Cadernos do LOGIS

On the exact solution of a large class of parallel machine scheduling problems

Teobaldo Bulhões, Ruslan Sadykov,
Anand Subramanian, Eduardo Uchoa

Volume 2018, Number 3

October, 2018

On the exact solution of a large class of parallel machine scheduling problems

Teobaldo Bulhões[a,*], Ruslan Sadykov[b], Anand Subramanian[a], Eduardo Uchoa[d]

[a] *Departamento de Computação, Centro de Informática, Universidade Federal da Paraíba, João Pessoa, Brazil*
[b] *Inria Bordeaux — Sud-Ouest, France*
[c] *Institut de Mathématiques de Bordeaux, Université de Bordeaux, France*
[d] *Departamento de Engenharia de Produção, Universidade Federal Fluminense, Brazil*

Abstract

This work deals with a very generic class of scheduling problems with identical/uniform/unrelated parallel machine environment. It considers well-known attributes such as release dates or sequence-dependent setup times and accepts any objective function defined over job completion times. Non-regular objectives are also supported. We introduce a branch-cut-and-price algorithm for such problems that makes use of non-robust cuts, i.e., cuts which change the structure of the pricing problem. This is the first time that such cuts are employed for machine scheduling problems. The algorithm also embeds other important techniques such as strong branching, reduced cost fixing and dual stabilization. Computational experiments over literature benchmarks showed that the proposed algorithm is indeed effective and could solve many instances to optimality for the first time.

Keywords: Parallel machines, Unified algorithm, Branch-cut-and-price

1. Introduction

We consider a class of problems to schedule a set of jobs J ($n = |J|$) on a set of machines of different types $k \in K$ without preemption. For each machine type $k \in K$, there exists m_k machines available ($m = \sum_{k \in K} m_k$). A job j is not allowed to be processed before its release date r_j, and its processing time on a machine of type $k \in K$ is denoted as p_j^k. Also, s_{ij}^k denotes the setup time required to process job j immediately after job i on a machine of type k. In this paper, unless stated otherwise, we assume anticipatory setups, meaning that the setup operations of a job can start before its release date. Each job j is associated to a cost function $f_j(C_j)$ defined over its completion time C_j. The objective function is to minimize $\sum_{j \in J} f_j(C_j)$. This function is very general and can model many criteria. For example, suppose each job has an earliness $E_j = \max\{d_j - C_j, 0\}$ and a tardiness $T_j = \max\{C_j - d_j, 0\}$ that is computed based on its due

*Corresponding author
Email addresses: tbulhoes@ci.ufpb.br (Teobaldo Bulhões), Ruslan.Sadykov@inria.fr (Ruslan Sadykov), tbulhoes@ci.ufpb.br (Anand Subramanian), uchoa@producao.uff.br (Eduardo Uchoa)

date d_j. A classical objective is to minimize the total weighted earliness and tardiness given by $\sum_{j\in J}(w'_j E_j + w_j T_j)$, where w'_j and w_j are penalty coefficients associated with job j.

According to the notation introduced by Graham et al (1979), the general case of the class of problems described above can be denoted as $R|r_j, s^k_{ij}|\sum f_j(C_j)$. A considerable number of $\mathcal{NP}$-hard problems arise as special cases of problem $R|r_j, s^k_{ij}|\sum f_j(C_j)$, as they usually generalize standard problems such as $1||\sum w_j T_j$ and $P||\sum w_j C_j$, which are known to be $\mathcal{NP}$-hard (Lenstra et al, 1977). Dozens of variants can thus be derived by simply combining the existing attributes (i.e., release dates, setup times) with the machine environment (identical, uniform, unrelated) and one of the various objective functions (e.g., $\sum_{j\in J} w_j T_j$, $\sum_{j\in J}(w'_j E_j + w_j T_j)$, $\sum_{j\in J} w_j C_j$). Furthermore, remark that cost functions that include earliness penalties are not regular and may have optimal solutions that include idle times between jobs, which is a critical aspect to be considered when developing efficient algorithms for this type of problems. The problem with release dates may also have optimal solutions with idle times.

There is a vast literature associated to variants of the problem $R|r_j, s^k_{ij}|\sum f_j(C_j)$. Kramer and Subramanian (2017) recently presented an annotated bibliography of works related to this problem. They enumerated at least 130 works published in the last 25 years. Almost all of them proposed a solution approach for a particular variant or a very limited class of problems, such as the single machine exact algorithms by van den Akker et al (2000); Sourd and Kedad-Sidhoum (2003); Avella et al (2005); Sourd (2005); Sourd and Kedad-Sidhoum (2008); Tanaka and Fujikuma (2008); Bigras et al (2008); Pan and Shi (2008); Tanaka and Araki (2013) and the parallel machine exact algorithms by Chen and Powell (1999); Liaw et al (2003); Yalaoui and Chu (2006); Shim and Kim (2007a,b); Nessah et al (2008); Tanaka and Araki (2008); Jouglet and Savourey (2011); Schaller (2014); Bülbül and Şen (2017); Kowalczyk and Leus (2018).

Yet, there are highly successful general single machine exact approaches that were designed to cope with a gamut of variants such as the successive sublimation dynamic programming (SSDP) algorithm (Tanaka et al, 2009; Tanaka and Fujikuma, 2012). It was developed for problems without sequence-dependent setup times, where the authors could solve instances of different problems with up to 300 jobs. We could not identify, however, general purpose exact algorithms for problems involving setup times and a non-regular objective function. As a matter of fact, the performance of current exact algorithms tends to degrade when such attribute is incorporated. One possible explanation is that the one-machine subproblems that are solved in such algorithms become more challenging when setup times are considered. For example, the performance of the best known exact method for problem $1|s_{ij}|\sum w_j T_j$ (Tanaka and Araki, 2013) when solving 60-job instances to optimality varied quite a lot, ranging from few seconds to 30 days of CPU time.

We were also not able to identify many exact algorithms with the ability of efficiently solving a large number of parallel machine variants related to problem $R|r_j, s^k_{ij}|\sum f_j(C_j)$. The branch-cut-and-price algorithm by Pessoa et al (2010) could solve instances with up to 4 machines and 100 jobs for problems involving identical parallel machines and the weighted tardiness objective function.

This algorithm was later improved by Oliveira and Pessoa (2018), where the authors managed to solve more instances. The algorithm by Şen and Bülbül (2015) for problem $R||\sum w_jC_j$ combines a preemption-based relaxation with Benders' decomposition. Sequence-dependent setup times were not taken into account in both works. Pereira Lopes and Valério de Carvalho (2007) studied a variant similar to our general case, in particular, problem $R|a_k, r_j, s^k_{ij}|\sum w_jT_j$, where a_k is the availability date of machine k. They put forward a branch-and-price algorithm that could solve instances with up to 50 machines and 150 jobs. It is worth mentioning that the objective function here is regular. Moreover, the largest instances considered by the authors contain a very small number of jobs per machine what makes them much easier.

In this work we present a novel exact algorithm that is capable of solving problem $R|r_j, s^k_{ij}|\sum f_j(C_j)$ and the large class of problems that can be derived as particular cases from it. The proposed algorithm consists of a branch-cut-and-price approach that combines several features such as non-robust cuts, strong branching, reduced cost fixing and dual stabilization. We report improved bounds for benchmark instances of several problems. Moreover, we evaluate computationally the impact of non-robust cuts, which were employed for the first time for solving machine scheduling problems.

The remainder of the paper is organized as follows. Section 2 presents the extended mathematical formulation of the problem and the column generation approach to solve its linear relaxation. In Section 3 we present the cutting planes we then employ to strengthen the column generation dual bound. Section 4 describes the labeling algorithm for solving the pricing problem of the column generation approach. A procedure to eliminate arc variables by reduced cost arguments is presented in Section 5. Section 6 describes the proposed branch-cut-and-price approach. Section 7 contains the results of the computational experiments, whereas conclusions are drawn in Section 8.

2. Mathematical formulation

We start by defining some notation required to introduce the mathematical formulation. Let T be an upper bound on the maximum completion time of a job in some optimal solution. Let $G_k = (V_k = R_k \cup O_k, A_k = A^1_k \cup A^2_k \cup A^3_k \cup A^4_k)$ be the acyclic graph associated with each machine type $k \in K$, where set $R_k = \{(j,t,k) : j \in J, t = r_j + p_j, \ldots, T\}$ contains the vertices associated with the jobs. We adopt the convention that idle times only occur after the job has been processed. We assume that job j has been processed when arriving at vertex (j,t,k), but note that j did not necessarily finish at time t because of the existence of idle times. Set $O_k = \{(0,t,k) : t = 0, \ldots, T\}$ contains the vertices associated with dummy job 0. For brevity, we denote arc $((i,t,k),(j,t+s^k_{ij}+p^k_j,k))$ as (i,j,t,k).

Set $A^1_k = \{(i,j,t,k) = ((i,t,k),(j,t+s^k_{ij}+p^k_j,k)) : (i,t,k) \in R_k, (j,t+s^k_{ij}+p^k_j,k) \in R_k, j \in J \setminus \{i\}\}$ contains the arcs connecting the vertices of R_k. Set $A^2_k = \{(0,j,t,k) = ((0,t,k),(j,t+s^k_{0j}+p^k_j,k)) : (j,t+s^k_{0j}+p^k_j,k) \in R_k, j \in J\}$ contains all arcs connecting the vertices from O_k to R_k.

Set $A_k^3 = \{(j,0,t,k) = ((j,t,k),(0,T,k)) : (j,t,k) \in R_k\}$ contains all arcs connecting the vertices from R_k to O_k. Set $A_k^4 = \{(j,j,t,k) = ((j,t,k),(j,t+1,k)) : (j,t,k) \in R_k \cup O_k, (j,t+1,k) \in R_k \cup O_k\}$ contains the arcs associated with idle times. Let $c_a = c(i,j,t,k)$ be the cost of an arc $a = (i,j,t,k) \in A_k^1 \cup A_k^2$, which is the cost incurred if job j finishes to be processed at time $t + s_{ij}^k + p_j^k$: $c(i,j,t,k) = f_j(t + s_{ij}^k + p_j^k)$. Note that $c_a = 0$, $\forall a \in A_k^3 \cup A_k^4$.

