) & \text{if } (\Lambda_{\lambda_i} = 0) \wedge (g_i \neq \emptyset) \\ f_9(w,i,\Pi,\Lambda) & \text{if } (\Lambda_{\lambda_i} = 0) \wedge (g_i = \emptyset) \end{cases} \\
f_3(w,i,\Pi,\Lambda) = & \begin{cases} f_{12}(w,i,\Pi,\Lambda) & \text{if } g_i \neq \emptyset \\ f_{13}(w,i,\Pi,\Lambda) & \text{if } g_i = \emptyset \end{cases} \\
f_4(w,i,\Pi,\Lambda) = & \begin{cases} f_6(w,i,\Pi,\Lambda) & \text{if } \theta_i \in \Phi \\ \min(f_6(w,i,\Pi,\Lambda), f(w,i-1,\Pi,\Lambda)) & \text{if } \theta_i \notin \Phi \end{cases} \\
f_5(w,i,\Pi,\Lambda) = & \begin{cases} f_7(w,i,\Pi,\Lambda) & \text{if } \theta_i \in \Phi \\ \min(f_7(w,i,\Pi,\Lambda), f(w,i-1,\Pi,\Lambda)) & \text{if } \theta_i \notin \Phi \end{cases} \\
f_6(w,i,\Pi,\Lambda) = & \begin{cases} c_6(w,i,\Pi,\Lambda) & \text{if } \Pi_i = 1 \wedge c_6(w,i,\Pi,\Lambda) \leq \bar{d}_i \\ \infty & \text{if } \Pi_i = 0 \vee c_6(w,i,\Pi,\Lambda) > \bar{d}_i \end{cases} \\
c_6(w,i,\Pi,\Lambda) = & \begin{cases} \max(r_i, f(w-\omega_i, i-1, \Pi', \Lambda)) + p_i & \text{if } w \geq \omega_i, \text{with } \Pi'_i = 0, \\ & \qquad \Pi'_k = \Pi_k, k \neq i \\ \infty & \text{if } w < \omega_i \end{cases} \\
f_7(w,i,\Pi,\Lambda) = & \begin{cases} c_7(w,i,\Pi,\Lambda) & \text{if } c_7(w,i,\Pi,\Lambda) \leq \bar{d}_i \\ \infty & \text{if } c_7(w,i,\Pi,\Lambda) > \bar{d}_i \end{cases} \\
c_7(w,i,\Pi,\Lambda) = & \begin{cases} \max(r_i, f(w-\omega_i, i-1, \Pi, \Lambda)) + p_i & \text{if } w \geq \omega_i \\ \infty & \text{if } w < \omega_i \end{cases} \\
f_8(w,i,\Pi,\Lambda) = & \begin{cases} f_{10}(w,i,\Pi,\Lambda) & \text{if } \theta_i \in \Phi \\ \min(f_{10}(w,i,\Pi,\Lambda), f(w,i-1,\Pi,\Lambda)) & \text{if } \theta_i \notin \Phi \end{cases} \\
f_9(w,i,\Pi,\Lambda) = & \begin{cases} f_{11}(w,i,\Pi,\Lambda) & \text{if } \theta_i \in \Phi \\ \min(f_{11}(w,i,\Pi,\Lambda), f(w,i-1,\Pi,\Lambda)) & \text{if } \theta_i \notin \Phi \end{cases} \\
f_{10}(w,i,\Pi,\Lambda) = & \min(f_6(w,i,\Pi,\Lambda), f_6(w,i,\Pi,\Lambda')) \quad \text{with } \Lambda'_{\lambda_i} = 1, \Lambda'_\lambda = \Lambda_\lambda, \lambda \neq \lambda_i \\
f_{11}(w,i,\Pi,\Lambda) = & \min(f_7(w,i,\Pi,\Lambda), f_7(w,i,\Pi,\Lambda')) \quad \text{with } \Lambda'_{\lambda_i} = 1, \Lambda'_\lambda = \Lambda_\lambda, \lambda \neq \lambda_i \\
f_{12}(w,i,\Pi,\Lambda) = & \begin{cases} f(w,i-1,\Pi,\Lambda) & \text{if } \Lambda_{\lambda_i} = 0 \wedge \theta_i \notin \Phi \\ \infty & \text{if } \Lambda_{\lambda_i} = 0 \wedge \theta_i \in \Phi \\ \min\{f(w,i-1,\Pi,\Lambda), & \\ \qquad f_6(w,i,\Pi,\Lambda), & \text{if } \Lambda_{\lambda_i} = 1 \text{with } \Lambda'_{\lambda_i} = 0, \Lambda'_\lambda = \Lambda_\lambda, \lambda \neq \lambda_i \\ \qquad f_6(w,i,\Pi,\Lambda')\} & \end{cases} \\
f_{13}(w,i,\Pi,\Lambda) = & \begin{cases} f(w,i-1,\Pi,\Lambda) & \text{if } \Lambda_{\lambda_i} = 0 \wedge \theta_i \notin \Phi \\ \infty & \text{if } \Lambda_{\lambda_i} = 0 \wedge \theta_i \in \Phi \\ \min\{f(w,i-1,\Pi,\Lambda), & \\ \qquad f_8(w,i,\Pi,\Lambda), & \text{if } \Lambda_{\lambda_i} = 1 \text{with } \Lambda'_{\lambda_i} = 0, \Lambda'_\lambda = \Lambda_\lambda, \lambda \neq \lambda_i \\ \qquad f_8(w,i,\Pi,\Lambda')\} & \end{cases}
\end{array}
\right.
\tag{B.1}
$$

