Scheduling jobs with release dates on identical parallel machines by minimizing the total weighted completion time

Arthur Kramer[*1,2], Mauro Dell'Amico[2], Dominique Feillet[3], and Manuel Iori[2]

[1]*Departamento de Engenharia de Produção, Universidade Federal do Rio Grande do Norte, Brazil*

[2]*Dipartimento di Scienze e Metodi dell'Ingegneria, Università degli Studi di Modena e Reggio Emilia, Italy*

[3]*École des Mines de Saint-Étienne and UMR CNRS 6158 LIMOS, CMP Georges Charpak, France*

Abstract

This paper addresses the problem of scheduling a set of jobs that are released over the time on a set of identical parallel machines, aiming at the minimization of the total weighted completion time. This problem, referred to as $P|r_j|\sum w_j C_j$, is of great importance in practice, because it models a variety of real-life applications. Despite its importance, the $P|r_j|\sum w_j C_j$ has not received much attention in the recent literature. In this work, we fill this gap by proposing mixed integer linear programs and a tailored branch-and-price algorithm. Our branch-and-price relies on the decomposition of an arc-flow formulation and on the use of efficient exact and heuristic methods for solving the pricing subproblem. Computational experiments carried out on a set of randomly generated instances prove that the proposed methods can solve to the proven optimality instances with up to 200 jobs and 10 machines, and provide very low gaps for larger instances.

1 Introduction

Scheduling problems are among the most important problems in the combinatorial optimization field, being intensively studied since the early 1950s. These problems are relevant because of their practical and theoretical importance. Many practical situations are modeled by means of scheduling problems, including personnel (see, e.g., van den Bergh *et al.* 2013), production (see, e.g., Allahverdi 2015), health care (see, e.g., Hall 2012) and project (see, e.g., Demeulemeester and Herroelen 2002) problems. In addition to their practical importance, solving scheduling problems is usually a challenging task because many of them are $\mathcal{NP}$-hard. Thus, the use of advanced optimization techniques such as column generation, branch-and-bound and/or Benders decomposition, just to cite some, is fundamental for their efficient solution.

In this work, we address a particular scheduling problem in which we are given a set $J = \{1, \ldots, n\}$ of jobs to be processed, without preemption, on a set $M = \{1, \ldots, m\}$ of identical parallel machines. Each job $j \in J$ has a processing time p_j, a release date r_j and a weight w_j. The objective is to find a schedule for which the sum of the weighted completion times of the jobs, $\sum_{j \in J} w_j C_j$, is a minimum, where the completion time of a job, C_j, is defined as the time in which its processing is finished. Using the scheduling

---

[*]email: arthur.kramer@ct.ufrn.br

classification of Graham *et al.* (1979), this problem can be referred to as $P|r_j|\sum w_j C_j$. The $P|r_j|\sum w_j C_j$ is $\mathcal{NP}$-hard because generalizes the $1|r_j|\sum C_j$, which was proven to be $\mathcal{NP}$-hard by Lenstra *et al.* (1977).

In the scheduling literature, only a limited number of works address the $P|r_j|\sum w_j C_j$. Among them, Baptiste *et al.* (2008) propose lower bounds and Nessah *et al.* (2008) a branch-and-bound algorithm. We are not aware of any recent work proposing a more efficient exact method, thus representing a lack in the literature that we try to fill with this paper.

A variety of combinatorial optimization problems have been successfully solved by means of decomposition techniques, i.e., methods that rely on decomposing the problem in master and subproblems (see, e.g., Valério de Carvalho 1999; Feillet 2010; Delorme *et al.* 2017 and Kowalczyk and Leus 2018). In this article, we focus on the application of these techniques to the $P|r_j|\sum w_j C_j$. In particular, we present a *time-indexed* (TI) formulation (see, e.g., Sousa and Wolsey 1992), an *arc-flow* (AF) formulation (see, e.g., Valério de Carvalho 1999; Delorme *et al.* 2016 and Kramer *et al.* 2019a), and a tailored *branch-and-price* (B&P) algorithm, and use them to exactly solve large sized instances of the $P|r_j|\sum w_j C_j$. Our B&P is based on the *Dantzig-Wolfe* (DW) decomposition of the AF formulation. On one hand, AF formulations are characterized by having a pseudo-polynomial number of variables and constraints, and hence their application to large size instances may be prohibitive. On the other hand, the DW decomposition makes it possible to obtain a reformulation with very few constraints but with an exponential number of variables. The B&P algorithm consists in solving at each node of a *branch-and-bound* (B&B) tree a linear relaxation of the DW reformulation by means of a *column generation* (CG) algorithm. To this aim, we propose CG algorithms that combine the use of dominance rules and the use of exact and heuristic methods to tackle the pricing subproblem. We performed computational experiments on randomly generated instances to evaluate the performance of the proposed methods. The results that we obtained show that our best proposed method is able to solve to the proven optimality instances with up to 200 jobs and 10 machines.

The remainder of this work is organized as follows. A literature review on the $P|r_j|\sum w_j C_j$ and closely related problems is provided in Section 2. *Mixed integer linear programming* (MILP) TI and AF formulations are presented in Section 3. The proposed B&P algorithm is detailed in Section 4. The performance evaluation of the proposed methods is presented in Section 5, and then the concluding remarks and further research directions are reported in Section 6.

2   Literature review

The $P|r_j|\sum w_j C_j$ considers some important characteristics of production scheduling environments that are commonly faced in practice, such as the presence of multiple machines,

release dates and priorities on the jobs. The problem has not received much attention in the recent literature, but the literature on closely related problems is rich and gives some interesting insights also on how to tackle the $P|r_j|\sum w_j C_j$. In the following, we briefly revise scheduling problems with release dates and involving objective functions related to the total weighted completion time minimization.

The $1|r_j|\sum w_j C_j$ is a single machine $\mathcal{NP}$-hard problem for which heuristic methods, lower bounding schemes, B&B approaches and MILP formulations have been proposed in the literature. Hariri and Potts (1983) proposed a simple heuristic method that consists in iteratively sequencing the jobs according to the non-increasing order of the w_j/p_j ratio, in such a way that the release dates are satisfied and unnecessary idle times are avoided. The authors also proposed a lower bounding scheme based on the combination of their heuristic algorithm with a Lagrangean relaxation of the release date constraints. In addition, a B&B integrating these methods with dominance rules was developed. With the resulting method, the authors solved instances with up to 50 jobs. Dyer and Wolsey (1990) provided a theoretical analysis and developed MILP formulations to compute lower bounds for the $1|r_j|\sum w_j C_j$. Lower bounding techniques were also proposed by Belouadah *et al.* (1992). Their main technique is based on the idea of dividing the jobs in pieces so that an efficiently solvable problem is derived.

Still with regard to the $1|r_j|\sum w_j C_j$, a TI formulation was proposed by van den Akker *et al.* (2000). The main advantage of this formulation concerns the strength of its linear relaxation, while its main drawback is due to its pseudo-polynomial size. To work around this fact, the authors applied a DW decomposition and solved the linear relaxation of the problem by means of column and cut generation techniques, thus obtaining a valid lower bound. Avella *et al.* (2005) focused on obtaining small duality gaps. They combined a TI formulation with Lagrangean relaxation techniques to obtain near-optimal solutions for large sized instances with up to 400 jobs. A framework featuring B&B, *dynamic programming* (DP) and constraint programming techniques was proposed by Pan and Shi (2008). With this framework the authors were able to solve instances with up to 200 jobs to the proven optimality. Later, Tanaka and Fujikuma (2012) developed a DP based algorithm that considers state elimination, dominance rules and Lagrangean relaxation concepts. To the best of our knowledge, this method represents the state-of-the-art approach for solving the $1|r_j|\sum w_j C_j$, being able to solve to the proven optimality instances with up to 200 jobs much faster than Pan and Shi's method.

Concerning identical parallel machine variants, many authors studied problems where preemption is allowed. Among them, Du *et al.* (1990) and Baptiste *et al.* (2007) investigated the complexity of the problem of minimizing the total flow time when preemption is allowed, where the flow time F_j of a job is defined as the difference between the time in which its processing is finished and its release date, i.e., $F_j = C_j - r_j$. The former work proved that the $P|r_j, pmtn|\sum F_j$ is $\mathcal{NP}$-hard, while the latter demonstrated that if all the processing times of the jobs are identical (i.e., $p_j = p$ for all $j \in J$) the problem is solv-

able in polynomial time. A similar result has been achieved by Brucker and Kravchenko (2005), who proposed a polynomial algorithm for solving the closely related $P|r_j, p_j = p, pmtn| \sum C_j$. Regarding again the $P|r_j, pmtn| \sum F_j$, Leonardi and Raz (2007) analyzed the *shortest remaining processing time* (SRPT) rule, which at each time assigns to the machines the jobs already released according to their shortest remaining processing time. They showed that the SRPT rule is an $\mathcal{O}(\log(\min(n/m, p_{\max})))$-approximation algorithm for the $P|r_j, pmtn| \sum F_j$, where $p_{\max} = \max_{j \in J}\{p_j\}$.

Regarding parallel machine problems without preemption, Leonardi and Raz (2007) developed an approximation algorithm for the $P|r_j| \sum F_j$ that is based on their previous results for the problem with preemption. Their method relies on transforming a preemptive schedule into a non-preemptive one. Yalaoui and Chu (2006), in turn, tackled the problem of minimizing the related total completion time of the jobs. In their work, B&B methods that consider theoretical properties of the problem as well as lower and upper bounding schemes based on job splitting and release dates relaxation are presented. The authors state that their work was the first to derive an exact approach to the $P|r_j| \sum C_j$. The results that they obtained show that instances with up to 120 jobs and 5 machines could be solved to the proven optimality.