Moreover, let set $R_k^j = \{(i,t,k) \in R_k : i = j\}$ denote the vertices associated with job j. Finally, for each subset $S \subseteq V_k$, let $\delta^-(S)$ and $\delta^+(S)$ be the sets representing the arcs entering and leaving S, respectively. The proposed arc-time-indexed formulation is as follows.

$$\text{(F1)} \quad \min \sum_{k \in K} \sum_{a \in A_k} c_a x_a \tag{1}$$

$$\text{s.t.} \sum_{k \in K} \sum_{a \in \delta^-(R_k^j) \setminus A_k^4} x_a = 1, \qquad \forall j \in J \tag{2}$$

$$\sum_{a \in A_k^2} x_a \leq m_k, \qquad \forall k \in K \tag{3}$$

$$\sum_{a \in \delta^-(\{v\})} x_a - \sum_{a \in \delta^+(\{v\})} x_a = 0,$$

$$\forall k \in K, \forall v \in V_k \setminus \{(0,0,k),(0,T,k)\} \tag{4}$$

$$x \geq 0, \tag{5}$$

$$x \text{ integer} \tag{6}$$

Objective function (1) minimizes the completion time dependent costs. Constraints (2) state that each job $j \in J$ must be processed exactly once. Constraints (3) impose that at most m_k machines of type $k \in K$ can be used. Constraints (4) are related to the flow conservation. It is worth mentioning that F1 can be easily adapted to cope with non-anticipatory setups, it is just a matter of removing some arcs from the network.

We now present an example for a small $R|r_j, s_{ij}^k|\sum w'_j E_j + w_j T_j$ instance with 8 jobs and 2 types of machines ($K = \{1,2\}$), where $m_1 = 2$ and $m_2 = 1$. All setup times are equal to 1, regardless of the machine type. The values of the attributes associated with each of the 8 jobs are described in Table 1.

Figure 1 illustrates the network representation and its associated Gantt chart of an optimal solution with cost 89 for the instance above. Variables x_a associated with arcs $(0,0,0,1)$, $(0,0,1,1)$ are set to two in this solution, while variables x_a associated with the following arcs are set to one: $(0,7,2,1)$, $(7,2,6,1)$, $(2,8,9,1)$, $(8,0,12,1)$, $(0,0,2,1)$, $(0,4,3,1)$, $(4,1,7,1)$, $(1,0,12,1)$, $(0,0,0,2)$, $(0,0,1,2)$, $(0,5,2,2)$, $(5,6,5,2)$, $(6,6,8,2)$, $(6,3,9,2)$, $(3,0,12,2)$.

Define $\mathcal{P}_k$ as the set of paths, in graph G_k, starting at vertex $(0,0,k)$ and ending at vertex $(0,T,k)$. Let b_P^a be the number of times path P traverses $a \in A_k$ and let λ_P be a binary variable

Table 1: Values of the attributes associated with each job

j	p_j^1	p_j^2	r_j	w'_j	w_j	d_j
1	4	4	4	7	16	11
2	2	3	5	6	13	7
3	3	2	9	4	11	12
4	3	4	4	10	21	8
5	3	2	3	7	20	5
6	3	2	4	8	21	8
7	3	2	3	4	10	7
8	2	3	9	5	11	9

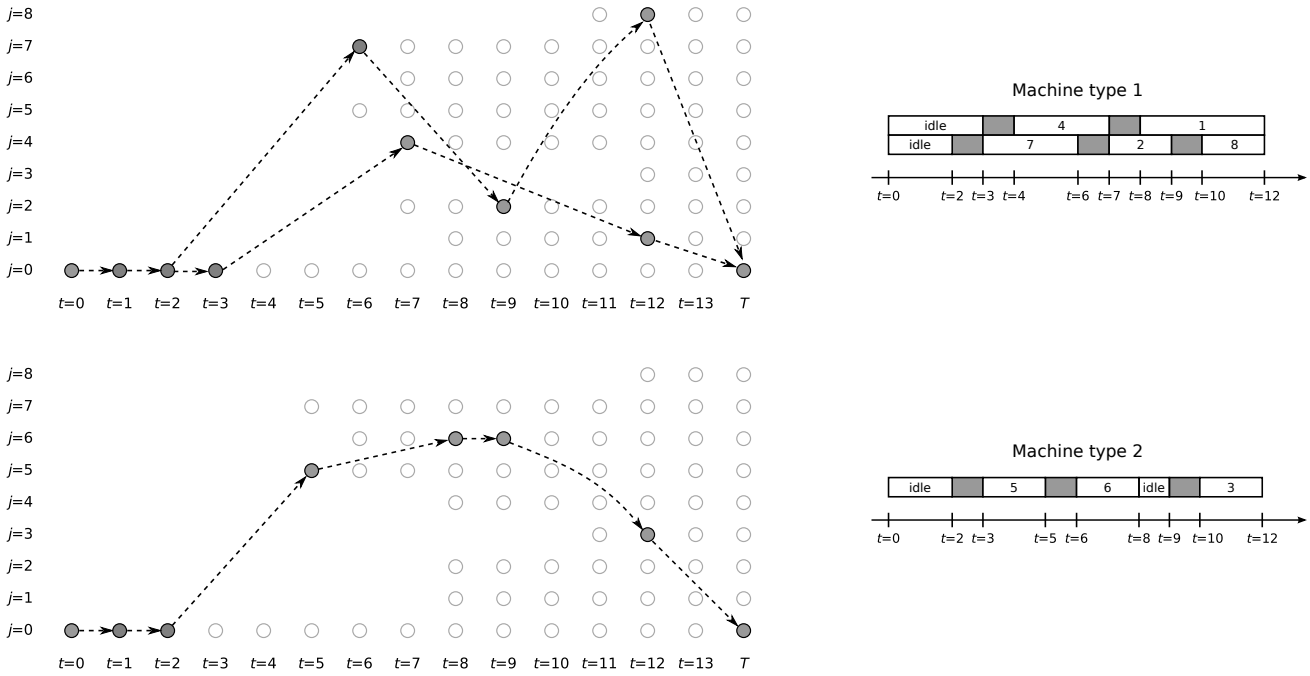

Figure 1: Network representation and its associated Gantt charts of an optimal solution for a small instance with cost 89.

that assumes value 1 if, and only if, P is in the solution. Formulation F1 can be rewritten in terms of variables x and λ as follows:

$$\min \sum_{k \in K} \sum_{a \in A_k} c_a x_a \tag{7}$$

$$\text{s.t.} \quad \sum_{P \in \mathcal{P}_k} b_a^P \lambda_P = x_a, \qquad \forall k \in K, \forall a \in A_k \tag{8}$$

$$\sum_{k \in K} \sum_{a \in \delta^-(R_k^j) \setminus A_k^4} x_a = 1, \qquad \forall j \in J \tag{9}$$

$$\sum_{a \in A_k^2} x_a \leq m_k, \qquad \forall k \in K \tag{10}$$

$$x, \lambda \geq 0, \tag{11}$$

$$x, \lambda \text{ integer} \tag{12}$$

By writing the formulation above only in terms of λ and relaxing the integrality constraints, we obtain the following Dantzig-Wolfe Master linear program:

$$\text{(DWM)} \quad \min \sum_{k \in K} \sum_{P \in \mathcal{P}_k} \Big(\sum_{a \in A_k} b_a^P c_a \Big) \lambda_P \tag{13}$$

$$\text{s.t.} \quad \sum_{k \in K} \sum_{P \in \mathcal{P}_k} \Big(\sum_{a \in \delta^-(R_k^j) \setminus A_k^4} b_a^P \Big) \lambda_P = 1, \qquad \forall j \in J \tag{14}$$

$$\sum_{P \in \mathcal{P}_k} \Big(\sum_{a \in A_k^2} b_a^P \Big) \lambda_P \leq m_k, \qquad \forall k \in K \tag{15}$$

$$\lambda \geq 0 \tag{16}$$

DWM is solved by dynamically generating variables λ through column generation. The pricing subproblem for a machine type $k \in K$ corresponds to finding a path $P \in \mathcal{P}_k$ with the smallest reduced cost. The reduced cost of a path can be decomposed into arc reduced costs, being the reduced cost of an arc (i, j, t, k) defined as:

$$\bar{c}(i,j,t,k) = \begin{cases} c(i,j,t,k) - \pi_j, \text{ if } i \neq j, i \neq 0, j \neq 0 \\ c(i,j,t,k) - \pi_j - \nu_k, \text{ if } i = 0, j \neq 0, \\ 0, \text{ otherwise,} \end{cases} \tag{17}$$

where π_j is the value of the dual variable of the constraint in (14) associated with job j and ν_k is the value of the dual variable of the constraint in (15) associated with machine type k. Note

that a path $P \in \mathcal{P}_k$ corresponds to a so-called pseudo-schedule, i.e. a schedule on machine type k in which some jobs may be processed more than once.

3. Rank-1 Chvátal-Gomory cuts

Valid inequalities can be obtained by a Chvátal-Gomory rounding of constraints in (14). Let $u_j \in [0, 1)$ be a rational multiplier associated with job $j \in J$. The Chvátal-Gomory rank of the resulting inequality

$$\sum_{k \in K} \sum_{P \in \mathcal{P}_k} \left(\left\lfloor \sum_{j \in J} u_j \sum_{a \in \delta^-(R_k^j) \setminus A_k^4} b_a^P \right\rfloor \right) \lambda_P \leq \left\lfloor \sum_{j \in J} u_j \right\rfloor \tag{18}$$

is 1. It will be referred as a rank-1 cut. Note that arcs associated with idle times are disregarded in (18).

Starting with Jepsen et al (2008), that proposed a subfamily of inequalities known as subset row cuts, rank-1 cuts have been extensively used in exact algorithms for VRPs over a set partitioning formulation. For examples, we may mention (Baldacci et al, 2011; Jepsen et al, 2013; Contardo and Martinelli, 2014; Gauvin et al, 2014). According to the classification proposed in Poggi de Aragão and Uchoa (2003), cuts defined over the arc variables are *robust*, since the effect of their dual variables can be translated into arc reduced costs without changing the pricing structure. On the other hand, cuts directly defined over the path variables λ, such as rank-1 cuts, are classified as *non-robust* because they affect the pricing structure. Indeed, an additional resource should be added in the shortest path pricing problem for each active cut (Pecin et al, 2017b). Hence, one cannot add many rank-1 cuts to DWM because the pricing problem becomes time-consuming to solve.

The limited-memory technique proposed by Pecin et al (2017b) serves to mitigate the negative impact of rank-1 cuts on the difficulty of the pricing problem. Each rank-1 cut has an associated memory which can be a set of vertices (as in Pecin et al (2017b)) or a set of arcs (as in Pecin et al (2017a)). In this work we define the memory as a set of pairs of jobs. Thus every cut is characterized by multipliers $u_j = \alpha_j/\beta$, $\alpha_j < \beta$, $j \in J$, and a memory $M \subseteq J \times J$. The coefficient of variable λ_P in a given limited memory rank-1 cut

$$\sum_{k \in K} \sum_{P \in \mathcal{P}_k} \texttt{coeff}(ps(P), \alpha, \beta, M) \lambda_P \leq \left\lfloor \sum_{j \in J} u_j \right\rfloor \tag{19}$$

depends only on the sequence of the pseudo-schedule $ps(P) = (j_1, \ldots, j_{|ps(P)|})$ associated with the path $P \in \mathcal{P} = \cup_{k \in K} \mathcal{P}_k$ and does not depend on the timing of the pseudo-schedule and thus idle times can be disregarded. Function `coeff` which is used to compute coefficients of variables λ in (19) is defined in Algorithm 1.

