A stand-alone branch-and-price algorithm for identical parallel machine scheduling with conflicts

Nicola Bianchessi*, Emanuele Tresoldi

Dipartimento di Informatica, Università degli Studi di Milano, Via Celoria 18, I-20133 Milan, Italy.

Abstract

We consider a scheduling problem where a set of n jobs has to be processed in a non-preemptive way on a set of m identical parallel machines. Each job j is associated with a positive integer processing time p_j. The problem is also characterized by a conflict graph where adjacent nodes in the graph represent conflicting jobs that cannot be processed on the same machine. A schedule is an assignment of a time interval of p_j time units on m machines for each job j. The schedule is feasible if intervals on the same machine do not overlap, and jobs to be processed on the same machine are pairwise not conflicting. The aim is to find a feasible schedule that minimizes the maximum completion time of the jobs. The problem is NP-hard as it generalizes both the problem of scheduling jobs on identical parallel machines to the aim of minimizing the makespan ($P||C_{max}$) and the vertex coloring problem (VCP), two well-known NP-hard problems. We present the first stand-alone branch-and-price (BP) algorithm that works directly on the problem. In comprehensive computational experiments with benchmark instances, we prove that the new BP algorithm, even without using ad-hoc primal heuristics and feasibility check algorithms, is competitive with the best exact solution algorithm proposed so far in the literature. These results are mainly achieved thanks to the branching scheme we propose.

Key words: Scheduling, Identical parallel machine, Conflict graph, Agreement graph, Branch-and-price

1. Introduction

We consider a scheduling problem where a set $N = \{1, \dots, n\}$ of jobs has to be processed in a non-preemptive way on a set $M = \{1, \dots, m\}$ of identical parallel machines. Each job $j \in N$ is associated with a positive integer processing time p_j. A conflict graph $G = (N, E)$ defined over the set of jobs is also part of the problem description. Each edge $(j', j'') \in E$ models the impossibility of processing jobs j' and j'' on the same machine; j' and j'' are said to be conflicting jobs. A *schedule* is an assignment of a time interval of p_j time units on the machines for each job $j \in N$. The schedule is *feasible* if intervals on the same machine do not overlap, and $(j', j'') \notin E$ for each pair of jobs j' and j'' processed on the same machine. The aim is to find a feasible schedule that minimizes the maximum completion time of the jobs (i.e., the makespan).

The problem is NP-hard as it generalizes both the $P||C_{max}$ scheduling problem (i.e., the problem of scheduling jobs on identical parallel machines to the aim of minimizing the maximum makespan, denoted by means of the three-field notation introduced in (Graham *et al.*, 1979)) and the vertex coloring problem (VCP), two well-known NP-hard problems (Garey and Johnson, 1979). This characterization is first due to Bodlaender *et al.* (1994) who introduced the problem into the literature as the *scheduling with incompatible jobs problem.* Additionally, they proposed polynomial time approximation schemes (PTAS) for solving the problem in some special cases arising with particular structures of the conflict graph.

Recently, Kowalczyk and Leus (2015, 2017) addressed the problem referring to it as the *parallel machine scheduling problem with conflicts* (PMC). In their papers Kowalczyk and Leus (2015, 2017) define the PMC as "theoretically important" and describe several applications in many different fields such as the scheduling of computing services, TV advertisement scheduling and resource assignment in workforce planning. We refer the reader to the original papers for an extensive description of these applications.

*Corresponding author.
Email address: `bianchessi@di.unimi.it` (Nicola Bianchessi)

In the literature there are a few problems that share many similarities with PMC. For instance, the *problem of scheduling with bag constraints* (Page and Solis-Oba, 2018) and the *scheduling of identical jobs on uniform machines with a conflict graph* (Mallek *et al.*, 2019). In the former the set of jobs is partitioned in disjointed subsets called *bags* and no two jobs from the same bag are allowed to be assigned on the same machine. The latter considers a different configuration for the machines, uniform instead of identical and only jobs with unit processing times. Unfortunately, since these problems only represent special cases of PMC, the interesting results obtained cannot be directly applied to PMC. It is worth noting that, in the literature, the name *scheduling with conflicts* (Even *et al.*, 2009) and *scheduling with conflict graphs* (Hong and Lin, 2018) are also used for other problems not directly related to PMC. These problems consider a different definition of conflicts: if two jobs are in conflict, they cannot be scheduled concurrently on two different machines. The structure of the resulting problem is very different from PMC.