With regard to the weighted version, i.e., the $P|r_j| \sum w_j C_j$ studied in this work, we highlight the contributions by Hall *et al.* (1997), Baptiste *et al.* (2008) and Nessah *et al.* (2008). Hall *et al.* (1997) proposed approximation algorithms with performance guarantee for a variety of scheduling problems to minimize the total weighted completion time, including the $P|r_j| \sum w_j C_j$. For this problem, in particular, their method is based on an approximation algorithm for the variant without release dates. The work of Baptiste *et al.* (2008) focused on the study of existing and new lower bounding schemes for a variety of parallel machine scheduling problems, including the $P|r_j| \sum w_j C_j$. Nessah *et al.* (2008), in turn, proposed dominance rules, lower bounding schemes and a heuristic method. They showed that the developed lower bounds and the heuristic method provide an average gap of around 3% for instances with up to 500 jobs and 9 machines. When these components are integrated into a B&B framework, instances with up to 100 jobs and 5 machines are solved to the proven optimality. To the best of our knowledge, this method is the state-of-the-art exact algorithm for the $P|r_j| \sum w_j C_j$.

3 Mathematical formulations

In this section, we present three different MILP formulations for $P|r_j| \sum w_j C_j$, namely, a time-indexed, an arc-flow and a set covering formulation.

3.1 Time-indexed formulation

Time-indexed (TI) formulations (see, e.g., Sousa and Wolsey 1992) seek for scheduling jobs over a sufficiently large time horizon by making use of time-indexed binary variables.

Such variables, referred to as x_{jt}, assume value 1 if job $j \in J$ starts its processing at time $t \in \{0, \ldots, T - p_j\}$, and 0 otherwise, where T represents the end of the time horizon. This time horizon should be sufficiently large to ensure that there exists at least an optimal solution for the problem, but needs to be as short as possible in order to reduce the formulation size. In our implementation we adopted

$$T = \left\lfloor \frac{1}{m} \sum_{j \in J} p_j + \frac{(m-1)}{m} p_{\max} \right\rfloor + r_{\max} \tag{1}$$

where $p_{\max}$ and $r_{\max}$ represent the maximum processing time and the maximum release date among all jobs $j \in J$, respectively. Note that Equation (1) adds $r_{\max}$ to the time horizon estimation of Kramer et $al.$ (2019a) for the $P || \sum w_j C_j$. Because this estimation guarantees to find an optimal solution for the $P || \sum w_j C_j$ (as also proved by van den Akker et $al.$ 1999) and because all jobs are available at time r_{max}, we are guaranteed to find an optimal solution within T.

A TI formulation for the $P|r_j| \sum w_j C_j$ is as follows.

$$\text{(TI)} \quad \min \sum_{j \in J} \left(w_j p_j + \sum_{t=r_j}^{T-p_j} w_j t x_{jt} \right) \tag{2}$$

$$\text{s.t.} \sum_{t=r_j}^{T-p_j} x_{jt} \geq 1 \qquad j \in J \tag{3}$$

$$\sum_{j \in J} \sum_{s=l_{jt}}^{u_{jt}} x_{js} \leq m \qquad t \in \{0, \ldots, T\} \tag{4}$$

$$x_{jt} \in \{0, 1\} \qquad j \in J, t \in \{r_j, \ldots, T - p_j\} \tag{5}$$

where $l_{jt} = \max\{r_j, t + 1 - p_j\}$ and $u_{jt} = \min\{t, T + 1 - p_j\}$. The objective function (2) seeks for the minimization of the total weighted completion time, constraints (3) ensure that all jobs are processed, constraints (4) guarantee that at most m jobs are processed in parallel, i.e., that at most one job at a time is processed on a machine, and constraints (5) define the variable domain. The TI formulation (2)-(5) is characterized by $\mathcal{O}(nT)$ variables and $\mathcal{O}(n + T)$ constraints.

3.2 Arc-flow formulation

Arc-$flow$ (AF) formulations model problems using flows on a pseudo-polynomial size capacitated network. This technique has been successfully employed to model classical $\mathcal{NP}$-hard optimization problems, such as cutting stock problems (Valério de Carvalho 1999; Delorme et $al.$ 2016) and scheduling problems (Kramer et $al.$ 2019a; Mrad and Souayah 2018).

To formulate the $P|r_j| \sum w_j C_j$ using an AF formulation, let us define an acyclic di-

rected multigraph $G = (N, A)$, and a set R containing all distinct job release dates plus $\{T\}$. The set of vertices $N \subseteq \{0, 1, \ldots, T\}$ represents the possible starting times and completion times of a job in the planning horizon, for non dominated feasible solutions. These times can be obtained by using Algorithm 1, and are usually called *normal patterns* in the literature (see, e.g., Côté and Iori 2018). The set of arcs A is partitioned in such a way that $A = \cup_{j \in J} A_j \cup A_0$, where $A_j = \{(q, r, j) : q \in N, r_j \leq q \leq T - p_j \text{ and } r = q + p_j\}$ is the set of job arcs associated to job j and $A_0 = \{(q, r, 0) : q \in N, r = \min(t \in R : t > q)\}$ is the set of loss arcs. Each job arc in A_j represents a possible processing of a job j on a machine, whereas loss arcs in A_0 model machine idle times. The procedure adopted to obtain set A is detailed in Algorithm 1.

Algorithm 1 Construction of $G = (N, A)$

1: **procedure** CREATE_GRAPH(T)
2:     **initialize** $R \leftarrow$ distinct values in $\{r_1, \ldots, r_n\} \cup \{T\}$ sorted in increasing order;
3:     **initialize** $i \leftarrow 0$;
4:     **initialize** $N \leftarrow \emptyset; A[0 \ldots n] \leftarrow \emptyset$;   ▷ N: set of vertices; $A[j]$: set A_j
5:     **initialize** $P[0 \ldots T] \leftarrow 0$;   ▷ P: array of size $T + 1$, used to store arc tails
6:     **for** $q \in R$ **do** $P[q] \leftarrow 1$;
7:     **for** $q \leftarrow 0$ to T **do**
8:         **if** $P[q] = 1$ **then**
9:             **for** $j \leftarrow 1$ to n **do**
10:                 **if** $q \geq r_j$ and $q + p_j \leq T$ **then** $P[q + p_j] \leftarrow 1; A[j] \leftarrow A[j] \cup \{(q, q + p_j, j)\}$;
11:     **for** $q \leftarrow 0$ to T **do**
12:         **if** $P[q] = 1$ **then**
13:             $N \leftarrow N \cup \{q\}$;
14:             **while** $q \geq R[i]$ **do** $i \leftarrow i + 1$;
15:             $A[0] \leftarrow A[0] \cup \{(q, R[i], 0)\}$;   ▷ $A[0]$: set of dummy/loss arcs
16:     $A \leftarrow \cup_{j \in J} A[j] \cup A[0]$;
17:     **return** N, A

In the AF formulation, a valid schedule of jobs on a machine corresponds to a path in G. Thus, we associate with each job arc a binary variable $x_{qrj}, (q, r, j) \in A_j$, assuming the value 1 if job j starts its processing at time q and finishes at time r, 0 otherwise, and with each loss arc a continuous variable $x_{qr0}, (q, r, 0) \in A_0$. In this way, the AF formulation models the $P|r_j| \sum w_j C_j$ as the problem of finding m paths in G containing all jobs $j \in J$ and having a minimum total weighted completion time, namely:

$$\text{(AF)} \quad \min \sum_{j \in J} \left(w_j p_j + \sum_{(q,r,j) \in A_j} w_j q x_{qrj} \right) \tag{6}$$

$$\text{s.t.} \sum_{(q,r,j) \in A_j} x_{qrj} \geq 1 \qquad j \in J \tag{7}$$

$$\sum_{(q,r,j) \in A} x_{qrj} - \sum_{(p,q,j) \in A} x_{pqj} = \begin{cases} m, & \text{if } q = r_{\min} \\ -m, & \text{if } q = T \\ 0, & \text{otherwise} \end{cases} \qquad q \in N \tag{8}$$

$$x_{qrj} \in \{0, 1\} \qquad (q, r, j) \in A \setminus A_0 \tag{9}$$

$$0 \leq x_{qr0} \leq m \qquad (q,r,0) \in A_0 \qquad (10)$$

with $r_{\min} = min_{j \in J}\{r_j\}$. The objective function (6) minimizes the sum of the weighted completion times; constraints (7) state that each job should be processed at least once; constraints (8) impose the flow conservation at each time instant of the time horizon; and constraints (9) and (10) define variable domains. The formulation has a pseudo-polynomial size, with $\mathcal{O}(n|N|)$ variables and $\mathcal{O}(n + |N|)$ constraints.

3.3   Set covering formulation

Set covering (SC) formulations have been widely used to model many combinatorial optimization problems, in particular after the application of the DW decomposition of a given formulation. Let S be the set of all possible elementary schedules for a machine, i.e., those schedules where all selected jobs respect their release dates, are scheduled at most once and are not overlapping among them. Formally, S is defined as the set of all solutions of polytope (11)-(14):

$$\sum_{(q,r,j)\in A_j} x_{qrj} \leq 1 \qquad j \in J \qquad (11)$$

$$\sum_{(q,r,j)\in A} x_{qrj} - \sum_{(p,q,j)\in A} x_{pqj} = \begin{cases} 1, & \text{if } q = r_{\min} \\ -1, & \text{if } q = T \\ 0, & \text{otherwise} \end{cases} \qquad q \in N \qquad (12)$$

$$x_{qrj} \in \{0,1\} \qquad (q,r,j) \in A \setminus A_0 \qquad (13)$$

$$0 \leq x_{q,r,0} \leq 1 \qquad (q,r,0) \in A_0 \qquad (14)$$

For a given $s \in S$, let $a_{jq}^s \in \{0,1\}$ indicate whether job j starts at time q in schedule s or not. Then, $a_j^s = \sum_{(q,r,j)\in A_j} a_{jq}^s$ determines if job j is included in the schedule ($a_j^s = 1$) or not ($a_j^s = 0$), and $c_s = \sum_{j \in J} \sum_{(q,r,j)\in A_j} w_j \, a_{jq}^s \, (q + p_j)$ is the cost of the schedule.

By defining $\pi_s, \forall s \in S$, as a binary variable assuming value 1 if schedule s is in the solution and 0 otherwise, the $P|r_j|\sum w_j C_j$ can be modeled as:

$$(\text{SC}) \quad \min \sum_{s \in S} c_s \pi_s \qquad (15)$$

$$\text{s.t.} \sum_{s \in S} a_j^s \pi_s \geq 1 \qquad j \in J \qquad (16)$$

$$\sum_{s \in S} \pi_s \leq m \qquad (17)$$

$$\pi_s \in \{0,1\} \qquad s \in S \qquad (18)$$

The objective function (15) aims at minimizing the total weighted completion time; constraints (16) impose that each job $j \in J$ has to be processed at least once; constraint (17) imposes that no more than m schedules are used; and constraints (18) define the

variable domain.