Algorithm 1 Computing the coefficient of variable λ_P in a limited memory rank-1 cut, characterized by α, β, and M

1: **function** `coeff`$(ps(P) = (j_1, \ldots, j_{|ps(P)|}),\ \alpha,\ \beta,\ M)$
2: $\quad state \leftarrow 0,\ coeff \leftarrow 0$
3: $\quad$ **for** $i \leftarrow 1$ **to** $|ps(P)|$ **do**
4: $\quad\quad$ **if** $i > 1$ **and** $(j_{i-1}, j_i) \notin M$ **then**
5: $\quad\quad\quad state \leftarrow 0$
6: $\quad\quad state \leftarrow state + \alpha_{j_i}$
7: $\quad\quad$ **if** $state \geq \beta$ **then**
8: $\quad\quad\quad state \leftarrow state - \beta,\ coeff \leftarrow coeff + 1$
9: **return** $coeff$

The smaller is the cut's memory, the smaller are the coefficients of some variables λ in the cut. Hence, limited-memory cuts are possibly weaker than the original "full-memory" rank-1 cuts (18). However, when using limited-memory cuts, very substantial gain in the pricing problem solution time is usually achieved. Moreover, during cut separation, it is possible to adjust the memories to the current fractional solutions. In fact, when rank-1 cut separation converges, limited-memory cuts obtain exactly the same bounds that would be obtained with the original cuts (Pecin et al, 2017b).

Assume that a separation procedure finds a violated rank-1 cut (18), characterized by vector α and scalar β. The cut that will be actually added is the limited memory variant (19) of the cut, with a memory set obtained as described in Algorithm 2. Let λ^* be the fractional solution and let $\mathcal{P}(\lambda^*) = \{P \in \mathcal{P} : \lambda^*_P > 0\}$ be the set of paths corresponding to strictly positive variables in λ^*. The idea is to obtain a small memory set such that the coefficients of all variables λ_P, $P \in \mathcal{P}(\lambda^*)$, are the same in (18) and in (19). In this way, the cut violation will remain the same. Function `computeMemory` first finds a minimal memory $M^{\min}$ with that property. Then, some other pairs of jobs are added to the final memory (lines 13 and 14). Those pairs of jobs are likely to be useful for keeping the cut strong in future iterations. The overall goal is to speedup the convergence of the separation.

Consider now as an example a 7-job instance for problem $P||\sum w_j T_j$ with the following data: $K = \{1\}$, $m_1 = 2$, $(p^1_1, w_1, d_1) = (48, 8, 114)$, $(p^1_2, w_2, d_2) = (20, 8, 73)$, $(p^1_3, w_3, d_3) = (43, 13, 51)$, $(p^1_4, w_4, d_4) = (25, 15, 51)$, $(p^1_5, w_5, d_5) = (35, 16, 106)$, $(p^1_6, w_6, d_6) = (30, 15, 104)$, $(p^1_7, w_7, d_7) = (70, 9, 86)$. After column generation over DWM, the following LP can be obtained:

$$
\begin{aligned}
\min \quad & 64\lambda_1 + 256\lambda_2 + 693\lambda_3 + 313\lambda_4 + 32\lambda_5 \\
& 0\lambda_1 + 1\lambda_2 + 0\lambda_3 + 1\lambda_4 + 0\lambda_5 = 1 \\
& 1\lambda_1 + 1\lambda_2 + 1\lambda_3 + 0\lambda_4 + 0\lambda_5 = 1 \\
& 0\lambda_1 + 1\lambda_2 + 1\lambda_3 + 0\lambda_4 + 1\lambda_5 = 1 \\
& 1\lambda_1 + 0\lambda_2 + 0\lambda_3 + 1\lambda_4 + 0\lambda_5 = 1
\end{aligned}
$$

Algorithm 2 Computing the limited memory of the rank-1 cut, characterized by α, β, given λ^*

```
1: function computeMemory(α, β, λ*)
2:    M^min ← ∅
3:    for P ∈ P(λ*) do
4:       Let ps(P) = (j_1, ..., j_|ps(P)|)
5:       M^part ← ∅, state ← 0
6:       for i ← 1 to |ps(P)| do
7:          if i > 1 and state > 0 then
8:             M^part ← M^part ∪ {(j_{i-1}, j_i)}
9:          state ← state + α_{j_i}
10:         if state ≥ β then
11:            state ← state − β
12:            M^min ← M^min ∪ M^part, M^part ← ∅
13:   M ← M^min ∪ {(j', j'') ∈ J × J : (j'', j') ∈ M^min}
14:   M ← M ∪ {(j', j'') ∈ J × J : α_{j'} > 0, α_{j''} > 0}
15: return M
```

$$
\begin{aligned}
1\lambda_1 + 1\lambda_2 + 0\lambda_3 + 0\lambda_4 + 1\lambda_5 &= 1 \\
1\lambda_1 + 0\lambda_2 + 1\lambda_3 + 0\lambda_4 + 1\lambda_5 &= 1 \\
0\lambda_1 + 0\lambda_2 + 1\lambda_3 + 1\lambda_4 + 0\lambda_5 &= 1 \\
1\lambda_1 + 1\lambda_2 + 1\lambda_3 + 1\lambda_4 + 1\lambda_5 &\leq 2 \\
\lambda &\geq 0
\end{aligned}
$$

Variables λ_1, λ_2, λ_3, λ_4, and λ_5 are associated with pseudo-schedules $ps(1) = (4, 2, 6, 5)$, $ps(2) = (3, 2, 5, 1)$, $ps(3) = (3, 2, 6, 7)$, $ps(4) = (4, 7, 1)$ and $ps(5) = (3, 6, 5)$, respectively. An optimal solution for the problem above is $\lambda_4^* = \frac{2}{3}$ and $\lambda_1^* = \lambda_2^* = \lambda_3^* = \lambda_5^* = \frac{1}{3}$, with cost 557. Let us define the multipliers vector $u = (\frac{1}{2}, \frac{1}{2}, 0, \frac{1}{2}, 0, 0, 0)^\top$. Hence, we obtain:

$$
\begin{bmatrix} \frac{1}{2} & \frac{1}{2} & 0 & \frac{1}{2} & 0 & 0 & 0 \end{bmatrix}
\begin{bmatrix} 0 & 1 & 0 & 1 & 0 \\ 1 & 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 \\ 1 & 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 & 0 \end{bmatrix}
\begin{bmatrix} \lambda_1 \\ \lambda_2 \\ \lambda_3 \\ \lambda_4 \\ \lambda_5 \end{bmatrix}
= \begin{bmatrix} \frac{1}{2} & \frac{1}{2} & 0 & \frac{1}{2} & 0 & 0 & 0 \end{bmatrix}
\begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{bmatrix}
$$

By summing up the equalities above, we have the following inequality:

$$1\lambda_1 + 1\lambda_2 + 0.5\lambda_3 + 1\lambda_4 + 0\lambda_5 \leq 1.5$$

After rounding down both sides of the inequality above, we obtain the following rank-1 cut:

$$\lambda_1 + \lambda_2 + \lambda_4 \leq 1 \tag{20}$$

It can be observed that the inequality above is not satisfied by λ^*. Given λ^* and applying Algorithm 2, we can now compute the memory for the rank-1 cut (20) characterized by $\alpha = (1, 1, 0, 1, 0, 0, 0)$ and $\beta = 2$. We obtain $M^{\min} = \{(4, 2), (2, 5), (5, 1), (4, 7), (7, 1)\}$, and final memory $M = M^{\min} \cup \{(2, 4), (5, 2), (1, 5), (7, 4), (1, 7)\} \cup \{(1, 2), (2, 1), (1, 4), (4, 1)\}$. Note that the coefficients of variables λ_1, λ_2, λ_3, λ_4, and λ_5 are the same in this cut with limited memory M and with full memory.

This does not mean that the limited-memory cut will have the best possible coefficients on variables that may appear later, from the column generation. For example, suppose that a variable λ_P such that $ps(P) = (1, 5, 4, 3, 1, 5, 2, 4)$ is generated. In Table 2, we give the evolution of values *state* and $coeff$ values in Algorithm 1 when calculating the coefficient with full memory and with the above calculated limited memory M. It can be seen that the coefficient of λ_P in the limited memory cut is smaller than in the full memory one. However, if λ_P becomes positive in the next fractional solution, Algorithm 2 will consider that variable in the subsequent limited-memory rank-1 cut separation. In particular, it is possible that a limited-memory rank-1 cut characterized by the same $\alpha = (1, 1, 0, 1, 0, 0, 0)$ and $\beta = 2$, but with a new memory set that is large enough to make the coefficient of λ_P to be equal to 2, is separated.

Table 2: Evolution of *state* and $coeff$ values in Algorithm 1 for $ps(P) = (1, 5, 4, 3, 1, 5, 2, 4)$

		Full Memory		Lim. Memory M	
i	j_i	*state*	$coeff$	*state*	$coeff$
1	1	1	0	1	0
2	5	1	0	1	0
3	4	0	1	1	0
4	3	0	1	0	0
5	1	1	1	1	0
6	5	1	1	1	0
7	2	0	2	0	1
8	4	1	2	1	1

4. Pricing algorithm

In this section, we describe the labeling-based pricing algorithm which aims at finding, for each $k \in K$, a path in G_k between vertices $(0, 0, k)$ and $(0, T, k)$ with the smallest reduced cost. Let $\mathcal{L}$ be the set of active rank-1 cuts. We define $\mu_\ell < 0$ as the value of the dual variable associated with cut $\ell \in \mathcal{L}$. Moreover, the denominator of the multipliers of cut $\ell \in \mathcal{L}$ is denoted as β^ℓ, while the

vector of numerators of the multipliers associated with this cut is denoted as α^ℓ. The memory of cut $\ell \in \mathcal{L}$ is denoted as M^ℓ.