(Kowalczyk and Leus, 2015) presented compact mixed-integer linear programming (MILP) models for PMC defined by using binary variables that assign jobs to machines. The authors experimentally shown how addressing these kind of models is not effective, even when symmetry-breaking constraints (SBCs) are included into the formulation. This is in line with what happens for other scheduling problem variants (Unlu and Mason, 2010; Berghman *et al.*, 2014; Yu and Hung, 2016). For the latter scheduling problems, flow-based formulations may represent better alternatives (see, e.g., Mallek *et al.*, 2019). However, as for PMC, additional variables and/or constraints have to be considered in order to handle conflict constraints, making unattractive addressing these kind of formulations in this specific case. According to this line of arguments, Kowalczyk and Leus (2017) proposed the first effective exact algorithm for the solution of the problem. The algorithm exploits the relation between PMC and the bin packing problem with conflicts (BPPC), where the latter consists of packing items of given sizes, and that are pairwise in conflict, in a minimum number of bins with a limited capacity, such that two items in conflict are not in the same bin. In particular, the algorithm is inspired by the one presented in (Dell'Amico *et al.*, 2008) for the solution of $P||C_{max}$ and proceeds as follows. First, a lower bound $\underline{z}^*$ on the optimal solution value z^* is computed by exploiting known bounds for $P||C_{max}$. Then, an upper bound $\overline{z}^*$ is computed by applying two greedy heuristics and an improving local search algorithm. Additionally, if the heuristics fail in finding a feasible solution, a non-trivial feasibility check algorithm is run to see whether the problem instance is infeasible or not. When the instance is feasible, an iterative binary search algorithm is started to identify the optimal solution value. At each iteration, the condition $\lceil \underline{z}^* \rceil = \overline{z}^*$ is checked to see if the instance has been solved to optimality. If not, an instance of the BPPC with bins of capacity $\tilde{z}^* = \lfloor (\underline{z}^* + \overline{z}^*)/2 \rfloor$ is solved by means of a branch-and-price (BP) algorithm. The BP algorithm (and so the iteration) terminates as soon as the lower bound on the optimal value is greater than m or a feasible solution to the BPPC instance is found. In particular, to the aim of finding a feasible solution, a diving search heuristic is also run at every node of the tree (before branching is applied). When the BP algorithm terminates, if a feasible solution is found, then $\overline{z}^*$ is set to $\tilde{z}^*$, otherwise the value of $\underline{z}^*$ is updated to $\tilde{z}^* + 1$. A final remark concerns the feasibility check algorithm. First, several lower bounds on the chromatic number of the conflict graph G are computed. When all these bounds are less than, or equal to, m, the VCP associated with G is solved by means of an additional BP algorithm, also in this case making use of a diving search heuristic run at every node of the tree. This additional BP algorithm terminates either proving the infeasibility of the instance or describing a feasible colouring of G using at most m colours, i.e., an initial feasible solution to the PMC instance.

In this paper we focus on the exact solution of PMC. Our contribution is threefold: (i) First, we present the first stand-alone BP algorithm that works directly on the problem, embedding just an optional, straightforward, restricted master heuristic. In comprehensive computational experiments with benchmark instances, we prove that the new BP algorithm is highly competitive with the algorithm proposed by Kowalczyk and Leus (2017). (ii) While devising the BP algorithm, we define a branching scheme that does not introduce symmetries in the solution space and allows the column generation subproblems to share the same feasible region at every node of the branch-and-bound tree. This, in turn, allows to design the column generation iteration to potentially avoid solving some of the subproblems. (iii) Finally, we show that new BP algorithm is in line with state-of-the-art exact methods for $P||C_{max}$ with respect to some classes of benchmark instances.

The rest of the paper is organized as follows. Section 2 describes the new stand-alone branch-and-price algorithm we implemented to address PMC, with subsections on the extensive formulation, the column generation algorithm, and the branching scheme. In Section 3, we experimentally evaluate the BP algorithm using benchmark instances for PMC and $P||C_{max}$. Final conclusions are drawn in Section 4.

2. Branch-and-price algorithm

In this section we present the main features and components of the new, stand-alone, branch-and-price (BP) algorithm for solving PMC.

2.1. An extended set-covering formulation

Define S as the set of feasible schedules on any single machine. For each schedule $s \in S$, let a_j^s be a binary parameter equal to 1 if job $j \in N$ is included in s, 0 otherwise. The makespan associated with the schedule is defined as $c^s = \sum_{j \in N} p_j a_j^s$ and is independent of the actual sequence of the jobs included in the schedule. For each $s \in S$ and each $i \in M$, define a binary variable λ_i^s that takes value 1 if schedule s is chosen for machine i, 0 otherwise. Finally, let C_i be the makespan of the schedule assigned to machine $i \in M$. We assume, w.l.o.g., that schedules are assigned to machines from 1 to m in decreasing order of their makespan. Hence, machines 1 and m are assigned to the schedules having respectively the largest and the smallest makespan. This allows to reduce symmetries in the solution space.