As usual with SC formulations, the above model involves an exponential number of variables. In such cases, full enumeration of all variables is unpractical, and a CG approach must be used (see, e.g., Lübbecke and Desrosiers 2005; Feillet 2010). In the following sections, we detail the B&P method that we developed to solve this model.

4   Branch-and-price

4.1   Column generation principle

Column generation algorithms are often employed to solve *linear programs* (LPs) containing an exponential number of variables. These methods benefit from primal/dual information to avoid the full enumeration of columns. Basically, a CG algorithm consists of (a) solving a restricted LP containing a subset of variables; (b) retrieving dual information about the solution, identifying and adding new attractive variables (i.e., variables with a negative reduced cost, for a minimization problem) to the restricted LP. Steps (a) and (b) are repeated until no new attractive variable exists. In step (b), identifying new variables is not always a simple task. In general, this step requires solving a subproblem, known as pricing problem. This section details the components of our proposed CG method employed to solve the LP relaxation of formulation (15)-(18).

First, let us define MP as the LP relaxation of model SC (i.e., (15)-(18)), referred to as master problem, and MP(S'), where $S' \subseteq S$, as the restricted master problem obtained using only the variables in S'. Let us also define the dual of MP(S'), named DMP(S'), as:

$$(\text{DMP}(S')) \quad \max \sum_{j \in J} \lambda_j + m\lambda_0 \tag{19}$$

$$\text{s.t.} \sum_{j \in J} a_j^s \lambda_j + \lambda_0 \leq c_s \qquad\qquad s \in S' \tag{20}$$

$$\lambda_j \geq 0 \qquad\qquad j \in J \tag{21}$$

$$\lambda_0 \leq 0 \tag{22}$$

where $\lambda_j, j \in J$, and λ_0 are the dual variables associated with constraints (16) and (17), respectively.

The idea of the CG algorithm is to compute the optimal solution π^* to MP(S') to find the corresponding dual optimal solution λ^* and look for a column $s \in S \setminus \{S'\}$ with negative reduced cost, if any. Once there is no column with negative reduced cost, π^* is optimal for MP(S) and the process ends. The resulting CG procedure is shown in Algorithm 2.

In Algorithm 2, the set S' of initial columns is obtained by a simple heuristic procedure based on the combination of the earliest release date and the *weighted shortest processing*

Algorithm 2 Column generation

 1: **procedure** CG
 2:     **initialize** $S' \subseteq S$ with an initial subset of schedules;
 3:     **repeat**
 4:         **solve** MP(S') and obtain the optimal dual solution λ^*;
 5:         **if** $\exists\, s \in S \setminus \{S'\}$ with negative reduced cost **then** $S' \leftarrow S' \cup s$;
 6:     **until** $s = \emptyset$
 7:     **return** λ^*

---

time (WSPT) rules. The method consists in iteratively sequencing the jobs on machines. At each step, the job with lowest WSPT value among those already released is sequenced on the least loaded machine. Step 5, in turn, represents the process of finding and adding new columns, which is done by solving the pricing problem that we detail in the next sections.

4.2   Pricing problem

In our case, the pricing problem consists in finding single machine schedules with negative reduced cost $\bar{c}_s$ satisfying (11)-(14), where $\bar{c}_s$ is given by:

$$\bar{c}_s = c_s - \lambda_0 - \sum_{j \in J} a_j^s \lambda_j = \left(\sum_{j \in j} \sum_{(q,r,j) \in A} a_{jq}^s \left(w_j \left(q + p_j \right) - \lambda_j \right) \right) - \lambda_0. \qquad (23)$$

Let us consider the acyclic directed multigraph $G = (N, A)$ introduced in Section 3.2 and define a cost $\bar{c}_{(q,r,j)} = w_j(q + p_j) - \lambda_j$ on every arc $(q, r, j) \in A \setminus A_0$ and a cost $\bar{c}_{(q,r,0)} = 0$ on arcs in A_0. A machine schedule in S can then be represented by a path in G, starting from node r_{min} and ending at node T. Furthermore, the cost of this path is exactly the reduced cost of the machine schedule plus λ_0.

Unfortunately, the opposite is not true: a path between r_{min} and T in G does not necessarily represent a schedule of S, because some jobs might be repeated in the path (several arcs belonging to the same set A_j) while set S is limited to elementary schedules. For that reason, the pricing problem must include a further constraint, given by a non renewable resource preventing to select more than one arc from the same set A_j, for all $j \in J$. Following Feillet *et al.* (2004), n binary resources could be introduced to manage these constraints, at the expense of possibly prohibitive computing times.

4.3   Pricing problem with non-elementary schedules

An alternative emerges if we disregard constraints (11) and allow jobs to be taken more than once in a schedule. Let us denote by S_+ the set of machine schedules extended to these non-elementary schedules ($S \subseteq S_+$). Introducing in the master problem new variables for schedules in $S_+ \setminus S$ enlarges the solution space and might consequently weaken the lower bound provided by the model. On the other hand, it simplifies a lot the pricing problem, which becomes a standard *shortest path problem with resource constraints*

(SPPRC) in an acyclic (topologically-ordered) graph and can be solved by dynamic programming with a complexity $\mathcal{O}(nT)$. Algorithm 3 reports the *labeling correcting* (LC) algorithm that we implemented to solve this problem. In this algorithm, $F(p)$, $p \in N$, represents the best known cost among the paths that reach node p. An algorithm equivalent to Algorithm 3 was given in van den Akker *et al.* (1999).

Algorithm 3 LC_A

1: **procedure** $\text{LC}_A(G = (N, A))$
2:     **initialize** $F[r_{\min}] \leftarrow 0$; $F[p] \leftarrow +\infty$, $p \in N \setminus \{r_{\min}\}$;
3:     **for all** $p \in N$ in topological order **do**
4:         **if** $F[p] < F[p+1]$ **then** $F[p+1] \leftarrow F[p]$; $\triangleright$ idle time
5:         **for all** $(p, p+p_j, j) \in A_j, j \in J$ **do**
6:             $q \leftarrow p + p_j$; $c \leftarrow w_j q - \lambda_j$;
7:             **if** $F[p] + c \leq F[q]$ **then** $F[q] \leftarrow F[p] + c$;
8:     **return** $F[T] - \lambda_0$

It can be observed that the contribution of a job j scheduled to start at time q in the cost of a schedule is given by $w_j(q + p_j) - \lambda_j$. Note also that when constraints (11) are relaxed there are no constraints on the number of times a job can appear in a schedule. Considering these two facts, two reduction mechanisms can be derived.

First, arcs $(p, q, j) \in A_j$ can be removed from graph G as soon as $\bar{c}_{(pqj)} \geq 0$. Indeed, one can always replace these arcs with zero cost paths composed of arcs in A_0. Equivalently, defining $y_j = \lambda_j/w_j - p_j$, arcs (p, q, j) with $p \geq y_j$ can be removed; y_j can be interpreted as the starting time limit until which the job is worth being processed. In particular, if $\lambda_j \leq 0$, the job is never worth being processed: no arc from A_j is kept in the graph and job j is not considered in the pricing problem.

The following dominance criteria can be applied to further reduce the size of the graph.

Property 1. *Let us consider a pair of jobs* $(i, j) \in J$, *with* $i \neq j$ *and* $\lambda_i, \lambda_j > 0$. *If conditions:*

1) $r_i \leq r_j$;

2) $p_i \leq p_j$;

3) $w_i(r_j + p_i) - \lambda_i \leq w_j(r_j + p_j) - \lambda_j$; *and*

4) $w_i(\min(T - p_j, y_j) + p_i) - \lambda_i \leq w_j(\min(T - p_j, y_j) + p_j) - \lambda_j$

are satisfied, then job i *dominates job* j *and set* A_j *can be removed from the graph.*

Intuitively, it means that instead of using an arc in A_j, it is possible and preferable to use an arc in A_i (followed by one or several arcs from A_0 if $p_i < p_j$). This dominance relation is proved as follows.

Proof. Let us consider a schedule including an arc $(p, p + p_j, j) \in A_j$. We show that we can schedule job i instead of j, by improving the cost. Because of conditions 1) and 2), $(p, p+p_j, j)$ can be replaced with a path starting with arc $(p, p+p_i, i)$ and possibly followed by one or several zero-cost arcs from A_0. By construction and from the first reduction,

$p \in \{r_j, \min(T - p_j, y_j)\}$. Then, from conditions 3) and 4), and from the linearity of the cost function, we have $\bar{c}_{p,p+p_j,j} \geq \bar{c}_{p,p+p_i,i}$, which concludes the proof. $\qquad\square$

Values y_j and the set of dominated jobs can easily be precomputed before starting the solution of the pricing problem. Algorithm 3 with the graph modified according to the two reduction mechanisms forms our first pricing method, hereafter referred to as LC_A.

4.4 Pricing with non-elementary schedules without 1-cycles

A relevant alternative to LC_A is obtained by limiting the search in the pricing problem to schedules without k-cycles. A k-cycle is a sub-sequence $(j_{[i]}, \ldots, j_{[i+k]})$ of a schedule such that $j_{[i]} = j_{[i+k]}$. We denote by $S_{k\text{-cycle}}$ the subset of S_+ limited to non-elementary schedules without k-cycles. Note that $S \subseteq S_{k\text{-cycle}} \subseteq S_+$.