In the pricing, reduced cost $\bar{c}(P)$ of a path $P \in \mathcal{P}_k$ equals to the sum of reduced costs of its arcs together with scalar product of the vector of coefficients of λ_P in active limited memory rank-1 cuts and vector μ. Let $\texttt{coeff}(P, \ell)$ be the coefficient of variable λ_P in cut $\ell \in \mathcal{L}$. This coefficient is calculated as in Algorithm 1 for α^ℓ, β^ℓ, and M^ℓ. Then

$$\bar{c}(P) = \sum_{(i,j,t,k)\in P} \bar{c}(i,j,t,k) - \sum_{\ell\in\mathcal{L}} \mu_\ell \cdot \texttt{coeff}(P,\ell) \tag{21}$$

where $\bar{c}(i,j,t,k)$ is defined in (17).

In addition, let $t(i,j,k) = s^k_{ij} + p^k_j$ be the length of the arcs that directly connect jobs i and j processed on machine k, except when $i = j$ and $j = 0$, where $t(i,i,k) = 1$ and $t(i,0,k) = 0$, respectively. In the reminder of this section we will omit index k for more clarity.

The labeling algorithm consists in an enumeration of all possible paths, employing dominance rules to eliminate partial paths proved not to lead to an optimal path. Our algorithm applies a so-called bidirectional search, that is, a path between vertices $(0,0)$ and $(0,T)$ will be obtained from the concatenation of a partial forward path and a partial backward path. The first is a path that begins at $(0,0)$ and ends at an intermediate vertex, traversing arcs of graph G in its regular order. The latter is a path that begins at $(0,T)$ and ends at an intermediate vertex, traversing the arcs from G in a reverse order. The reader is referred to Righini and Salani (2006) for more details on bidirectional labeling.

A label L corresponds to a partial path P^L. Forward path starts from vertex $(0,0)$ and backward path starts from vertex $(0,T)$. Each label L is characterized by vector $(\bar{c}^L, t^L, j^L, S^L)$, where

- $\bar{c}^L$ is the reduced cost of path P^L.
- (j^L, t^L) is the last vertex in P^L.
- S^L is a vector of states associated to cuts in $\mathcal{L}$.

If P^L is a forward path, then it corresponds to a partial pseudo-schedule ending at time t^L where the last job is j^L (due to idle times, j^L may have completed before t^L). If P^L is a backward path, then it corresponds to a partial pseudo-schedule starting at time t^L, with the additional commitment that j^L should be the last job to be completed before or at time t^L. Value S^L_ℓ for a forward path P^L is calculated in the same way as value *state* in Algorithm 1 for λ_{P^L}, α^ℓ, β^ℓ, and M^ℓ. For a backward path its calculation is done in the reverse order as it will be shown below.

A forward label L' dominates forward label L'' if for any partial path P, such that $(P^{L''}, P)$ is a complete path, $(P^{L'}, P)$ is also a complete path, and $\bar{c}((P^{L'}, P)) < \bar{c}((P^{L''}, P))$. Therefore, one can disregard dominated label L'' as it will not lead to an optimum solution. A backward

label L' dominates backward label L'' if for any partial path P, such that $(P, P^{L''})$ is a complete path, $(P, P^{L'})$ is also a complete path, and $\bar{c}((P, P^{L'})) < \bar{c}((P, P^{L''}))$. It can be verified that the following conditions are sufficient, in both forward and backward cases, for the dominance of label L'' by label L':

- $j^{L'} = j^{L''}$,
- $t^{L'} = t^{L''}$,
- $\bar{c}^{L'} < \bar{c}^{L''} + \sum_{\ell \in \mathcal{L}:\, S_\ell^{L'} > S_\ell^{L''}} \mu_\ell$.

Before describing the labeling algorithm we need to present auxiliary procedures `ExtendForward` and `ExtendBackward` which extend a forward or backward label L to a vertex corresponding to job j. Note that this vertex can be uniquely defined given j^L, t^L, and j. The procedures are formally defined in Algorithm 3. Note that the vector of states S^L does not change when extending label L along an idle time arc.

Algorithm 3 Label Extension Algorithm

Procedure `ExtendForward`(L, j)
$j^{L'} \leftarrow j$
$\bar{c}^{L'} \leftarrow \bar{c}^L + \bar{c}(j^L, j, t^L)$
$t^{L'} \leftarrow t^L + t(j^L, j)$
$S^{L'} \leftarrow S^L$
for $\ell \in \mathcal{L}$ **do**
  if $j \neq j^L$ **and** $(j^L, j) \notin M_\ell$ **then**
    $S_\ell^{L'} \leftarrow 0$
  if $j \neq j^L$ **then**
    $S_\ell^{L'} \leftarrow S_\ell^{L'} + \alpha_j^\ell$
    if $S_\ell^{L'} \geq \beta_\ell$ **then**
      $S_\ell^{L'} \leftarrow S_\ell^{L'} - \beta_\ell$
      $\bar{c}^{L'} \leftarrow \bar{c}^{L'} - \mu_\ell$
return L'

Procedure `ExtendBackward`(L, j)
$j^{L'} \leftarrow j$
$\bar{c}^{L'} \leftarrow \bar{c}^L + \bar{c}(j, j^L, t^L - t(j, j^L))$
$t^{L'} \leftarrow t^L - t(j^L, j)$
$S^{L'} \leftarrow S^L$
for $\ell \in \mathcal{L}$ **do**
  if $j \neq j^L$ **then**
    $S_\ell^{L'} \leftarrow S_\ell^{L'} + \alpha_{j^L}^\ell$
    if $S_\ell^{L'} \geq \beta_\ell$ **then**
      $S_\ell^{L'} \leftarrow S_\ell^{L'} - \beta_\ell$
      $\bar{c}^{L'} \leftarrow \bar{c}^{L'} - \mu_\ell$
  if $j \neq j^L$ **and** $(j, j^L) \notin M_\ell$ **then**
    $S_\ell^{L'} \leftarrow 0$
return L'

In the forward labeling algorithm, we store in forward bucket $F(j,t)$ all labels L such that $j^L = j$ and $t^L = t$. First, initial label $L = (0, 0, 0, \mathbf{0})$ representing empty path $P^L = \emptyset$ is inserted in bucket $F(0,0)$. Next, forward buckets $F(j,t)$ are visited in a non-decreasing order of time t, from 0 to $t^* - 1$, where t^* is a threshold value that determines up to which time the forward label extensions are performed. Every time a bucket $F(j,t)$ is considered, the procedure extends each label in it along all possible arcs $(j, j', t) \in A$. The new label originated from such extension is inserted to bucket $F(j, t + t(j,j'))$ only if it is not dominated by any of the labels in the latter bucket. Existing labels in $F(j, t + t(j,j'))$ that are dominated by the new label are removed. The backward labeling algorithm is similar to its forward counterpart and works with backward buckets $B(j,t)$, the extension of labels is performed in a backward direction. However, there

is an asymmetry between those complementary algorithms: there are no backward buckets for times smaller than t^*. Those buckets will not be needed in the concatenation procedure that will be later described. In both forward and backward labeling, in order to speedup the dominance checks, the labels in a given bucket are stored in a non-decreasing order of reduced cost. The forward and backward labeling algorithms are formally described in Algorithm 4.

Algorithm 4 Mono-directional labeling algorithm

Procedure `ForwardLabeling`(t^*, A, P^{best})
$F(j,t) \leftarrow \emptyset$; $j \in J \cup \{0\}$, $t \in \{0,1,\ldots,T\}$
$F(0,0) \leftarrow \{L = (0,0,0,\mathbf{0})\}$
for $t = 0$ **to** $t^* - 1$ **do**
  for each $j \in J \cup \{0\}$ **do**
    for each $(j, j', t) \in A$ **do**
      for each $L \in F(i,t)$ **do**
        $L' \leftarrow$ `ExtendForward`(L, j')
        if $t^{L'} > T$ **then**
          continue
        if L' is not dominated by $L'' \in F(j^{L'}, t^{L'})$ **then**
          insert L' to bucket $F(j^{L'}, t^{L'})$ and remove from it all labels L'' dominated by L'
        if $j^{L'} = 0$ **and** $\bar{c}^{L'} < \bar{c}(P^{best})$ **then**
          $P^{best} \leftarrow P^{L'}$
return (F, P^{best})

Procedure `BackwardLabeling`(t^*, A, P^{best})
$B(j,t) \leftarrow \emptyset$; $j \in J \cup \{0\}$, $t \in \{t^*,\ldots,T\}$
$B(0,T) \leftarrow \{L = (0,0,T,\mathbf{0})\}$
for $t = T$ **downto** $t^* + 1$ **do**
  for each $j \in J \cup \{0\}$ **do**
    for each $(j', j, t - t(j', j)) \in A$ **do**
      for each $L \in B(i,t)$ **do**
        $L' \leftarrow$ `ExtendBackward`(L, j')
        if $t^{L'} < t^*$ **then**
          continue
        if L' is not dominated by $L'' \in B(j^{L'}, t^{L'})$ **then**
          insert L' to bucket $B(j^{L'}, t^{L'})$ and remove from it all labels L'' dominated by L'
        if $j^{L'} = 0$ **and** $\bar{c}^{L'} < \bar{c}(P^{best})$ **then**
          $P^{best} \leftarrow P^{L'}$
return (B, P^{best})

The principle of a bidirectional labeling algorithm is to obtain complete paths by concatenating forward and backward labels. Concatenation is performed between labels in buckets $F(j,t)$ and $B(j,t)$, for each t, $t^* \leq t \leq T$, and each $j \in J$. When concatenating labels L and L' we need to take into account states S^L and $S^{L'}$ and adjust the reduced cost of the obtained complete path if necessary. In fact the reduced cost of the concatenated path is obtained as the sum of the reduced costs of forward and backward partial paths P^L and $P^{L'}$ and the concatenation cost. The latter is non-negative and can be strictly positive because the coefficient of the concatenated path in a rank-1 cut $\ell \in \mathcal{L}$ can be one unit larger than the sum of the coefficients of forward and backward paths in ℓ. This happens when the sum of states S^L_ℓ and $S^{L'}_\ell$ is greater than or equal to β_ℓ. The concatenation procedure is formally given in Algorithm 5.

The full bi-directional labeling procedure is described in Algorithm 6. In it we first run the forward and backward mono-directional labeling up to time threshold t^*. Then the concatenation is performed.