Using this notation, PMC can be formulated as follows:

$$
\begin{aligned}
\min \quad & C^1 && && \text{(1a)}\\
s.t. \quad & \sum_{i \in M} \sum_{s \in S} a_j^s \lambda_i^s \geq 1 && j \in N && \text{(1b)}\\
& \sum_{s \in S} \lambda_i^s \leq 1 && i \in M && \text{(1c)}\\
& \sum_{s \in S} c^s \lambda_i^s \leq C^i && i \in M && \text{(1d)}\\
& C^i \geq C^{i+1} && i \in \{1, \dots, m-1\} && \text{(1e)}\\
& \lambda_i^s \in \{0,1\} && i \in M,\ s \in S && \text{(1f)}\\
& C^i \geq 0 && i \in M && \text{(1g)}
\end{aligned}
$$

The objective function (1a) minimizes the largest makespan. This is ensured by constraints (1d) together with (1e). Constraints (1d) define the makespans for all the machines. Then, constraints (1e) impose the makespans associated with machines from 1 to m to be sorted in non-increasing order. Constraints (1b) impose each job to be included in a schedule assigned to a machine, whereas constraints (1c) allow to assign at most one schedule to each machine. Finally, constraints (1f) and (1g) define the domains of the variables.

In the BP algorithm, at each node of the branch-and-bound tree the linear relaxation of (1), eventually augmented by constraints arising from branching decisions, is solved iteratively by means of column generation. The starting point is a linear relaxation of (1) defined over a subset $\bar{S} \subseteq S$ of the feasible machine schedules, called reduced master program (RMP). At each iteration, column generation alternates between the optimization of the RMP and the solution of column generation subproblems. The former allows to retrieve optimal dual variable values w.r.t. set $\bar{S}$. The latter, on the basis of the dual variable values, generates negative reduced cost schedule variables λ_i^s to be included in the RMP, if any. When no negative reduced cost variable is found, the optimal solution of the RMP is also the optimal solution of the linear relaxation of (1) (Desaulniers *et al.*, 2005). Branching is finally required to ensure the integrality of the solution.

2.2. Column generation

Let $\mu_j \in \mathbb{R}_+$ be the dual variable associated with constraints (1b) of job $j \in N$. Define $\theta_i \in \mathbb{R}_-$ and $\rho_i \in \mathbb{R}_-$ as the dual variables associated with constraints (1c) and (1d), respectively, of machine $i \in M$. Finally, let $\sigma_i \in \mathbb{R}_+$ represent the dual variables associated with constraints (1e) for $i \in \{1, \dots m-1\}$.

The dual of the linear relaxation of (1) is:

$$
\begin{aligned}
\max \quad & \sum_{j \in N} \mu_j + \sum_{i \in M} \theta_i && && \text{(2a)}\\
s.t. \quad & \sum_{j \in N} a_j^s \mu_j + \theta_i + c_s \rho_i \leq 0 && i \in M,\ s \in S && \text{(2b)}\\
& -\rho_1 + \sigma_1 \leq 1 && && \text{(2c)}
\end{aligned}
$$

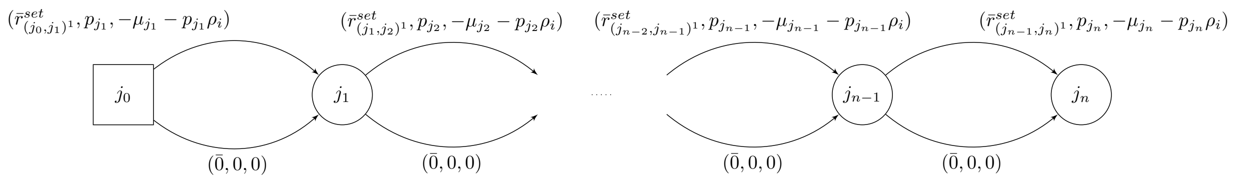

$(\bar{r}^{set}_{(j_0,j_1)^1}, p_{j_1}, -\mu_{j_1} - p_{j_1}\rho_i)$
$(\bar{r}^{set}_{(j_1,j_2)^1}, p_{j_2}, -\mu_{j_2} - p_{j_2}\rho_i)$
$(\bar{r}^{set}_{(j_{n-2},j_{n-1})^1}, p_{j_{n-1}}, -\mu_{j_{n-1}} - p_{j_{n-1}}\rho_i)$
$(\bar{r}^{set}_{(j_{n-1},j_n)^1}, p_{j_n}, -\mu_{j_n} - p_{j_n}\rho_i)$
j_0
j_1
j_{n-1}
j_n
$(\bar{0}, 0, 0)$
$(\bar{0}, 0, 0)$
$(\bar{0}, 0, 0)$
$(\bar{0}, 0, 0)$