When $k = 1$, this problem can be solved in $\mathcal{O}(n^2 T)$ time by an label correcting procedure based on the DP algorithm shown in Pessoa *et al.* (2010). In the next sections, we refer to this method as LC_P. In the following, we show how to solve this problem in $\mathcal{O}(nT)$. The proposed approach extends LC_A and relies on the storage of two labels (i.e., two states) at each node q of the graph, representing the two best partial schedules that finish at time q with different final jobs. We introduce $F_l(q), l \in \{1, 2\}$, where $F_1(q)$ stores the best partial schedule finishing at q, and $F_2(q)$ represents the best schedule finishing at q in which the last job is different from the last job of the first schedule. A state L is represented with two values: $L.val$ is the cost of the label, $L.last$ the last job. Then, for each state $F_l(q)$ of the algorithm, two main classes of expansions can be performed: (i) to $q + p_j, j \in J$, representing that j starts in q and finishes in $q + p_j$; and (ii) to the subsequent release date $r \in R$, for each $j \in J : q + p_j < r$, representing the schedule of job j starting in q and finishing in $q + p_j$ plus an idle time from $q + p_j$ to r. Let us define $nr(p)$ as the subsequent release date of p, where $nr(p) = \min\{t : t > p, t \in R \cup \{T\}\}$ (initialized as shown in Algorithm 5). This approach, referred to as LC_{2l} from now on, is detailed in Algorithm 6.

Algorithm 4 Update

1: **procedure** UPDATE(L, F, q)
2:     **if** $L.val < F_1[q].val$ **then** $\triangleright$ L will become the best label at node q
3:         **if** $L.last \neq F_1[q].last$ **then** $\triangleright$ $F_1[q]$ will become the second label
4:             $F_2[q] \leftarrow F_1[q]$;
5:         $F_1[q] \leftarrow L$;
6:     **else if** $L.val < F_2[q].val$ **then**
7:         **if** $L.last \neq F_1[q].last$ **then** $\triangleright$ L will become the second label
8:             $F_2[q] \leftarrow L$;

By defining $Z^*_{MP(X)}$ as the optimal objective value of $MP(X)$, where X refers to the set of columns, the following relation can be stated:

$$Z^*_{MP(S_+)} \leq Z^*_{MP(S_{1\text{-cycle}})} \leq Z^*_{MP(S)}$$

Algorithm 5 Initialize vector nr

1: **procedure** INIT_NR( )
2:     **initialize** $R[\,]$; $r \leftarrow 0$; $\triangleright$ $R[\,]$: list of distinct release dates in increasing order
3:     $R \leftarrow R \cup T$; $\triangleright$ include T at the end of the list
4:     **for all** $p \in N$ in topological order **do**
5:         **if** $p = R[r]$ **then** $r \leftarrow r + 1$;
6:         $nr[p] \leftarrow R[r]$;

Algorithm 6 LC_{2l}

1: **procedure** $LC_{2l}(G = (N, A))$
2:     **initialize** $F_l[r_{\min}].val \leftarrow 0, l \in \{1,2\}$; $F_l[p].val \leftarrow \infty, p \in N \setminus \{r_{\min}\}, l \in \{1,2\}$;
3:     **initialize** $F_l[p].last \leftarrow 0, p \in N, l \in \{1,2\}$;
4:     Init_NR( ); $\triangleright$ initialize list $nr[\,]$
5:     **for all** $p \in N$ in topological order **do**
6:         **for all** $(p, p + p_j, j) \in A_j, j \in J$ **do**
7:             $l \leftarrow 1$;
8:             **if** $F_1[p].last = j$ **then** $l \leftarrow 2$ $\triangleright$ expansion from best label with $last \neq j$
9:             $L.val \leftarrow F_l[p].val + (p + p_j)\, w_j - \lambda_j$; $L.last \leftarrow j$;
10:            Update$(L, F, p + p_j)$; $\triangleright$ expansion with job j
11:            Update$(L, F, nr[p + p_j])$; $\triangleright$ expansion with job j and idle time until next release date
12:     **return** $F_1[T] - \lambda_0$

---

so that the lower bound obtained by taking into consideration only elementary schedules is stronger than the lower bound obtained by the 1-cycle relaxation, that in turn provides a stronger lower bound than the one obtained when all non-elementary schedules are allowed. Following this reasoning, and considering that for both of the aforementioned relaxations of the master problem, pricing problems are efficiently solved in pseudo-polynomial time, the two relaxations and the three algorithms LC_A, LC_P and LC_{2l} can be all considered good options for solving the problem.

4.5   Heuristic pricing

As the CG algorithm finishes when there is no column with negative reduced cost to be added to the master problem, one could heuristically solve the pricing and invoke the exact methods only when the heuristic methods fails in finding such columns. This combined approach is employed to speed up the CG algorithm. In this sense, we propose two greedy algorithms to solve the pricing subproblems.

Our first greedy method, denoted as H_{elem}, aims at finding columns in S, i.e., elementary schedules. The method is based on LC_A, with the difference that, for each node q of the graph, a list containing the jobs previously scheduled is stored in memory. Then, for a given node q, only expansions to jobs not in the list are allowed. This heuristic, detailed in Algorithm 7, runs in $\mathcal{O}(n^2 T)$ time and returns elementary schedules. It differs from Algorithm 3 by the use of list $L[p]$ to guarantee that only elementary schedules are obtained. This list is initialized at step 2 and updated in $\mathcal{O}(n)$ at steps 5 and 8.

The second proposed heuristic, shown in Algorithm 8 and referred to as $H_{\text{1-cycle}}$, seeks for schedules without 1-cycles, that is, schedules in $S_{\text{1-cycle}}$. This method is similar to

---

Algorithm 7 H_{elem}

1: **procedure** H_ELEM($G = (N, A)$)
2:     **initialize** $L[p] \leftarrow \emptyset, p \in N$;   ▷ $L[p]$: set of all jobs in the schedule associated with p
3:     **initialize** $F[r_{\min}] \leftarrow 0; F[p] \leftarrow +\infty, p \in N \setminus \{r_{\min}\}$;
4:     **for all** $p \in N$ in topological order **do**
5:         **if** $F[p] < F[p + 1]$ **then** $F[p + 1] \leftarrow F[p]; L[p + 1] \leftarrow L[p]$;   ▷ idle time
6:         **for all** $(p, p + p_j, j) \in A_j, j \in J \setminus L[p]$ **do**
7:             $q \leftarrow p + p_j; c \leftarrow w_j q - \lambda_j$;
8:             **if** $F[p] + c \leq F([q]$ **then** $F[q] \leftarrow F[p] + c; \quad L[q] \leftarrow L[p] \cup \{j\}$
9:     **return** $F[T] - \lambda_0$

H_{elem}, but instead of storing in memory a list of previously scheduled jobs per node, it maintains only the last scheduled job. Then, for each node q, expansions are limited to jobs different from the last scheduled job.

Algorithm 8 $H_{\text{1-cycle}}$

1: **procedure** H_1-CYCLE($G = (N, A)$)
2:     **initialize** $L[p] \leftarrow \emptyset, p \in N$;   ▷ $L[p]$: last job in the schedule associated with p
3:     **initialize** $F[r_{\min}] \leftarrow 0; F[p] \leftarrow +\infty, p \in N \setminus \{r_{\min}\}$;
4:     **for all** $p \in N$ in topological order **do**
5:         **if** $F[p] < F[p + 1]$ **then** $F[p + 1] \leftarrow F[p]; L[p + 1] \leftarrow L[p]$;   ▷ idle time
6:         **for all** $(p, p + p_j, j) \in A_j, j \in J \setminus L[p]$ **do**
7:             $q \leftarrow p + p_j; c \leftarrow w_j q - \lambda_j$;
8:             **if** $F[p] + c \leq F([q]$ **then** $F[q] \leftarrow F[p] + c; \quad L[q] \leftarrow j$
9:     **return** $F[T] - \lambda_0$

4.6   Column generation solution method

The methods that we detailed in the previous sections are integrated together in the CG Algorithm 9. Firstly, we iteratively invoke a heuristic approach (step 7) until it fails in finding columns with negative reduced cost, then we invoke an exact method (step 9). This configuration was chosen based on preliminary experiments.

Algorithm 9 Column generation - solution method

1: **procedure** CG
2:     **initialize** $S' \subseteq S$ with an initial subset of schedules;
3:     **initialize** $heur \leftarrow true$;
4:     **repeat**
5:         **solve** $MP(S')$ and obtain λ^*;
6:         **if** $heur = true$ **then**
7:             $s' \leftarrow$ pricing_heuristic();   ▷ returns the most negative column found by the heuristic
8:             **if** $s' = \emptyset$ **then** $heur \leftarrow false$;
9:         **if** $heur = false$ **then** $s' \leftarrow$ pricing_exact();   ▷ returns the most negative column
10:         $S' \leftarrow S' \cup s'$;
11:     **until** $s' = \emptyset$
12:     **return** λ^*

4.7   Branching strategy

Once the CG termination condition is attained, the optimal solution of MP may be either integer of fractional. If it is integer, it is also optimal for SC (15)-(18) and the process terminates. If instead it is fractional, branching has to be performed. The branching rule is an important decision to be taken in a B&P framework. In a classical branching rule, a fractional variable π_s is chosen and two new branches are obtained by fixing $\pi_s = 1$ and $\pi_s = 0$, respectively. Each π_s variable is associated with a schedule, and forbidding it to be generated during pricing is not easy to be managed without increasing the pricing complexity. To tackle this issue, a branching rule based on the variables of AF formulation (6)–(10) has been proposed in van den Akker *et al.* (1999). We adopt the same rule.

Let π^* be an optimal MP solution, with π_s^* giving the value taken by each variable π_s. If π^* is fractional we check if it satisfies *Theorem 2* in van den Akker *et al.* (1999), which presents a set of conditions that make it possible to transform a fractional solution into an integer feasible solution having the same cost. If the theorem is not satisfied the process is not finished and branching is needed. Let $C_j(s)$ represent the completion time of job j in schedule s. We denote by $a_j = \min\{C_j(s)|\pi_s^* > 0, a_j^s \geq 1\}$ the minimal completion time of j in the schedules of π^*, and $b_j = \max\{C_j(s)|\pi_s^* > 0, a_j^s \geq 1\}$ the maximal completion time. Let also $\bar{J} = \{j \in J | a_j \neq b_j\}$. Set $\bar{J}$ contains jobs that can be selected for branching. We select the job $j \in \bar{J}$ that maximizes $(b_j - a_j)$. Then, we create two branches:

(i) in the first branch, we forbid arcs $(p, q, j) \in A_j$ with $q \leq \lfloor \frac{a_j + b_j}{2} \rfloor$;

(ii) in the second branch we forbid arcs $(p, q, j) \in A_j$ with $q \geq \lfloor \frac{a_j + b_j}{2} \rfloor + 1$.