Finally, the optimal value of the lifting problem $LIFT_2$ is $z = \max\{w | f(w, u, \Pi, \mathbf{0}) \neq \infty, w \in W, \Pi \in A\}$. Indeed, by definition of $S(w, u, \Pi, \mathbf{0})$, $\max\{f(w, u, \Pi, \mathbf{0}) | \Pi \in A\}$ is the minimum possible completion of a feasible schedule composed of jobs in Θ and of weight w. Each function in (B.1) is calculated in $\mathcal{O}(1)$ operations. Therefore, the time complexity of a straightforward dynamic programming algorithm is equal to the overall number of states, i.e. $\mathcal{O}(\bar{W} \cdot u \cdot 2^{|\pi|+\chi})$. The proof of validity for these equations is available as supplementary material.

Method		Model (B)			(B)+(SLCC) $LIFT_1$			(B)+(SLCC) $LIFT_2$ $\chi = 0$		
Problem type		#unsolved	Avg. time	Avg. LR	#unsolved	Avg. time	Avg. root	#unsolved	Avg. time	Avg. root
# jobs	# solved by all	1 h.	(sec.)	gap	1 h.	(sec.)	gap	1 h.	(sec.)	gap
$1\|r_i\|\sum w_iU_i$										
250	2	**0**	2336.7	6.48%	**0**	1728.5	4.78%	**0**	**523.6**	2.39%
300	0	1	[a]	7.17%	1	[a]	5.29%	**0**	[a]	3.24%
350	1	1	1494.6	2.49%	1	766.5	2.06%	**0**	**598.8**	**1.39%**
400	2	5	1493.3	2.28%	4	1768.9	1.79%	**1**	**1240.9**	1.31%
450	1	3	3212.1	1.85%	**2**	**607.0**	1.41%	**2**	986.4	1.13%
500	2	5	1691.1	1.74%	**3**	**1042.2**	1.42%	4	1266.4	1.21%
$1\|r_i, nr-a\|\sum w_iU_i$										
300	2	5	2400.4	3.00%	5	499.7	2.37%	2	808.4	**1.72%**
400	4	10	1885.9	2.05%	10	2765.5	1.58%	1	1465.7	**1.13%**
500	11	21	1962.9	1.43%	21	912.6	1.12%	10	954.7	**0.93%**
$1\|r_i, ST_{si}\|\sum w_iU_i$										
200	3	5	1302.7	0.70%	**0**	23.3	**0.68%**	**0**	34.3	**0.68%**
300	11	11	2416.9	2.82%	6	1433.4	2.16%	**2**	725.0	1.48%
400	21	27	1882.0	1.95%	17	1567.2	1.54%	**9**	**864.4**	**1.14%**
500	16	55	1954.8	1.34%	36	1879.8	1.09%	**23**	**1186.3**	**0.91%**
Method					(B)+(SLCC) $LIFT_2$ $\chi = 1$			(B)+(SLCC) $LIFT_2$ $\chi = 2$		
Problem type					#unsolved	Avg. time	Avg. root	#unsolved	Avg. time	Avg. root
# jobs	# solved by all				1 h.	(sec.)	gap	1 h.	(sec.)	gap
$1\|r_i\|\sum w_iU_i$										
250	2				**0**	586.6	2.39%	**0**	720.7	**2.33%**
300	0				**0**	[a]	**3.17%**	**0**	[a]	3.18%
350	1				**0**	748.7	1.43%	**0**	661.8	1.46%
400	2				2	2154.1	**1.30%**	2	1085.7	1.32%
450	1				**2**	1189.1	**1.09%**	**2**	1563.0	1.15%
500	2				**3**	1064.4	1.21%	**3**	1351.2	**1.19%**
$1\|r_i, nr-a\|\sum w_iU_i$										
300	2				**1**	1488.6	1.73%	**1**	**510.7**	1.74%
400	4				**0**	1771.9	1.14%	**0**	**1504.0**	1.14%
500	11				**7**	**942.8**	**0.93%**	11	879.5	0.94%
$1\|r_i, ST_{si}\|\sum w_iU_i$										
200	3				1	32.0	0.71%	1	22.1	**0.68%**
300	11				**2**	**673.1**	1.48%	3	706.9	**1.47%**
400	21				11	1200.9	**1.14%**	**9**	1049.3	1.15%
500	16				25	1037.0	**0.91%**	29	1118.0	0.92%

Table B.2: Results obtained using (SLCC) on instances left open using Model (B) within 1000 sec. Average times and numbers of nodes are calculated on instances that are solved by all methods in the table. The number of these instances is reported in the column "# solved by all" for each problem type and each number of jobs. [a] Model (B) alone cannot solve this instance in 1000 sec., nor in one hour.