Figure 1: Multi-graph $\mathcal{G}^i = (\mathcal{V}, \mathcal{A})$ for machine $i \in M$.

$$-\rho_i - \sigma_{i-1} + \sigma_i \leq 0 \qquad i \in \{2, \dots, m-2\} \tag{2d}$$

$$-\rho_m - \sigma_{m-1} \leq 0 \tag{2e}$$

$$\mu_j \geq 0 \qquad j \in N \tag{2f}$$

$$\theta_i, \rho_i \leq 0 \qquad i \in M \tag{2g}$$

$$\sigma_i \geq 0 \qquad i \in \{1, \dots, m-1\} \tag{2h}$$

Thus, for PMC there is one distinct subproblem for each machine $i \in M$. In particular, given the duals $(\boldsymbol{\mu}, \boldsymbol{\theta}, \boldsymbol{\rho}, \boldsymbol{\sigma})$, the subproblem for machine $i \in M$ consists of finding a minimum reduced cost schedule to be assigned to the machine, where the reduced cost $\bar{c}_i^s(\boldsymbol{\mu}, \boldsymbol{\rho})$ of schedule $s \in S$ to be assigned to the machine is defined as

$$\bar{c}_i^s(\boldsymbol{\mu}, \boldsymbol{\rho}) = -\sum_{j \in N} a_j^s \mu_j - c_s \rho_i \tag{3}$$

The solution represents a negative reduced cost variable if its value is less than θ_i.

It is worth noting that, when an upper bound $\overline{C}^1$ is available for C^1, the subproblem associated with each machine $i \in M$ can be formulated as a knapsack problem with general conflict graph:

$$\min \quad -\sum_{j \in N} x_j \mu_j - \rho_i \sum_{j \in N} x_j p_j \tag{4a}$$

$$s.t. \quad \sum_{j \in N} x_j p_j \leq \overline{C}^1 - 1 \tag{4b}$$

$$x_{j'} + x_{j''} \leq 1 \qquad (j', j'') \in E \tag{4c}$$

$$x_j \in \{0, 1\} \qquad j \in N \tag{4d}$$

in which binary variable x_j, $j \in N$, takes value 1 if job j is part of the schedule assigned to machine i, 0 otherwise. The subproblems are thus strongly NP-hard (Pferschy and Schauer, 2009) and have identical feasible regions. In Section 2.3 we will see how the designed branching rules keep identical the solution spaces of the subproblems at each node of the branch-and-bound tree.

The subproblems can be addressed by solving directly formulation (4) with general-purpose MIP-solvers. Alternatively, an ad-hoc solution technique can be designed by exploiting the subproblem features. This is illustrated in Section 2.2.1.

2.2.1. Alternative subproblem modeling

The subproblem arising for each machine $i \in M$ can be addressed as a shortest path problem with resource constraints (SPPRC) on a directed, acyclic, multi-graph $\mathcal{G}^i = (\mathcal{V}, \mathcal{A})$, with vertex set $\mathcal{V} = \{j_0\} \cup \mathcal{N}$ and arc set $\mathcal{A}$. The vertices $\mathcal{N} = \{j_1, \dots, j_n\}$ represent the jobs, arbitrarily ordered, to be scheduled on the machines, and (source) vertex j_0 represents a dummy job with which each schedule starts, with $p_0 = 0$. Set $\mathcal{A}$ includes two ingoing arcs $(j_{k-1}, j_k)^0$ and $(j_{k-1}, j_k)^1$ for each vertex $j_k \in \mathcal{N}$ (see Figure 1).