In practice, the procedure consists in creating new release dates and deadlines on the jobs. Notably, it does not change the complexity of the algorithms given in Section 4.2.

Another important decision in a B&P algorithm concerns the choice of the branching node. In our framework, the best bound strategy is adopted: the node with the smallest lower bound among the open nodes is chosen for branching. The main advantage of this strategy is that the lower bound of the B&B tree increases quickly. However, finding feasible solutions and closing the gap can be difficult.

5   Computational experiments

In this section, we present the computational experiments carried out to evaluate the performance of the different proposed methods for the $P|r_j| \sum w_j C_j$. These methods are the MILP formulations TI (2)-(5) and AF (6)-(10) and the B&P methods implemented to solve the SC formulation (15)-(18). The B&P methods differ one another by the algorithms (described in Section 4.2) employed to solve the CG pricing subproblem.

The methods have been coded in C++ and Gurobi Optimizer 8.1 has been adopted as MILP solver imposing it to run on a single thread. Our experiments were run on a

computer equipped with an Intel Xeon E3-1245 processor with 3.50GHz and 32 GB of RAM, under Ubuntu 16.04.5 LTS operation system. We first detail the instance generation procedure, then present the method used to generate initial feasible solutions, and finally present and discuss the obtained results.

5.1 Benchmark instances

To the best of our knowledge, there are no publicly available benchmark instances from the literature regarding the $P|r_j|\sum w_j C_j$, so we decided to generate a new set of instances. This set was created by generalizing the scheme previously proposed by Nessah *et al.* (2008). The processing times p_j and weights w_j of each job j were generated from the uniform distributions $U[1, p_{\max}]$ and $U[1, w_{\max}]$, respectively. Regarding the release dates of the jobs, these were created from the uniform distribution $U[0, \frac{1}{m}(\alpha\, n\, 50.5)]$, were n represents the number of jobs, m the number of machines and α is a parameter that controls the release date tightness. Note that as the α parameter increases, the release dates become more dispersed, i.e., the problem becomes more constrained. In our generation scheme, we adopted the following values: $\alpha \in \{0.2, 0.6, 1.0, 1.5, 2.0, 3.0\}$, $p_{\max} \in \{100, 1000\}$, $w_{\max} \in \{10\}$, $n \in \{20, 50, 100, 125, 150, 200\}$ and $m \in \{2, 3, 5, 10\}$. For each combination of such parameters, 5 instances were created, resulting in a total of 1440 instances.

5.2 Upper bound

In all our tests, all exact methods have been provided with an initial valid upper bound given by the *iterated local search* (ILS) metaheuristic proposed by Kramer and Subramanian (2019). With the aim on quickly obtaining a valid upper bound, in our experiments we ran the ILS algorithm with a time limit of $n/5$ seconds or for at least one complete iteration of the inner local search phase, which is performed by a randomized variable neighborhood descent procedure.

5.3 Computational results

In this section, we present and discuss the results obtained by the proposed solution methods. For the CG based methods, if the root node is not solved to the optimality within the time limit, then the algorithm is stopped and a valid lower bound is obtained by following the method of Vanderbeck and Wolsey (1996), also employed by van den Akker *et al.* (2000). In the next subsections, we separately present the results on the instances with $p_{\max} = 100$ and $p_{\max} = 1000$, because this parameter has a relevant impact on the algorithmic performance.

5.3.1 Instances with $p_{\max} = 100$

First, we evaluate the exact pricing methods by running the CG algorithm with the three exact methods for solving the subproblems. In Table 1, these methods are confronted

by means of average execution times, $t(s)$, number of generated columns, *cols*, and root relaxation lower bound, lb_{lp}, per each group of 30 instances with up to 100 jobs and 10 machines. The last line provide overall average values.

Table 1: Comparison of exact methods for solving the pricing subproblem (best results in boldface) – instances with $p_{\max} = 100$

n	m	LC_A			LC_P			LC_{2l}		
		lb_{lp}	cols	$t(s)$	lb_{lp}	cols	$t(s)$	lb_{lp}	cols	$t(s)$
20	2	49126.8	**294.9**	**0.1**	**49137.9**	355.3	0.4	**49137.9**	354.8	0.2
	3	32017.7	**207.0**	**0.0**	**32023.1**	218.1	0.2	**32023.1**	218.9	0.1
	5	22258.1	**144.1**	**0.0**	**22260.9**	144.6	0.1	**22260.9**	144.9	0.0
	10	13092.9	78.1	**0.0**	**13094.0**	70.5	0.0	**13094.0**	**70.2**	0.0
50	2	269913.4	**2027.8**	**5.3**	269987.3	3094.4	58.6	269987.3	3148.8	16.0
	3	182051.8	**1336.9**	**2.6**	182085.0	2067.0	27.7	182085.0	2041.2	7.6
	5	115666.4	**1003.0**	**1.3**	115675.4	1396.0	11.8	115675.4	1399.6	3.5
	10	63809.8	**618.0**	**0.5**	63813.9	729.0	3.0	63813.9	729.5	0.9
100	2	1072592.8	**7759.6**	**135.8**	1072764.3	14351.7	2385.7	1072764.3	14370.8	581.5
	3	711303.4	**5319.6**	**64.0**	711370.5	11676.3	1982.4	711370.5	12054.5	893.6
	5	426377.6	**3224.3**	**23.7**	426398.8	5562.0	342.9	426398.8	5771.0	81.9
	10	238179.3	**2131.6**	**10.1**	238182.7	3179.4	111.3	238182.7	3185.9	26.6
Avg.		266365.8	**2012.1**	**20.3**	266399.5	3570.4	410.3	266399.5	3624.2	134.3

From Table 1, it can be noticed that the CG methods with LC_P and LC_{2l} always provide bounds equal to or better than the ones obtained with LC_A. This result is expected and explained by the fact that LC_P and LC_{2l} solve the SPPRC without one-cycles as pricing subproblem, whereas LC_A solves the SPPRC itself. Thus, the solution space for the first problem is included in the solution space of the second, hence resulting in a possible stronger bound. Considering the number of columns generated, the CG with LC_A is the method that requires less columns to prove the relaxation optimality. This results is also reflected in the execution times, that are, on average, the smallest ones among the three approaches. Regarding LC_P and LC_{2l}, the number of columns they generate are very similar, the execution times, on the contrary, are quite different, indicating the effectiveness of LC_{2l}. The CG algorithm with LC_{2l} required, on average, 134.3 seconds to solve the selected instances, whereas the CG with LC_P needed 410.3 seconds. Therefore, one could select LC_A as a method to solve the pricing if the objective is to quickly obtain a valid bound, and choose LC_{2l} if the goal is to obtain stronger bounds.

In Table 2, we evaluate the impact of using heuristic approaches for solving the pricing subproblems so as to speed up the CG convergence. We report the results for the combinations of LC_A and LC_{2l} with $H_{\text{1-cycle}}$ and H_{elem}, resulting in four methods. For each method and group of instances, we report the average execution times, $t(s)$, the total number of generated columns, *cols*, and the average percentage number of columns generated by the exact methods, $\%cols_e = 100*(cols_{\text{exact}}/cols)$, where $cols_{\text{exact}}$ represents the number of columns generated by the considered exact algorithm.

The results shown in Table 2 indicate that the performance of H_{elem} is, on average, better than the one by $H_{\text{1-cycle}}$, because of the reduced number of columns generated as well as the smaller average execution times. For example, the CG algorithm required, on average, 3101.8 columns and 48 seconds to solve the problem when using LC_{2l} and

Table 2: Impact evaluation of combining heuristic and exact methods for solving the pricing subproblem (best results in boldface) – instances with $p_{\max} = 100$

n	m	LC_A						LC_{2l}					
		$H_{\text{1-cycle}}$			H_{elem}			$H_{\text{1-cycle}}$			H_{elem}		
		cols	%cols$_e$	$t(s)$	cols	%cols$_e$	$t(s)$	cols	%cols$_e$	$t(s)$	cols	%cols$_e$	$t(s)$
20	2	323.2	11.0	0.1	161.7	20.1	0.1	292.9	**1.6**	0.1	**138.2**	8.2	**0.1**
	3	203.7	6.2	0.0	102.9	10.2	0.0	193.5	**1.4**	0.0	**94.2**	3.4	**0.0**
	5	132.3	2.9	0.0	73.4	3.8	0.0	129.5	**0.8**	0.0	**71.7**	1.9	**0.0**
	10	67.7	1.8	0.0	41.1	3.4	**0.0**	67.4	**1.5**	0.0	**40.6**	2.6	0.0
50	2	3163.0	13.7	11.0	1318.7	31.6	3.1	2885.0	**4.3**	10.2	**1061.8**	15.9	**2.7**
	3	1974.5	6.9	5.1	651.0	14.6	1.2	1892.3	**2.1**	5.0	**583.4**	5.6	**1.1**
	5	1230.8	3.5	2.1	445.3	6.2	0.6	1202.8	**0.4**	2.0	**415.3**	0.8	**0.6**
	10	641.3	1.7	0.5	338.7	1.8	0.3	633.8	**0.2**	0.5	**333.0**	0.3	**0.3**
100	2	13947.7	17.5	354.1	6244.0	36.8	92.1	12787.2	**10.2**	321.1	**5334.9**	27.0	**78.9**
	3	9510.9	9.5	167.3	3064.0	22.6	24.3	8975.6	**3.5**	160.0	**2624.4**	10.6	**21.6**
	5	5388.9	3.9	59.4	1564.7	8.2	7.9	5296.1	**1.8**	58.8	**1476.9**	3.7	**7.6**
	10	2884.7	1.0	18.3	1201.6	1.5	4.3	2865.9	**0.2**	18.2	**1185.8**	0.4	**4.2**
Avg.		3289.1	6.6	51.5	1267.3	13.4	11.1	3101.8	**2.3**	48.0	**1113.4**	6.7	**9.8**

$H_{\text{1-cycle}}$. By keeping LC_{2l} but replacing $H_{\text{1-cycle}}$ by H_{elem}, the average number of generated columns is reduced by more than 60% and the average execution time drops to only 9.8 seconds. In other words, H_{elem} seems to be able to found "better" columns than $H_{\text{1-cycle}}$, thus speeding up the convergence at the beginning of the CG algorithm. From Table 2, it can also be observed that most of the columns are generated by the heuristic methods.