Algorithm 5 Concatenation Algorithm

Procedure `Concatenation`(t^*, F, B, P^{best})
for $t = t^*$ **to** T **do**
  for each $j \in J$ **do**
    for each $L \in F(j,t)$ **do**
      for each $L' \in B(j,t)$ **do**
        $P \leftarrow (P^L, \texttt{revert}(P^{L'}))$
        $\bar{c}(P) \leftarrow \bar{c}^L + \bar{c}^{L'}$
        for each $\ell \in \mathcal{L}$ **do**
          if $S^L_\ell + S^{L'}_\ell \geq \beta_\ell$ **then**
            $\bar{c}(P) \leftarrow \bar{c}(P) - \mu_\ell$
        if $\bar{c}(P) < \bar{c}(P^{best})$ **then**
          $P^{best} \leftarrow P$
return P^{best}

Algorithm 6 Bi-directional Labeling Algorithm

Procedure `Labeling`(t^*)
$P^{best} \leftarrow \emptyset$
$(F, P^{best}) \leftarrow$ `ForwardLabeling`(t^*, P^{best})
$(B, P^{best}) \leftarrow$ `BackwardLabeling`(t^*, P^{best})
$P^{best} \leftarrow$ `Concatenation`(t^*, F, B, P^{best})
return P^{best}

5. Arc fixing by reduced costs

Once DWM, possibly with additional limited memory rank-1 cuts (19), is solved by column generation, we apply an arc elimination scheme by reduced cost similarly to what was implemented in Pessoa et al (2010). Here we generalize it in order to take into account active rank-1 cuts. For each machine type $k \in K$, one seeks to eliminate (or fix) arcs from G_k that can not be part of a path $P \in \mathcal{P}_k$ which belongs to a solution that improves the incumbent solution. In order to fix an arc $(i,j,t,k) \in A_k$, one should show that, given the optimum dual solution (π, ν, μ) of the formulation, the smallest reduced cost $\bar{c}_{\min}(i,j,t,k)$ of a path that traverses arc (i,j,t,k) is smaller than the primal-dual gap $UB - LB$. Here LB is the value of the dual bound given by the optimum solution value of the formulation, and UB is the value of the incumbent solution. Note that once both forward and backward mono-directional labeling algorithms have been executed for the full time horizon, value $\bar{c}_{\min}(i,j,t,k)$ for any arc $(i,j,t,k) \in A_k$ can be obtained using the pairwise comparison of labels in buckets $F_k(i,t)$ and $B_k(j, t + t(i,j))$. The formal procedure for arc fixing by reduced costs is given in Algorithm 7.

After applying the arc fixing procedure, usually the size of graph G_k may be further reduced. Consider an arc $a = (j,j,t,k) \in A_k$ and suppose that there exists no arc $(j,j',t',k) \in A_k$ such that $j' \in J \setminus \{j\}$, $t' > t$. Arc a can appear in a path only after the last processed job which is j. This arc can be safely removed from the graph as for a path passing through it there should exist

Algorithm 7 Algorithm for arc fixing by reduced costs for graph G_k

Procedure `RedCostFixing`(LB, UB, A_k)
$(F, \cdot) \leftarrow$ `ForwardLabeling`$(T, A_k, \emptyset)$
$(B, \cdot) \leftarrow$ `BackwardLabeling`$(0, A_k, \emptyset)$
for $(i, j, t, k) \in A_k$ **do**
  $\bar{c}_{\min} \leftarrow \infty$
  for $L \in F(i, t)$ **do**
    for $L' \in B(j, t + t(i, j, k))$ **do**
      $\bar{c} \leftarrow \bar{c}^L + \bar{c}^{L'}$
      for $\ell \in \mathcal{L}$ **do**
        if $S_\ell^L + S_\ell^{L'} \geq \beta_\ell$ **then**
          $\bar{c} \leftarrow \bar{c} - \mu_\ell$
      $\bar{c}_{\min} \leftarrow \min\{\bar{c}_{\min}, \bar{c}\}$
  if $\bar{c}_{\min} \geq UB - LB$ **then**
    $A_k \leftarrow A_k \setminus \{(i, j, t, k)\}$
return A_k

a path with the same reduced cost passing through arc $(j, 0, t, k)$.

After removing some arcs from graph G_k we can also remove vertices which are not anymore incident to any arc in A_k. Thus, the vertex-sets R_k^j corresponding to the times where job $j \in J$ can complete in machine type k can be reduced. Threshold value t_k^* used in the bi-directional labeling algorithm depends on the current set R_k. Let $t_j^{k,\min}$ and $t_j^{k,\max}$ be the minimum and maximum value for t such that vertex (j, t, k) exists in R_k. Then value t_k^* is defined as:

$$t_k^* = \frac{\sum_{j \in J} t_j^{k,\min} + \sum_{j \in J} t_j^{k,\max}}{2 \cdot |J|}.$$

6. Branch-cut-and-price algorithm

This section describes the proposed branch-cut-and-price (BCP) algorithm. At every node, linear programs DWM (possibly with additional rank-1 cuts and branching constraints) are solved by column generation. The automatic dual price smoothing stabilization technique Pessoa et al (2018) is used to speed up its convergence. The column generation is performed in two stages. In the first stage the pricing problem is solved heuristically: in each bucket in the labeling algorithm, at most one label, the one with the smallest reduced cost is extended. When the heuristic can not find a column with negative reduced cost, the second stage starts. In that stage the pricing problem is always solved exactly, until convergence. At every iteration in the first/second stage of column generation we generate up to 50/300 columns with negative reduced cost per pricing subproblem.

When column generation converges, the arc fixing by reduced costs described in Section 5 is applied to reduce the size of graphs G_k, $k \in K$, and to adjust threshold values t_k^*, $k \in K$. After

that, separation of limited memory rank-1 cuts is performed. In our tests we used only cuts which are characterized by a vector u of multipliers in which exactly one or three components are equal to $\frac{1}{2}$ and other components are zero. These types of cuts correspond to the subset-row cuts (Jepsen et al, 2008) with one or three rows. These families of rank-1 cuts can be easily separated by inspection, testing all jobs or triples of jobs. For every added cut, we compute the memory as described in Algorithm 2. Note that if there already exists an active cut $\ell \in \mathcal{L}$ characterized by the same vector of multipliers, the memory of the new cut is only added to the memory of cut ℓ. In any separation round, we add at most 50 one-row cuts and at most 75 three-row cuts. Column generation is then executed again. When it converges, any rank-1 cut that became inactive (zero dual variable) is immediately removed from the master problem. Separation rounds are performed while the added cuts have significant impact on reducing the primal-dual gap. For this we define a tailing-off stopping condition: if the primal-dual gap decreased by less than 2% in the last two separation rounds, separation stops and branching is performed.

We use two branching strategies in our algorithm. In the first one, we branch on aggregated variables $z_j^k = \sum_{a \in \delta^-(R_k^j)} x_a$, which correspond to assignment of a job $j \in J$ to be processed on a machine of type $k \in M$. In the second one, we branch on aggregated variables $z_{ij} = \sum_{k \in K} \sum_{t:a=(i,j,t,k) \in A_k} x_a$, which correspond to immediate precedence between jobs $i \in J$ and $j \in J \setminus \{i\}$ on some machine. Note that adding branching constraints on variables z_j^k and z_{ij} does not change the structure of the pricing problem, as the branching constraints just modify the reduced costs of arcs.

We implemented a 2-phase strong branching scheme, similar to the ones proposed in Røpke (2012) and Pecin et al (2017b). The branching candidates correspond to pairs "job-machine" for the first strategy and pairs "job-job" for the second strategy. In the first phase, up to 50 candidates are evaluated. This number can be smaller if the estimated size of the branching subtree rooted at the current node is small. Half of the candidates come from the branching history (except on the root node, when no branching history exists), selected by their pseudo-costs (Achterberg, 2007). Among the remaining candidates, half corresponds to the assignment variables z_j^k and half to precedence variables z_{ij}. These candidates are chosen according to the fractional value of the variable (closest to 0.5 first). In the first phase, for each branch of each candidate (variable fixed to 0 and variable fixed to 1), we add the corresponding branching constraint and resolve the restricted master problem without generating additional columns. Three candidates are selected for the second phase. They are chosen according to the product of the lower bound increase in the both branches, the maximum value for this product first (product rule (Achterberg, 2007)). In the second phase, the candidates are evaluated more precisely, because heuristic column generation is performed. The best candidate from the second phase, again according to the product rule is chosen for branching.

7. Computational experiments

All experiments were conducted on an Intel Xeon E5-2680 v3 2.5 GHz and 128 GB of RAM with a single thread and a time limit of 12 hours per instance.

We used the $R||\sum w'_jE_j + w_jT_j$ instances of Şen and Bülbül (2015) (up to 120 jobs), the $R|r_j, s^k_{ij}|\sum w'_jE_j + w_jT_j$ instances of Kramer (2015) (up to 80 jobs), the $R|a_k, r_j, s^k_{ij}|\sum w_jT_j$ instances of Pereira Lopes and Valério de Carvalho (2007) (up to 100 jobs), and the $P||\sum w_jC_j$ instances of Kowalczyk and Leus (2018) (up to 100 jobs). It should be noted that the instances of Kramer (2015) were derived from the instances of Şen and Bülbül (2015) by adding two types of sequence-dependent setup times: small (S) and large (L). Moreover, all instances with unrelated machines satisfy $m_k = 1$ for all $k \in K$. The value of T for the $R||\sum w'_jE_j + w_jT_j$ instances is computed as in (Şen and Bülbül, 2015) and as in (Kowalczyk and Leus, 2018) for the $P||\sum w_jC_j$ instances. For the remaining instances, we use the following equations:.

$$T = \begin{cases} \left\lceil \sum_{j\in J} \left(\max\limits_{k\in K} \left(p^k_j\right) + \max\limits_{\substack{i\in J\cup\{0\}\\ i\neq j\\ k\in K}} \left(s^k_{ij}\right)\right)/m \right\rceil + \max\limits_{j\in J}\left(d_j\right) \\ + \max\limits_{\substack{j\in J\\ k\in K}} \left(p^k_j\right) + \max\limits_{\substack{i,j\in J\cup\{0\}\\ i\neq j, j\neq 0\\ k\in K}} \left(s^k_{ij}\right), \\ \text{for problem } R|r_j, s^k_{ij}|\sum w'_jE_j + w_jT_j \\ \\ \left\lceil \sum_{j\in J} \left(\max\limits_{k\in K} \left(p^k_j\right) + \max\left(s^k_{0j} + a_k, \max\limits_{\substack{i\in J\\ i\neq j}} \left(s^k_{ij}\right)\right)\right)/m \right\rceil \\ + \max\limits_{\substack{j\in J\\ k\in K}} \left(p^k_j\right) + \max\limits_{j\in J}\left(\max\left(s^k_{0j} + a_k, \max\limits_{\substack{i\in J\\ i\neq j}} \left(s^k_{ij}\right)\right)\right) \\ + \max\limits_{j\in J}\left(r_j\right), \\ \text{for problem } R|a_k, r_j, s^k_{ij}|\sum w_jT_j \end{cases} \tag{22}$$

Tables 3–8 report the results obtained by the proposed BCP, with and without Rank-1 cuts. In these tables, **Root Gap (%)** indicates the average percentage gap between the root relaxations and the best known upper bounds, **Root Cuts** corresponds to the average number of cuts added at the root node, **Root Time** denotes the average CPU time in seconds required to solve the root node, **Gap** is the average gap in per cent between the lower bounds and the best known upper bounds, **Sol.** indicates the number of instances that were solved to optimality, **Nd.** represents the average number of nodes of the BCP tree, and **Time** is the average CPU time in seconds required by the BCP algorithm.