By comparing the results in Table 2 with the ones in Table 1, it can be noticed the effectiveness of the combination of exact and heuristic methods in the CG pricing problem. Taking as example the results obtained by the CG with LC_{2l} in Table 1, and the results by the CG with LC_{2l} and H_{elem} in Table 2, it can be seen that the average number of generated columns drops from 3624.2 to only 1113.4. The same occurs with the average execution times that passes from 134.3 to 9.8 seconds.

Considering the above results, we thus adopted the combined use of LC_{2l} and H_{elem} as the method for solving the pricing subproblem in our CG algorithm, and embedded them into our B&P framework, as described in Section 4.7. In the following, the resulting method is named $B\&P_h$. In Tables 3 and 4, $B\&P_h$ is compared with the TI (2)-(5) and AF (6)-(10) formulations in terms of number of instances optimally solved, opt, percentage gap for the continuous and for the best lower bound, $gap_{lp}(\%)$ and $gap(\%)$, respectively, number of open nodes, nd, and execution times, $t(s)$. The results are aggregated by n and m, where for each combination of (n, m), six instances (one for each value of $\alpha \in \{0.2, 0.6, 1.0, 1.5, 2.0, 3.0\}$) have been considered. For each run of TI, AF and $B\&P_h$, a time limit of 1800 seconds has been imposed. The last line gives the sum of the opt values, as well as average values for the other columns.

With regard to the results obtained for the instances with up to 100 jobs, in Table 3, it can be noticed that TI and AF perform better than $B\&P_h$ in terms of number of instances solved to the proven optimality and execution times. The $B\&P_h$ algorithm, however, provides stronger continuous relaxation bounds and requires the exploration of a smaller amount of nodes. The results for the large-sized instances with $n > 100$ are given in Table 4. For the instances with 200 jobs and 2 machines, TI and AF suffer from

Table 3: Comparison of B&P$_h$ performance with MILP models TI and AF - instances with $p_{\max} = 100$ and $n \in \{20, 50, 100\}$ (best results in boldface)

n	m	TI					AF					B&P$_h$				
		opt	gap_{lp} (%)	gap (%)	nd	$t(s)$	opt	gap_{lp} (%)	gap (%)	nd	$t(s)$	opt	gap_{lp} (%)	gap (%)	nd	$t(s)$
20	2	**6**	0.083	**0.000**	**0.2**	1.7	**6**	0.083	**0.000**	**0.2**	0.3	**6**	**0.055**	**0.000**	0.7	**<0.1**
	3	**6**	0.033	**0.000**	0.3	1.2	**6**	0.033	**0.000**	**0.2**	0.2	**6**	**0.023**	**0.000**	1.0	**<0.1**
	5	**6**	0.068	**0.000**	**0.3**	0.6	**6**	0.068	**0.000**	**0.3**	0.1	**6**	**0.054**	**0.000**	1.3	**<0.1**
	10	**6**	**0.000**	**0.000**	**0.0**	0.2	**6**	**0.000**	**0.000**	**0.0**	**<0.1**	**6**	**0.000**	**0.000**	**0.0**	**<0.1**
50	2	**6**	0.092	**0.000**	938.0	48.3	**6**	0.092	**0.000**	573.2	**19.6**	4	**0.059**	0.007	**80.8**	610.3
	3	**6**	0.083	**0.000**	662.2	24.8	**6**	0.083	**0.000**	859.2	19.4	**6**	**0.054**	**0.000**	**46.5**	44.4
	5	**6**	0.037	**0.000**	221.5	9.3	**6**	0.037	**0.000**	267.0	3.6	**6**	**0.026**	**0.000**	**58.3**	11.3
	10	**6**	0.007	**0.000**	2.2	2.6	**6**	0.007	**0.000**	1.7	0.4	**6**	**0.003**	**0.000**	9.5	**0.4**
100	2	**4**	0.314	0.256	3396.3	**863.6**	3	0.314	0.256	7385.8	910.8	2	**0.264**	**0.215**	**22.3**	1226.8
	3	**4**	0.081	0.055	1167.0	734.6	**4**	0.081	0.055	2675.5	**710.5**	3	**0.065**	**0.035**	50.7	931.3
	5	5	0.042	0.005	7010.0	**383.9**	5	0.042	**0.002**	21661.3	450.1	3	**0.037**	0.005	**199.3**	907.4
	10	**6**	0.013	**0.000**	239.8	22.7	**6**	0.013	**0.000**	252.2	9.0	4	**0.009**	0.002	611.0	607.1
Sum/Avg.		**67**	0.071	0.026	1136.5	174.5	66	0.071	0.026	2806.4	**177.0**	58	**0.054**	**0.022**	**90.1**	361.6

memory limit, indicated by the entry "*m.lim*", due to their large (pseudo-polynomial) number of variables and constraints. We thus provide two overall Sum/Avg. lines, one referring to all instances and the other only to those for which no memory limit was encountered by a method. Summarizing, on one side B&P$_h$ is an interesting solution method because it does not suffer from memory limits and provides very low gaps. On the other side, TI and AF are very efficient and find a higher number of proven optimal solutions overall.

Table 4: Comparison of B&P$_h$ performance with MILP models TI and AF- instances with $p_{\max} = 100$ and $n \in \{125, 150, 200\}$ (best results in boldface)

n	m	TI					AF					B&P$_h$				
		opt	gap_{lp} (%)	gap (%)	nd	$t(s)$	opt	gap_{lp} (%)	gap (%)	nd	$t(s)$	opt	gap_{lp} (%)	gap (%)	nd	$t(s)$
125	2	**4**	0.361	**0.338**	2852.0	**774.0**	**4**	0.361	0.341	2831.0	787.7	2	**0.343**	0.339	**10.3**	1303.1
	3	**4**	0.192	0.175	83.8	642.2	**4**	0.192	0.178	597.5	**612.8**	3	**0.176**	**0.169**	**35.3**	943.8
	5	**4**	0.084	0.054	1846.2	**878.1**	3	0.084	0.056	2397.0	902.7	3	**0.070**	**0.043**	141.2	913.2
	10	**6**	0.015	**0.000**	2638.8	138.4	**6**	0.015	**0.000**	3265.2	**91.9**	3	**0.012**	0.002	**700.0**	905.4
150	2	**3**	0.287	0.278	**2.0**	965.0	**3**	0.287	0.281	277.5	**910.0**	1	**0.278**	**0.277**	3.7	1568.9
	3	**3**	0.436	0.428	21.3	942.1	**3**	0.436	0.426	577.5	906.0	**3**	**0.425**	**0.385**	**20.0**	1329.1
	5	**3**	0.280	0.274	854.5	919.9	**3**	0.280	0.273	2561.3	902.6	**3**	**0.276**	**0.268**	**73.8**	924.4
	10	**6**	0.011	**0.000**	586.5	260.5	**6**	0.011	**0.000**	1244.2	285.8	4	**0.009**	0.003	**307.8**	807.1
200	2	0				m.lim	0				m.lim	0	0.290	0.290	0.5	t.lim
	3	3	0.242	0.236	229.7	988.4	3	0.242	0.238	1505.8	**932.4**	2	**0.234**	**0.230**	**3.0**	1377.5
	5	**3**	0.243	0.238	**0.7**	941.1	**3**	0.243	0.240	752.0	905.2	**3**	**0.238**	**0.234**	25.5	1073.2
	10	**4**	0.047	0.043	470.5	639.0	**4**	0.047	0.042	1424.8	**608.0**	3	**0.045**	**0.040**	116.3	947.6
Sum/Avg.*		**43**	0.200	0.188	871.5	735.3	42	0.200	0.189	1584.9	**713.2**	30	**0.191**	**0.181**	**130.6**	1099.4
Sum/Avg.												30	**0.200**	**0.190**	**119.8**	1157.8

*Sum and average values for all instances, but the ones with $(n, m) \in \{(200, 2)\}$

5.3.2 Instances with $p_{\max} = 1000$

In this section, we replicate the experiments described in Section 5.3.1, but now considering the instances with $p_{\max} = 1000$. In Table 5, we evaluate the performance of the three exact algorithms employed to solve the pricing subproblem. The results we obtained are aligned with the ones reported in Table 1 for $p_{\max} = 100$, thus confirming the performance gain achieved when replacing LC_P by LC_{2l}.

In Table 6, we evaluate the combined use of heuristic and exact methods in the CG.

Table 5: Comparison of exact methods for solving the pricing subproblem (best results in boldface) – instances with $p_{max} = 1000$

n	m	LC_A			LC_P			LC_{2l}		
		lb_{lp}	cols	$t(s)$	lb_{lp}	cols	$t(s)$	lb_{lp}	cols	$t(s)$
20	2	195006.7	**229.1**	**0.1**	**195141.0**	232.0	1.5	**195141.0**	232.5	0.5
	3	148947.7	**171.1**	**0.1**	**149007.6**	176.4	0.8	**149007.6**	176.4	0.2
	5	102597.2	123.9	0.0	**102599.0**	121.4	0.3	**102599.0**	**120.9**	0.1
	10	71462.7	76.0	0.0	**71463.8**	74.3	0.1	**71463.8**	**73.4**	0.0
50	2	1049597.4	**1425.1**	**4.8**	**1050223.8**	1648.2	207.8	**1050223.8**	1637.4	18.3
	3	799434.9	**952.1**	**2.3**	**799746.8**	1075.5	91.2	**799746.8**	1084.7	8.5
	5	523452.1	**647.4**	**1.0**	**523544.7**	693.8	34.7	**523544.7**	700.9	3.4
	10	285946.9	**399.6**	**0.4**	**285969.7**	422.8	11.0	**285969.7**	420.5	1.2
100	2	4173231.9	**6914.0**	**136.0**	**4174522.1**	8231.0	8930.3	**4174522.1**	8401.6	432.7
	3	2842626.6	**3797.9**	**41.0**	**2843104.3**	4421.7	3127.0	**2843104.3**	4369.0	135.8
	5	1765972.9	**2170.3**	**15.1**	**1766155.7**	2552.4	1166.9	**1766155.7**	2531.9	52.8
	10	994029.4	**1297.6**	**4.9**	**994066.5**	1435.1	353.8	**994066.5**	1430.9	16.3
Avg.		1079358.9	**1517.0**	**17.1**	**1079628.7**	1757.1	1160.4	**1079628.7**	1765.0	55.8

It can be seen that, differently from the results for the instances with $p_{max} = 100$, the use

Table 6: Impact evaluation of combining heuristic and exact methods for solving the pricing subproblem (best results in boldface) – instances with $p_{max} = 1000$

n	m	LC_A						LC_{2l}					
		$H_{1\text{-cycle}}$			H_{elem}			$H_{1\text{-cycle}}$			H_{elem}		
		cols	%cols$_e$	$t(s)$	cols	%cols$_e$	$t(s)$	cols	%cols$_e$	$t(s)$	cols	%cols$_e$	$t(s)$
20	2	265.2	14.7	0.2	218.9	21.2	0.3	229.5	**2.0**	**0.1**	**188.1**	8.3	0.3
	3	180.2	8.0	0.1	145.2	11.7	0.2	165.7	**0.7**	**0.1**	**132.1**	3.4	0.2
	5	122.8	3.7	0.0	97.6	6.2	0.1	118.9	**0.9**	0.0	**93.1**	1.9	0.1
	10	71.0	2.8	**0.0**	58.9	2.9	0.0	70.1	**1.5**	0.0	**58.2**	1.7	0.0
50	2	2093.8	25.6	7.1	1743.0	33.6	16.2	1691.9	**8.2**	**6.5**	**1451.8**	20.0	17.1
	3	1241.2	14.7	3.1	1011.4	18.6	7.6	1095.0	**3.4**	**2.9**	**900.0**	8.7	7.8
	5	734.4	6.5	1.1	580.5	8.0	2.9	691.3	**0.8**	**1.1**	**549.5**	3.0	2.9
	10	423.5	2.7	0.4	346.2	3.2	1.0	413.2	**0.3**	**0.4**	**338.0**	0.9	1.0
100	2	10732.0	40.8	243.2	10461.6	49.4	488.8	**8684.4**	26.8	**242.8**	9624.9	44.2	563.0
	3	5523.0	24.0	66.0	4462.1	30.9	135.6	4522.2	**7.3**	**57.4**	**3698.6**	17.0	134.7
	5	2832.7	11.3	20.3	2226.4	14.7	45.5	2583.0	**2.8**	**19.0**	**2017.1**	5.8	45.4
	10	1470.0	3.6	5.3	1244.8	4.5	14.3	1422.0	**0.4**	**5.1**	**1204.3**	1.3	14.3
Avg.		2140.8	13.2	28.9	1883.1	17.1	59.4	1807.3	**4.6**	**27.9**	**1688.0**	9.7	65.6

of LC_{2l} combined with $H_{1\text{-cycle}}$ as heuristic method turns out to be the best alternative with regard to execution time. This result can be explained by the fact that $H_{1\text{-cycle}}$ runs in $\mathcal{O}(nT)$ time while H_{elem} needs $\mathcal{O}(n^2T)$ time. However, it can also be observed that by using H_{elem} as heuristic approach the number of required columns to prove the optimality is smaller than when $H_{1\text{-cycle}}$ is used. Hence, there exists a trade-off between execution times and number of generated columns.

In the next experiments, reported in Tables 7 and 8, we thus decided to test both B&P methods that use LC_{2l} with H_{elem} and LC_{2l} with $H_{1\text{-cycle}}$. The two B&P methods are labeled as B&P$_h$ for H_{elem} and B&P$_1$ for $H_{1\text{-cycle}}$. For the instances with up to 100 jobs, it can be observed that TI is not able to manage instances with more than 50 jobs due to memory limitations, while AF performs better, being able to tackle instances with up to 100 jobs, especially when m is large. The B&P methods in turn, solve instances with up to 100 jobs, but with average execution times slightly better than those of AF.

Concerning the experiments on the larger instances, TI reached memory limit on all of them and was consequently disregarded from Table 8. From this table, it can be

Table 7: Comparison of B&P$_h$ and B&P$_1$ performance with MILP models TI and AF - instances with $p_{\max} = 1000$ and $n \in \{20, 50, 100\}$ (best results in boldface)

n	m	TI					AF					B&P$_h$					B&P$_1$				
		opt	gap_{lp} (%)	gap (%)	nd	$t(s)$	opt	gap_{lp} (%)	gap (%)	nd	$t(s)$	opt	gap_{lp} (%)	gap (%)	nd	$t(s)$	opt	gap_{lp} (%)	gap (%)	nd	$t(s)$
20	2	6	0.328	**0.000**	0.7	617.6	6	0.328	**0.000**	**0.5**	13.9	6	**0.091**	**0.000**	3.7	0.7	6	**0.091**	**0.000**	3.3	**0.3**
	3	6	**0.000**	**0.000**	0.0	280.5	6	**0.000**	**0.000**	0.0	5.4	6	**0.000**	**0.000**	0.0	0.2	6	**0.000**	**0.000**	0.0	**<0.1**
	5	6	**0.000**	**0.000**	0.0	148.6	6	**0.000**	**0.000**	0.0	2.7	6	**0.000**	**0.000**	0.0	**<0.1**	6	**0.000**	**0.000**	0.0	**<0.1**
	10	6	**0.000**	**0.000**	0.0	67.2	6	**0.000**	**0.000**	0.0	1.1	6	**0.000**	**0.000**	0.0	**<0.1**	6	**0.000**	**0.000**	0.0	**<0.1**
50	2	0				m.lim	3	0.122	0.057	56.5	946.9	6	0.079	**0.000**	8.5	81.5	6	0.079	**0.000**	8.3	**37.5**
	3	0				m.lim	6	0.074	**0.000**	40.3	179.2	5	**0.021**	0.006	50.2	317.8	6	**0.021**	**0.000**	24.3	**60.2**
	5	4	0.044	**0.030**	**0.8**	1515.1	6	0.044	**0.000**	441.2	238.6	6	**0.026**	**0.000**	33.5	18.3	6	**0.026**	**0.000**	29.0	**8.3**
	10	6	0.002	**0.000**	0.3	531.5	6	0.002	**0.000**	0.3	7.7	6	**0.001**	**0.000**	2.7	1.3	6	**0.001**	**0.000**	4.0	**0.5**
100	2					m.lim	1	0.167	0.161	9.2	**1635.2**	0	**0.142**	0.132	9.0	t.lim	0	**0.142**	**0.131**	12.2	t.lim
	3					m.lim	1	0.102	0.100	**13.7**	1556.8	0	**0.093**	0.064	33.3	t.lim	0	**0.093**	**0.062**	45.0	t.lim
	5					m.lim	1	0.095	0.073	**72.0**	1523.5	1	**0.079**	**0.056**	194.2	1558.8	2	**0.079**	**0.056**	227.2	**1314.5**
	10					m.lim	5	0.012	**<0.001**	163.2	637.0	4	**0.007**	0.001	748.2	719.0	4	**0.007**	0.001	820.7	643.5
Sum/Avg.*		34	0.062	0.005	0.3	526.8	36	0.062	**0.000**	73.7	44.9	36	**0.020**	**0.000**	6.6	3.4	36	**0.020**	**0.000**	6.1	**1.5**
Sum/Avg.							53	0.079	0.033	66.4	562.3	52	**0.045**	0.022	78.8	524.1	54	**0.045**	**0.021**	97.8	**472.1**

*Sum and average results for all instances, but the ones with $(n, m) \in \{(50, 2), (50, 3), (100, 2), (100, 3), (100, 5), (100, 10)\}$

Table 8: Comparison of B&P$_h$ and B&P$_1$ performance with AF model - instances with $p_{\max} = 1000$ and $n \in \{125, 150\}$ (best results in boldface)

n	m	AF					B&P$_h$					B&P$_1$				
		opt	gap_{lp} (%)	gap (%)	nd	$t(s)$	opt	gap_{lp} (%)	gap (%)	nd	$t(s)$	opt	gap_{lp} (%)	gap (%)	nd	$t(s)$
125	2	0	16.885	16.884	4.0	t.lim	0	**0.209**	0.209	**1.2**	t.lim	0	**0.209**	**0.207**	2.2	t.lim
	3	0	0.176	0.173	**7.3**	t.lim	0	**0.157**	0.141	14.3	t.lim	0	**0.157**	**0.139**	21.8	t.lim
	5	1	0.141	0.140	**11.3**	1594.1	1	**0.138**	0.128	78.3	1634.6	0	**0.138**	**0.128**	114.3	t.lim
	10	2	0.080	0.078	**68.2**	1304.6	2	**0.072**	**0.045**	583.5	1388.8	2	**0.072**	**0.045**	793.5	1365.9
150	2	0				m.lim	0	0.282	0.282	**0.2**	t.lim	0	**0.281**	**0.253**	0.7	t.lim
	3	0				m.lim	0	0.185	0.184	**4.7**	t.lim	0	**0.185**	**0.180**	8.3	t.lim
	5	0				m.lim	1	0.224	0.212	**43.3**	1777.5	0	0.224	**0.211**	74.5	t.lim
	10	0				m.lim	0	0.097	0.094	**547.0**	t.lim	0	0.097	0.094	681.7	t.lim
200	2	0				m.lim	0	0.405	0.405	0.0	t.lim	0	**0.341**	**0.341**	**0.0**	t.lim
	3	0				m.lim	0	0.205	0.205	0.0	t.lim	0	**0.199**	**0.199**	**0.2**	t.lim
	5	0				m.lim	0	0.272	0.267	**9.5**	t.lim	0	0.272	**0.266**	19.2	t.lim
	10	0				m.lim	0	**0.211**	**0.207**	152.2	t.lim	0	**0.211**	**0.207**	215.2	t.lim
Sum/Avg.*		3	4.321	4.318	**23.5**	1624.7	3	**0.144**	0.131	169.3	1655.8	2	**0.144**	**0.130**	233.0	1691.5
Sum/Avg.							4	0.205	0.198	**119.5**	**1750.1**	2	**0.199**	**0.189**	161.0	1763.8

*Sum and average results for the instances with $n = 125$

noticed that for instances with 125 jobs, AF, B&P$_h$ and B&P$_1$ have a similar performance. However, AF was not able to tackle the instances with 150 jobs due to memory limit. In these cases, the B&P methods are a more robust alternative, being able to prove the optimality of one instance and to provide very low gaps for the other ones, always below 0.4% on each group of instances. In particular, B&P$_h$ performs better than B&P$_1$ on the larger instances, solving 4 instances to the proven optimality instead of 2.