Regarding the initial primal bound adopted by our BCP algorithm, we made use of the best upper bounds (UBs) reported in the literature for the instances of Şen and Bülbül (2015) and Pessoa et al (2010) and we executed the algorithm by Kramer and Subramanian (2017) to compute

UBs for the instances of Pereira Lopes and Valério de Carvalho (2007), Kowalczyk and Leus (2018) and Kramer (2015).

Table 3: Aggregate results obtained for the $R||\sum w'_jE_j + w_jT_j$ instances by Şen and Bülbül (2015)

$(n, m, \#\text{Ins})$	Root Gap	Root Cuts	Root Time	Gap	Sol.	Nd.	Time
$(40, 2, 60)$	0.04	330.5	22.4	0.00	60	3.2	208.1
$(60, 2, 60)$	0.04	532.9	84.3	0.00	60	3.1	239.2
$(60, 3, 60)$	0.04	564.1	36.5	0.00	60	3.0	94.8
$(80, 2, 60)$	0.02	827.5	275.2	< 0.01	59	4.6	1022.4
$(80, 4, 60)$	0.12	511.4	64.9	0.00	60	3.1	163.0
$(90, 3, 60)$	0.05	882.8	192.0	0.00	60	3.8	493.1
$(100, 5, 60)$	0.20	654.0	134.6	0.00	60	8.9	816.4
$(120, 3, 60)$	0.16	877.8	619.3	< 0.01	59	11.2	3411.6
$(120, 4, 60)$	0.26	692.8	371.8	0.01	59	9.6	2392.1
Avg.	0.10	652.6	200.1	< 0.01	59.7	5.6	982.3

(a) With Rank-1 Cuts

$(n, m, \#\text{Ins})$	Root Gap	Root Time	Gap	Sol.	Nd.	Time
$(40, 2, 60)$	0.29	13.0	< 0.01	59	52.8	319.1
$(60, 2, 60)$	0.24	30.6	0.02	55	175.8	3278.0
$(60, 3, 60)$	0.27	10.4	0.00	60	56.7	298.0
$(80, 2, 60)$	0.22	72.3	0.01	56	189.6	5809.5
$(80, 4, 60)$	0.32	23.9	< 0.01	58	111.3	1488.6
$(90, 3, 60)$	0.27	52.2	0.01	55	176.1	4962.2
$(100, 5, 60)$	0.40	49.9	0.01	56	177.0	4514.2
$(120, 3, 60)$	0.33	165.8	0.12	41	156.5	15409.1
$(120, 4, 60)$	0.43	137.7	0.10	46	137.3	10471.9
Avg.	0.31	61.7	0.03	54.0	136.2	5098.1

(b) Without Rank-1 Cuts

Table 4: Aggregate results obtained for the $R|r_j, s^k_{ij}|\sum w'_jE_j + w_jT_j$ instances of Kramer (2015)

$(n, m, \text{set.}, \#\text{ Ins})$	Root Gap	Root Cuts	Root Time	Gap	Sol.	Nd.	Time
$(40, 2, S, 60)$	0.00	845.8	193.4	0.00	60	1.0	193.5
$(40, 2, L, 60)$	0.19	1734.8	580.8	0.00	60	1.7	834.0
$(60, 2, S, 60)$	0.07	1991.7	1102.5	0.00	60	1.4	1403.6
$(60, 2, L, 60)$	0.80	1715.7	1847.1	0.01	59	9.8	5966.4
$(60, 3, S, 60)$	0.41	1898.7	688.1	0.00	60	5.2	1338.3
$(60, 3, L, 60)$	2.12	1051.9	1066.6	0.13	53	34.0	11750.0
$(80, 2, S, 60)$	0.13	2446.5	3376.2	0.00	60	3.6	4806.3
$(80, 2, L, 60)$	1.69	1029.5	4922.3	0.55	33	32.3	25819.5
$(80, 4, S, 60)$	1.27	926.2	1576.3	0.15	49	40.7	12212.0
$(80, 4, L, 60)$	3.73	584.9	2972.2	1.75	14	59.1	30909.6
Avg.	1.04	1422.6	1832.6	0.26	50.8	18.9	9523.3

(a) With Rank-1 Cuts

$(n, m, \text{set.}, \#\text{ Ins})$	Root Gap	Root Time	Gap	Sol.	Nd.	Time
$(40, 2, S, 60)$	1.59	163.3	0.00	60	31.5	281.8
$(40, 2, L, 60)$	3.27	260.3	0.00	60	71.5	690.5
$(60, 2, S, 60)$	1.66	622.1	< 0.01	59	151.6	2913.7
$(60, 2, L, 60)$	3.31	1133.3	0.10	53	414.9	11064.3
$(60, 3, S, 60)$	1.73	473.8	0.00	60	131.7	1461.0
$(60, 3, L, 60)$	3.86	816.6	0.14	54	513.3	11902.5
$(80, 2, S, 60)$	1.52	2049.2	0.08	47	333.4	17000.5
$(80, 2, L, 60)$	3.55	3760.7	1.53	6	174.7	34073.9
$(80, 4, S, 60)$	2.27	1280.5	0.22	48	475.8	13826.2
$(80, 4, L, 60)$	5.05	2352.8	2.32	10	213.3	32588.2
Avg.	2.78	1291.3	0.44	45.7	251.2	12580.3

(b) Without Rank-1 Cuts

From the results obtained in Tables 3 and 4, it can be observed that adding Rank-1 cuts clearly improve the performance of the BCP algorithm for the instances with the earliness-tardiness objective on unrelated machines. In particular, 486 and 457 instances of the benchmarks of Şen and Bülbül (2015) and Kramer (2015), respectively, were solved by the version without rank-1 cuts, whereas 537 and 508 instances of such benchmarks were solved, respectively, when rank-1 cuts were considered. The gains in the root relaxation quality due to rank-1 cuts are clear: the root gaps are about 3 times smaller on average, and the number of nodes is smaller by an order of magnitude. From these tables one can see that the rank-1 cuts are more useful for instances with smaller number of machines. Although, the root time increases significantly for these instances, the root gap decrease is more substantial.

The BCP performance tends to decrease when setup times are considered as the value of T is likely to increase considerably. While only 3 instances (up to 120 jobs) of those proposed by Şen

and Bülbül (2015) remain open, 122 of the instances of Kramer (2015) (up to 80 jobs) were not solved to optimality. The difference in performance is also visible when larger setup values are considered. More specifically, while only 11 instances with small setup times were not solved by BCP, 101 of those instances with relatively large setup values could not be solved.

Note that the instances with the earliness-tardiness objective we consider have never been approached by an exact algorithm in the literature. The algorithm in Şen and Bülbül (2015) is a primal-dual approach which may prove optimality for some instances, but does not have a mechanism to close the primal-dual gap in general. Table 5 compares the results obtained by our BCP algorithm with the average gaps and number of optimal solutions found by the algorithm developed in Şen and Bülbül (2015). It is worth mentioning that such average gaps were computed in terms of the lower bounds achieved by each method and the best upper bounds, considering those obtained by our BCP. In other words, we are comparing the quality of the lower bounds determined by each method. It can be seen from the paper that our approach is much better in terms of number of optimal solutions obtained and the primal-dual gap. Many instances were solved to optimality for the first time. However the running time of our algorithm is larger, especially for the largest instances. Nevertheless, from Table 3 it can be seen that if we limit our algorithm to the root, much smaller gaps than by Şen and Bülbül (2015) can be obtained in a comparable time.

Table 5: Results for $R||\sum w'_jE_j + w_jT_j$ instances: comparison with the results obtained by Şen and Bülbül (2015)

	This paper				Literature		
$(n, m, \#\text{Ins})$	Gap	Sol.	Nd.	Time	Gap	Sol.	Time
$(40, 2, 60)$	0.00	60	3.2	208.1	0.13	22	52
$(60, 2, 60)$	0.00	60	3.1	239.2	0.43	5	109
$(60, 3, 60)$	0.00	60	3.0	94.8	0.38	4	120
$(80, 2, 60)$	< 0.01	59	4.6	1022.4	0.36	2	134
$(80, 4, 60)$	0.00	60	3.1	163.0	2.14	0	228
$(90, 3, 60)$	0.00	60	3.8	493.1	1.26	0	153
$(100, 5, 60)$	0.00	60	8.9	816.4	6.03	0	297
$(120, 3, 60)$	< 0.01	59	11.2	3411.6	3.21	0	165
$(120, 4, 60)$	0.01	59	9.6	2392.1	5.20	0	217
Avg.	< 0.01	59.7	5.6	984.2	2.13	3.7	163.9

In Table 6, we report the average improvement of the best known solutions (BKS) reported in the literature (Kramer, 2015; Kramer and Subramanian, 2017; Şen and Bülbül, 2015) by our algorithm. Improved solutions were found for numerous instances. Nevertheless, the relative improvement can be considered marginal. This suggest that the quality of the heuristics in the literature is rather high.