6 Concluding remarks

In this work, we investigated the problem of scheduling a set of jobs that are released over the time on a number of identical parallel machines, with the aim of minimizing the total weighted completion time. This problem, denoted as $P|r_j| \sum w_j C_j$, is of high interest because can model several applications in practice. We proposed two MILP formulations, a time-indexed and an arc-flow, and a tailored B&P algorithm. Our experiments showed that for small and medium-sized instances all attempted methods performed quite well, with the arc-flow model achieving the best performance in terms of number of proven optimal solutions. Regarding the large-sized instances involving more than 200 jobs, or

instances with large processing times, we notice that the MILP formulations suffers from memory limit due to their pseudo-polynomial sizes, whereas the B&P remains a robust and interesting alternative being able to provide very low gaps in a reasonable amount of time. Using the proposed methods we were able to solve for the first time instances with up to 200 jobs to the proven optimality.

Concerning future research direction we observe that a large number of instances, especially involving large processing times, remain unsolved, thus leaving space for further investigation. Moreover, we envisage the need of a deeper study of the proposed models aiming to reduce their sizes by considering combinatorial properties of the problem, as well as the use of advanced concepts for the design of the B&P algorithm, such as, the introduction of cutting planes, the use of stabilization techniques, the implementation of an early termination approach for the CG algorithm and the use of a strong branching scheme. The developed techniques could be also adapted to tackle other problem variants, especially those involving due dates and the minimization of (weighted) tardiness, lateness or just-in-time objective functions.

Acknowledgments

References

Allahverdi, A. (2015), The third comprehensive survey on scheduling problems with setup times/costs. *European Journal of Operational Research*, v. 246, n. 2, p. 345 – 378.

Avella, P., Boccia, M. and D'Auria, B. (2005), Near-optimal solutions of large-scale single-machine scheduling problems. *INFORMS Journal on Computing*, v. 17, n. 2, p. 183–191.

Baptiste, P., Brucker, P., Chrobak, M., Dürr, C., Kravchenko, S. A. and Sourd, F. (2007), The complexity of mean flow time scheduling problems with release times. *Journal of Scheduling*, v. 10, n. 2, p. 139–146.

Baptiste, P., Jouglet, A. and Savourey, D. (2008), Lower bounds for parallel machine scheduling problems. *International Journal of Operational Research*, v. 3, n. 6, p. 643–664.

Belouadah, H., Posner, M. and Potts, C. (1992), Scheduling with release dates on a single machine to minimize total weighted completion time. *Discrete Applied Mathematics*, v. 36, n. 3, p. 213 – 231.

Brucker, P. and Kravchenko, S. A. Polynomial algorithm for parallel machine mean flow time scheduling problem with release dates. Gervasi, O., Gavrilova, M. L., Kumar, V., Laganá, A., Lee, H. P., Mun, Y., Taniar, D. and Tan, C. J. K. (Eds.), *Computational Science and Its Applications – ICCSA 2005*, p. 182–191, Berlin, Heidelberg. Springer Berlin Heidelberg, 2005.

Côté, J.-F. and Iori, M. (2018), The meet-in-the-middle principle for cutting and packing problems. *INFORMS Journal on Computing*, v. 30, n. 4, p. 646–661.

Delorme, M., Iori, M. and Martello, S. (2016), Bin packing and cutting stock problems: Mathematical models and exact algorithms. *European Journal of Operational Research*, v. 255, n. 1, p. 1–20.

Delorme, M., Iori, M. and Martello, S. (2017), Logic based Benders' decomposition for orthogonal stock cutting problems. *Computers & Operations Research*, v. 78, p. 290–298.

Demeulemeester, E. and Herroelen, W. *Project Scheduling: A Research Handbook*. International Series in Operations Research & Management Science. Kluwer Academic Publishers, 2002.

Du, J., Leung, J. Y.-T. and Young, G. H. (1990), Minimizing mean flow time with release time constraint. *Theoretical Computer Science*, v. 75, n. 3, p. 347 – 355.

Dyer, M. E. and Wolsey, L. A. (1990), Formulating the single machine sequencing problem with release dates as a mixed integer program. *Discrete Applied Mathematics*, v. 26, n. 2, p. 255 – 270.

Feillet, D. (2010), A tutorial on column generation and branch-and-price for vehicle routing problems. *4OR*, v. 8, n. 4, p. 407–424.

Feillet, D., Dejax, P., Gendreau, M. and Gueguen, C. (2004), An exact algorithm for the elementary shortest path problem with resource constraints: Application to some vehicle routing problems. *Networks*, v. 44, n. 3, p. 216–229.

Graham, R., Lawler, E., Lenstra, J. and Kan, A. Optimization and approximation in deterministic sequencing and scheduling: a survey. P.L. Hammer, E. J. and Korte, B. (Eds.), *Discrete Optimization II Proceedings of the Advanced Research Institute on Discrete Optimization and Systems Applications*, volume 5 of *Annals of Discrete Mathematics*, p. 287 – 326. Elsevier, 1979.

Hall, L. A., Schulz, A. S., Shmoys, D. B. and Wein, J. (1997), Scheduling to minimize average completion time: Off-line and on-line approximation algorithms. *Mathematics of Operations Research*, v. 22, n. 3, p. 513–544.

Hall, R. W. *Handbook of healthcare system scheduling*. International Series in Operations Research & Management Science. Springer, 2012.

Hariri, A. and Potts, C. (1983), An algorithm for single machine sequencing with release dates to minimize total weighted completion time. *Discrete Applied Mathematics*, v. 5, n. 1, p. 99 – 109.

Kowalczyk, D. and Leus, R. (2018), A branch-and-price algorithm for parallel machine scheduling using ZDDs and generic branching. *INFORMS Journal on Computing*, v. 30, n. 4, p. 768–782.

Kramer, A., Dell'Amico, M. and Iori, M. (2019a), Enhanced arc-flow formulations to minimize weighted completion time on identical parallel machines. *European Journal of Operational Research*, v. 275, n. 1, p. 67 – 79.

Kramer, A., Iori, M. and Lacomme, P. (2019b), Mathematical formulations for scheduling jobs on identical parallel machines with family setup times and total weighted completion time minimization. *European Journal of Operational Research*. . Forthcoming.

Kramer, A. and Subramanian, A. (2019), A unified heuristic and an annotated bibliography for a large class of earliness–tardiness scheduling problems. *Journal of Scheduling*, v. 22, n. 1, p. 21–57.

Lenstra, J., Kan, A. R. and Brucker, P. Complexity of machine scheduling problems. Hammer, P., Johnson, E., Korte, B. and Nemhauser, G. (Eds.), *Studies in Integer Programming*, volume 1 of *Annals of Discrete Mathematics*, p. 343 – 362. Elsevier, 1977.

Leonardi, S. and Raz, D. (2007), Approximating total flow time on parallel machines. *Journal of Computer and System Sciences*, v. 73, n. 6, p. 875 – 891.

Lübbecke, M. E. and Desrosiers, J. (2005), Selected topics in column generation. *Operations Research*, v. 53, n. 6, p. 1007–1023.

Mrad, M. and Souayah, N. (2018), An arc-flow model for the makespan minimization problem on identical parallel machines. *IEEE Access*, v. 6, p. 5300–5307.

Nessah, R., Yalaoui, F. and Chu, C. (2008), A branch-and-bound algorithm to minimize total weighted completion time on identical parallel machines with job release dates. *Computers & Operations Research*, v. 35, n. 4, p. 1176 – 1190.

Pan, Y. and Shi, L. (2008), New hybrid optimization algorithms for machine scheduling problems. *IEEE Transactions on Automation Science and Engineering*, v. 5, n. 2, p. 337–348.

Pessoa, A., Uchoa, E., de Aragão, M. P. and Rodrigues, R. (2010), Exact algorithm over an arc-time-indexed formulation for parallel machine scheduling problems. *Mathematical Programming Computation*, v. 2, n. 3, p. 259–290.

Sousa, J. P. and Wolsey, L. A. (1992), A time indexed formulation of non-preemptive single machine scheduling problems. *Mathematical Programming*, v. 54, n. 1, p. 353–367.

Tanaka, S. and Fujikuma, S. (2012), A dynamic-programming-based exact algorithm for general single-machine scheduling with machine idle time. *Journal of Scheduling*, v. 15, n. 3, p. 347–361.

Valério de Carvalho, J. (1999), Exact solution of bin-packing problems using column generation and branch-and-bound. *Annals of Operations Research*, v. 86, n. 0, p. 629–659.

van den Akker, J. M., Hoogeveen, J. A. and Van de Velde, S. L. (1999), Parallel machine scheduling by column generation. *Operations Research*, v. 47, n. 6, p. 862–872.

van den Akker, J., Hurkens, C. and Savelsbergh, M. (2000), Time-indexed formulations for machine scheduling problems: Column generation. *INFORMS Journal on Computing*, v. 12, n. 2, p. 111–124.

van den Bergh, J., Belin, J., Bruecker, P. D., Demeulemeester, E. and De Boeck, L. (2013), Personnel scheduling: A literature review. *European Journal of Operational Research*, v. 226, n. 3, p. 367 – 385.

Vanderbeck, F. and Wolsey, L. A. (1996), An exact algorithm for IP column generation. *Operations Research Letters*, v. 19, n. 4, p. 151 – 159.

Yalaoui, F. and Chu, C. (2006), New exact method to solve the $Pm|r_j|\sum C_j$ schedule problem. *International Journal of Production Economics*, v. 100, n. 1, p. 168 – 179.