In Tables 7 and 8, we present results for instances by Pereira Lopes and Valério de Carvalho (2007) and Kowalczyk and Leus (2018). We would like to emphasis that the purpose of these results is to illustrate the impact of using non-robust cuts. The aim of the experiment was

Table 6: Comparison of upper bounds found with those known in the literature

$R\|\|\sum w'_j E_j + w_j T_j$			$R\|r_j, s^k_{ij}\|\sum w'_j E_j + w_j T_j$		
$(n, m, \#\text{Ins})$	Improv. (%)	New	$(n, m, \#\text{Insts})$	Improv. (%)	New
$(40, 2, 60)$	0.00	0	$(40, 2, S, 60)$	0.00	0
$(60, 2, 60)$	0.00	0	$(40, 2, L, 60)$	0.01	2
$(60, 3, 60)$	< 0.01	1	$(60, 2, S, 60)$	0.01	2
$(80, 2, 60)$	< 0.01	2	$(60, 2, L, 60)$	0.04	13
$(80, 4, 60)$	0.06	14	$(60, 3, S, 60)$	0.01	3
$(90, 3, 60)$	0.03	13	$(60, 3, L, 60)$	0.20	25
$(100, 5, 60)$	0.12	27	$(80, 2, S, 60)$	0.02	11
$(120, 3, 60)$	0.10	23	$(80, 2, L, 60)$	0.11	24
$(120, 4, 60)$	0.16	30	$(80, 4, S, 60)$	0.05	18
			$(80, 4, L, 60)$	0.07	11
Avg.	0.05	13.8	Avg.	0.06	12.1

Table 7: Aggregate results obtained for the $R|a_k, r_j, s^k_{ij}|\sum w_j T_j$ instances by Pereira Lopes and Valério de Carvalho (2007)

$(n, m, \#\text{Ins})$	Root Gap	Root Cuts	Root Time	Gap	Sol.	Nd.	Time
$(20, 2, 50)$	0.00	25.4	10.4	0.00	50	1.0	10.4
$(30, 2, 50)$	0.00	45.2	42.0	0.00	50	1.0	42.0
$(30, 4, 50)$	0.00	45.9	33.8	0.00	50	1.1	33.9
$(40, 2, 50)$	0.00	174.5	116.2	0.00	50	1.0	116.2
$(40, 4, 50)$	0.16	144.6	94.0	0.00	50	1.2	95.4
$(40, 6, 50)$	0.04	27.0	79.5	0.00	50	1.1	79.8
$(40, 8, 50)$	0.32	23.0	74.5	0.00	50	1.9	76.0
$(50, 2, 50)$	0.00	181.6	303.3	0.00	50	1.0	303.3
$(50, 4, 50)$	0.22	269.8	207.8	0.00	50	1.4	212.2
$(50, 6, 50)$	3.82	143.1	183.3	0.00	50	2.1	201.9
$(50, 8, 50)$	0.24	38.5	161.9	0.00	50	2.0	164.9
$(60, 4, 50)$	0.09	215.4	440.0	0.00	50	1.2	442.6
$(60, 6, 50)$	0.40	234.1	361.2	0.00	50	2.1	376.3
$(60, 8, 50)$	0.28	131.3	323.4	0.00	50	2.7	332.2
$(70, 6, 50)$	0.64	282.1	695.4	0.00	50	2.9	753.0
$(70, 8, 50)$	0.32	214.1	596.0	0.00	50	3.4	613.3
$(70, 10, 50)$	0.95	142.5	532.4	0.00	50	8.2	662.9
$(80, 8, 50)$	0.86	308.8	974.4	0.00	50	8.3	1454.0
$(90, 10, 50)$	1.43	282.6	1495.5	0.03	49	10.8	2727.4
$(100, 10, 50)$	1.25	313.4	2278.4	0.00	50	14.1	3816.2
$(100, 20, 50)$	2.28	42.8	1806.3	0.00	50	18.8	2574.2
Avg.	0.63	156.5	514.7	< 0.01	50.0	4.2	718.5

(a) With Rank-1 Cuts

$(n, m, \#\text{Ins})$	Root Gap	Root Time	Gap	Sol.	Nd.	Time
$(20, 2, 50)$	1.30	9.9	0.00	50	2.7	10.6
$(30, 2, 50)$	1.82	40.2	0.00	50	2.6	43.7
$(30, 4, 50)$	0.65	31.6	0.00	50	3.1	33.1
$(40, 2, 50)$	1.18	102.6	0.00	50	7.6	122.4
$(40, 4, 50)$	0.99	85.9	0.00	50	5.2	92.1
$(40, 6, 50)$	0.84	74.1	0.00	50	2.9	76.9
$(40, 8, 50)$	0.74	70.3	0.00	50	4.0	73.3
$(50, 2, 50)$	1.36	264.7	0.00	50	12.8	338.6
$(50, 4, 50)$	2.30	183.8	0.00	50	14.8	216.2
$(50, 6, 50)$	4.70	167.0	0.00	50	6.3	181.6
$(50, 8, 50)$	0.59	150.9	0.00	50	4.2	155.1
$(60, 4, 50)$	1.06	392.8	0.00	50	11.7	429.7
$(60, 6, 50)$	1.15	326.0	0.00	50	9.2	353.4
$(60, 8, 50)$	0.79	296.5	0.00	50	8.6	313.8
$(70, 6, 50)$	1.41	622.9	0.00	50	14.5	694.8
$(70, 8, 50)$	0.97	537.1	0.00	50	11.2	572.3
$(70, 10, 50)$	1.46	484.0	0.00	50	18.8	567.7
$(80, 8, 50)$	1.55	864.7	0.00	50	28.1	1185.0
$(90, 10, 50)$	2.05	1337.1	0.00	50	30.4	1989.7
$(100, 10, 50)$	1.87	1959.4	0.00	50	55.7	3584.7
$(100, 20, 50)$	2.49	1540.5	0.00	50	29.0	2261.9
Avg.	1.49	454.4	0.00	50.0	13.5	633.2

(b) Without Rank-1 Cuts

Table 8: Aggregate results obtained for the $P||\sum w_jC_j$ instances by Kowalczyk and Leus (2018)

$(n, m, \#\text{Ins})$	Root Gap	Root Cuts	Root Time	Gap	Sol.	Nd.	Time
$(20, 3, 120)$	0.00	0.2	0.4	0.00	120	1.0	0.4
$(20, 5, 120)$	0.00	0.2	0.2	0.00	120	1.0	0.3
$(20, 8, 120)$	1.18	0.1	0.1	0.00	120	1.0	0.1
$(20, 10, 120)$	0.00	0.0	0.1	0.00	120	1.0	0.1
$(20, 12, 120)$	0.00	0.0	0.1	0.00	120	1.0	0.2
$(50, 3, 120)$	0.03	7.0	13.6	0.00	120	1.1	13.7
$(50, 5, 120)$	0.35	19.1	6.1	0.00	120	1.3	6.6
$(50, 8, 120)$	0.98	16.0	3.0	0.00	120	1.4	3.3
$(50, 10, 120)$	1.13	7.8	2.3	0.00	120	1.4	2.5
$(50, 12, 120)$	1.48	3.0	1.8	0.00	120	1.4	1.9
$(100, 3, 120)$	0.18	37.0	274.0	0.04	113	7.2	4198.1
$(100, 5, 120)$	0.78	58.8	102.0	0.34	105	11.2	5744.4
$(100, 8, 120)$	1.54	65.6	45.3	0.20	114	13.3	4663.1
$(100, 10, 120)$	1.93	53.5	30.2	0.12	116	13.1	3282.8
$(100, 12, 120)$	2.55	55.5	20.7	0.00	120	11.0	1153.3
Avg.	0.81	21.6	33.3	0.05	117.9	4.5	1271.4

(a) With Rank-1 Cuts

$(n, m, \#\text{Ins})$	Root Gap	Root Time	Gap Gap	Sol.	Nd.	Time Time
$(20, 3, 120)$	0.54	0.4	0.00	120	1.0	0.4
$(20, 5, 120)$	0.22	0.2	0.00	120	1.1	0.2
$(20, 8, 120)$	1.65	0.1	0.00	120	1.1	0.1
$(20, 10, 120)$	0.00	0.1	0.00	120	1.0	0.1
$(20, 12, 120)$	0.51	0.1	0.00	120	1.0	0.1
$(50, 3, 120)$	0.07	13.6	0.00	120	1.2	13.7
$(50, 5, 120)$	0.45	10.0	0.00	120	1.6	10.8
$(50, 8, 120)$	1.67	3.3	0.00	120	1.9	3.7
$(50, 10, 120)$	1.38	2.3	0.00	120	1.6	2.5
$(50, 12, 120)$	1.83	1.8	0.00	120	1.6	1.9
$(100, 3, 120)$	0.21	277.3	< 0.01	119	10.8	2323.3
$(100, 5, 120)$	0.95	102.6	0.00	120	13.8	1402.5
$(100, 8, 120)$	1.78	44.1	0.00	120	16.7	705.9
$(100, 10, 120)$	2.07	31.1	0.00	120	14.2	353.6
$(100, 12, 120)$	2.67	20.6	0.00	120	14.0	137.5
Avg.	1.07	33.8	0.00	119.9	5.5	330.4

(b) Without Rank-1 Cuts

not to compare the efficiency of our algorithm with those from the literature. Note that our implementation is more generic and thus slower. Note that the gaps are given in $10^{-4}\%$ in Table 8. It can be seen from the results that impact of rank-1 cuts is lower on these instances. The first reason is that the number of machines here is significantly larger on average, and it was already noticed above that non-robust cuts are less useful this case. The second reason is that the objective function for these instances is regular. This property seems to make them easier and the basic branch-and-price algorithm is already good enough to solve all these instances (except one) to optimality. The computational results in original papers confirm this observation. Nevertheless, rank-1 cuts allows us to significantly reduce the root gap and the number of nodes for the instances by Pereira Lopes and Valério de Carvalho (2007). However, the impact for the instances by Kowalczyk and Leus (2018) is very limited. We conjecture that this is because these instances have the following symmetry. If several identical machines process partial schedules starting and finishing at the same time in a solution, any permutation of these partial schedules among these machines produce another feasible solution with the same cost. For example, let two identical machines process schedules $((1, 2, 3, 4, 5), (6, 7, 8, 9))$, such that jobs 3 and 7 start at the same time, whereas jobs 4 and 8 end at the same time. Then, schedules $((1, 2, 7, 8, 5), (6, 3, 4, 9))$ would have the same cost. This kind of symmetry is not eliminated by the formulations we use.

In Table 9 we present the results obtained by the branch-cut-and-price algorithm at the root node on instances by Pessoa et al (2010) for the problem $P||\sum w_jT_j$. Again, the aim here is to estimate the impact of rank-1 cuts, and not to compare our algorithm with the literature. In the table, we give results for three variants of our algorithm. The columns from 2 to 3 correspond to the pure column generation algorithm without cut generation. The columns from 4 to 6 correspond to variant in which we generate robust cuts: extended capacity cuts from Pessoa et al

(2010) and overload elimination cuts from Oliveira and Pessoa (2018). We used the cut generation code provided by the authors of these papers. The columns from 7 to 9 correspond to the variant where both robust and non-robust rank-1 cuts are separated.

Table 9: Aggregate root node results obtained for the $P||\sum w_jT_j$ instances by Pessoa et al (2010)

	No Cuts		Robust cuts only			Rank-1 + Robust cuts		
$(n, m, \#\text{Ins})$	Gap	Time	Gap	#Cuts	Time	Gap	#Cuts	Time
$(40, 2, 16)$	2.30	1.1	0.04	423.1	39.8	< 0.01	1017.5	349.0
$(40, 4, 16)$	0.64	0.5	0.30	113.8	12.7	0.28	422.2	17.3
$(50, 2, 18)$	0.79	2.6	0.11	481.3	104.8	0.08	895.6	211.8
$(50, 4, 17)$	0.75	1.2	0.43	105.6	25.7	0.41	470.9	38.5
$(100, 2, 20)$	0.95	123.7	0.07	558.2	745.6	0.05	1212.0	1784.2
$(100, 4, 19)$	0.68	17.9	0.40	150.5	292.8	0.39	621.1	501.4
Avg.	1.02	24.5	0.22	305.4	203.6	0.20	773.2	483.7

Notice again that the impact of rank-1 cuts is limited here. They remove on average only about 10% of the residual gap left after separating robust cuts. Again, we conjecture that this may be caused by the same symmetry present here as for the problem $P||\sum w_jC_j$. The robust cuts from Pessoa et al (2010) and Oliveira and Pessoa (2018) are not sensitive to this symmetry. Whereas rank-1 cuts often separate only one symmetric solution, and should be generated again for other symmetric solutions. Thus, the convergence of the rank-1 cuts is severely affected and their generation is stopped by the tailing-off condition.

Note however that average gap reduction attained by rank-1 cuts is affected by the fact that 88 from 150 instances have zero gap after separation of robust cuts. From the detailed results (not presented here), one can see that for 12 instances rank-1 cuts closed 100% of the gap remaining after generation of robust cuts. Also, for 13 more instances more that 30% of the remaining gap was closed.

8. Concluding remarks

This work introduced a unified exact algorithm capable of solving a broad class of parallel machine scheduling problems considering different attributes such as release dates and sequence-dependent setup times, and any objective function defined in terms of the completion time of the jobs. In particular, we proposed a branch-cut-and-price algorithm with several ingredients such as arc fixing by reduced cost, dual stabilization, strong branching and rank-1-based (non-robust) cuts that is capable of solving problem $R|r_j, s^k_{ij}|\sum f_j(C_j)$ and its particular cases. Extensive computational experiments carried out on benchmark instances of different variants showed that our algorithm managed to find the optimal solutions of many open problems.

We also conducted an analysis to evaluate the impact of using rank-1 cuts in the algorithm. The results obtained indicated that separation of this cuts allows one to reduce significantly the root gap and the number of nodes in the branch-and-price algorithm. The non-robust cuts helped

us to solve many instances where the pure branch-and-price approach failed to determine the optimal solution. These rank-1 cuts are especially beneficial for the case when 1) the number of jobs per machine is relatively large, 2) the objective function is non-regular, 3) the sequence-dependent setup times are present. For some easier instances however, separation of these cuts does not pay off. So the general advice is to run first a pure branch-and-price algorithm for given instances. In the case it is not efficient enough, one should try to generate rank-1 cuts.

The most natural research direction for us is to decrease the running time of the proposed branch-cut-and-price algorithm, which may be slow for large instances. We see an opportunity in dropping the time discretization approach. For that purpose the labeling algorithm should be modified, for example using the technique of Ioachim et al (1998).

A future research should also focus on extending the proposed approach for solving more practical problems with additional constraints, see for example Bitar et al (2016). Often additional constrains in practice take the form of precedence relations and their variant such as time lags and synchronisation. Dealing with this problem generalisation is very challenging. However, any success in this direction will open a large scope of applications.

Acknowledgments

We would like to thank Artur Pessoa for the valuable comments and suggestions. This work is partially supported by the Brazilian research agency CNPq, grants 305223/2015-1 and 428549/2016-0.

Experiments presented in this paper were carried out using the PlaFRIM (Federative Platform for Research in Computer Science and Mathematics), created under the Inria PlaFRIM development action with support from Bordeaux INP, LABRI and IMB and other entities: Conseil Régional d'Aquitaine, Université de Bordeaux, CNRS and ANR in accordance to the "Programme d'Investissements d'Avenir" (see `www.plafrim.fr/en/home`).

References

Achterberg T (2007) Constraint integer programming. PhD thesis, Technische Universitat Berlin

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

Baldacci R, Mingozzi A, Roberti R (2011) New route relaxation and pricing strategies for the vehicle routing problem. Operations research 59(5):1269–1283

Bigras LP, Gamache M, Savard G (2008) The time-dependent traveling salesman problem and single machine scheduling problems with sequence dependent setup times. Discrete Optimization 5(4):685–699

Bitar A, Dauzère-Pérès S, Yugma C, Roussel R (2016) A memetic algorithm to solve an unrelated parallel machine scheduling problem with auxiliary resources in semiconductor manufacturing. Journal of Scheduling 19(4):367–376

Bülbül K, Şen H (2017) An exact extended formulation for the unrelated parallel machine total weighted completion time problem. Journal of Scheduling 20(4):373–389

Chen ZL, Powell WB (1999) Solving parallel machine scheduling problems by column generation. INFORMS Journal on Computing 11(1):78–94

Contardo C, Martinelli R (2014) A new exact algorithm for the multi-depot vehicle routing problem under capacity and route length constraints. Discrete Optimization 12:129–146

Gauvin C, Desaulniers G, Gendreau M (2014) A branch-cut-and-price algorithm for the vehicle routing problem with stochastic demands. Computers & Operations Research 50:141–153

Graham R, Lawler E, Lenstra J, Kan A (1979) Optimization and approximation in deterministic sequencing and scheduling: a survey. In: PL Hammer EJ, Korte B (eds) Discrete Optimization II Proceedings of the Advanced Research Institute on Discrete Optimization and Systems Applications of the Systems Science Panel of NATO and of the Discrete Optimization Symposium co-sponsored by IBM Canada and SIAM Banff, Aha. and Vancouver, Annals of Discrete Mathematics, vol 5, Elsevier, pp 287 – 326

Ioachim I, Gélinas S, Soumis F, Desrosiers J (1998) A dynamic programming algorithm for the shortest path problem with time windows and linear node costs. Networks 31(3):193–204

Jepsen M, Petersen B, Spoorendonk S, Pisinger D (2008) Subset-row inequalities applied to the vehicle-routing problem with time windows. Operations Research 56(2):497–511

Jepsen M, Spoorendonk S, Ropke S (2013) A branch-and-cut algorithm for the symmetric two-echelon capacitated vehicle routing problem. Transportation Science 47(1):23–37

Jouglet A, Savourey D (2011) Dominance rules for the parallel machine total weighted tardiness scheduling problem with release dates. Computers & Operations Research 38(9):1259 – 1266

Kowalczyk D, Leus R (2018) A branch-and-price algorithm for parallel machine scheduling using zdds and generic branching. INFORMS Journal on Computing Forthcoming

Kramer A (2015) Um método heurístico para a resolução de uma classe de problemas de sequenciamento da produção envolvendo penalidades por antecipação e atraso. Master's thesis, Programa de Pós-Graduação em Engenharia de Produção, Universidade Federal da Paraíba, João Pessoa, Brazil, in Portuguese

Kramer A, Subramanian A (2017) A unified heuristic and an annotated bibliography for a large class of earliness-tardiness scheduling problems. Journal of Scheduling First Online

Lenstra J, Kan AR, Brucker P (1977) Complexity of machine scheduling problems. In: PL Hammer BK EL Johnson, Nemhauser G (eds) Studies in Integer Programming, Annals of Discrete Mathematics, vol 1, Elsevier, pp 343–362

Liaw CF, Lin YK, Cheng CY, Chen M (2003) Scheduling unrelated parallel machines to minimize total weighted tardiness. Computers & Operations Research 30(12):1777 – 1789

Nessah R, Yalaoui F, 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 35(4):1176 – 1190

Oliveira D, Pessoa A (2018) An improved branch-cut-and-price algorithm for parallel machine scheduling problems. INFORMS Journal on Computing Forthcoming

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

Pecin D, Contardo C, Desaulniers G, Uchoa E (2017a) New enhancements for the exact solution of the vehicle routing problem with time windows. INFORMS Journal on Computing 29(3):489–502

Pecin D, Pessoa A, Poggi M, Uchoa E (2017b) Improved branch-cut-and-price for capacitated vehicle routing. Mathematical Programming Computation 9(1):61–100

Pereira Lopes MJ, Valério de Carvalho J (2007) A branch-and-price algorithm for scheduling parallel machines with sequence dependent setup times. European Journal of Operational Research 176(3):1508 – 1527

Pessoa A, Uchoa E, de Aragão M, Rodrigues R (2010) Exact algorithm over an arc-time-indexed formulation for parallel machine scheduling problems. Mathematical Programming Computation 2(3-4):259–290

Pessoa A, Sadykov R, Uchoa E, Vanderbeck F (2018) Automation and combination of linear-programming based stabilization techniques in column generation. INFORMS Journal on Computing 30(2):339–360

Poggi de Aragão M, Uchoa E (2003) Integer program reformulation for robust branch-and-cut-and-price. In: Wolsey L (ed) Annals of Mathematical Programming in Rio, Búzios, Brazil, pp 56–61

Righini G, Salani M (2006) Symmetry helps: Bounded bi-directional dynamic programming for the elementary shortest path problem with resource constraints. Discrete Optimization 3(3):255 – 273

Røpke S (2012) Branching decisions in branch-and-cut-and-price algorithms for vehicle routing problems. Presentation In Column Generation 2012

Schaller JE (2014) Minimizing total tardiness for scheduling identical parallel machines with family setups. Computers & Industrial Engineering 72(0):274 – 281

Şen H, Bülbül K (2015) A strong preemptive relaxation for weighted tardiness and earliness/tardiness problems on unrelated parallel machines. INFORMS Journal on Computing 27(1):135–150

Shim SO, Kim YD (2007a) Minimizing total tardiness in an unrelated parallel-machine scheduling problem. Journal of the Operational Research Society 58(3):346 – 354

Shim SO, Kim YD (2007b) Scheduling on parallel identical machines to minimize total tardiness. European Journal of Operational Research 177(1):135 – 146

Sourd F (2005) Earliness-tardiness scheduling with setup considerations. Computers & Operations Research 32(7):1849 – 1865

Sourd F, Kedad-Sidhoum S (2003) The one-machine problem with earliness and tardiness penalties. Journal of Scheduling 6(6):533–549

Sourd F, Kedad-Sidhoum S (2008) A faster branch-and-bound algorithm for the earliness-tardiness scheduling problem. Journal of Scheduling 11(1):49–58

Tanaka S, Araki M (2008) A branch-and-bound algorithm with lagrangian relaxation to minimize total tardiness on identical parallel machines. International Journal of Production Economics 113(1):446 – 458

Tanaka S, Araki M (2013) An exact algorithm for the single-machine total weighted tardiness problem with sequence-dependent setup times. Computers & Operations Research 40(1):344–352

Tanaka S, Fujikuma S (2008) An efficient exact algorithm for general single-machine scheduling with machine idle time. In: Automation Science and Engineering, 2008. CASE 2008. IEEE International Conference on, pp 371–376

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

Tanaka S, Fujikuma S, Araki M (2009) An exact algorithm for single-machine scheduling without machine idle time. Journal of Scheduling 12(6):575–593

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

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