Each path P from j_0 to j_n in graph $\mathcal{G}^i = (\mathcal{V}, \mathcal{A})$ represents a machine schedule including all jobs $j_k \in \mathcal{N}$ such that arc $(j_{k-1}, j_k)^1$ is traversed along the path. Let $\mathcal{N}(P)$ be the set of such jobs. The path P is feasible if $(j', j'') \notin E$ for each pair of jobs j' and j'' included in $\mathcal{N}(P)$. Feasibility is ensured by considering an order set of binary resources, one for each job $j_k \in \mathcal{N}$, and vectors $\bar{r}^{set}_{(j_{k-1}, j_k)} \in \mathbb{B}^{|\mathcal{N}|}$ that model the consumptions of the binary resources along each arc in $(j_{k-1}, j_k) \in \mathcal{A}$. Along arcs $(j_{k-1}, j_k)^1$, $j_k \in \mathcal{N}$, $\bar{r}^{set}_{(j_{k-1}, j_k)^1}$ implies unitary consumptions for the resources associated with jobs in $\{j_r \in \mathcal{N} \mid j_r = j_k \ \vee \ (j_r, j_k) \in E\}$. For arcs $(j_{k-1}, j_k)^0$, $j_k \in \mathcal{N}$, $\bar{r}^{set}_{(j_{k-1}, j_k)^0}$ is set to $\bar{0} \in \mathbb{B}^{|\mathcal{N}|}$. A path is feasible if the consumption corresponding to each binary resource is less than or equal to 1. Then, arcs $(j_{k-1}, j_k)^1$, $j_k \in \mathcal{N}$, are associated with processing time p_{j_k} and cost $-\mu_{j_k} - p_{j_k}\rho_i$, while arcs $(j_{k-1}, j_k)^0$, $j_k \in \mathcal{N}$, are associated with zero processing time and zero cost. This allows to correctly compute the makespan and cost associated with (the schedule corresponding to) each path. Finally, when an upper bound $\overline{C}^1$ is available for C^1, the processing times associated with the arcs in $\mathcal{A}$ can be interpreted as the consumptions of an additional resource taking values in $\mathbb{Z}_+$. In a feasible path, the consumption of the additional resource has to be less than $\overline{C}^1$.

The subproblem for machine $i \in M$ consists in finding a least cost feasible path P from j_0 to j_n in $\mathcal{G}^i$. The solution represents a negative reduced cost variable if its value is less than θ_i.

Dynamic programming algorithm. When modelled as SPPRCs, the subproblems can be solved by means of a *label setting dynamic programming algorithm* (see, e.g., Irnich and Desaulniers (2005)). In such kind of algorithms, each partial path from the source of the graph to vertex i is represented by a label $(i, \overline{R}, C)$. Vector $\overline{R}$ has as many components as the number of different resources characterizing the problem. Each component represents the consumption of the corresponding resource along the path up to vertex i, whereas C is the cost of the path. When a partial path associated with label $(i, \overline{R}, C)$ is extended along arc (i, j), a new label $(j, \overline{R}', C')$ associated with the partial path reaching vertex j is defined. The components of $\overline{R}'$ and the cost C' are defined based on $\overline{R}$ and C, respectively, taking into account the resource consumption and the cost associated with the arc (i, j). Extensions must be feasible with respect to the consumption of all the resources. To avoid enumerating all feasible paths, dominance rules can be applied to discard dominated partial paths reaching the same vertex. A solution is a path represented by a minimum cost label associated with the sink vertex.

For a given machine $i \in M$, let P be the partial path starting at vertex $j_0 = 0$ and ending at vertex $j_k \in \mathcal{V}$ in graph $\mathcal{G}^i$. We associate with P a label $L = L(P)$ that includes the following attributes:

j_k:	The last vertex of P;
$\overline{R}^{set} \in \mathbb{B}^{\lvert\mathcal{N}\rvert}$:	A vector with binary components modelling the set of not schedulable jobs through extensions of path P; the k-th component of the vector is associated with job $j_k \in \mathcal{N}$;
$C \in \mathbb{R}$:	The accumulated costs along path P.

Therefore, $\overline{R}$ includes for the moment only $|\mathcal{N}|$ components. Later, $\overline{R}$ will be extended to take into account additional constraints.

The initial label at vertex j_0 is defined as $(0, \bar{0}, 0)$.

A feasible path $P = (j_0, \ldots, j_k)$ associated with label $L(P) = (j_k, R^{set}, C)$ can be extended along arc $(j_k, j_{k+1})^1 \in \mathcal{A}$ if $R^{set}_{j_{k+1}} = 0$. When the extension is feasible, the new label $L(P') = (j_{k+1}, \overline{R}^{set'}, C')$ for $P' = (j_0, \ldots, j_k, j_{k+1})$ results from the following extension rules:

$$R_t^{set'} = \begin{cases} 1 & \text{if } t = k+1 \vee (j_{k+1}, j_t) \in E \\ R_t^{set} & \text{otherwise} \end{cases} \qquad \text{for } k+1 < n \tag{5a}$$

$$R_t^{set'} = \begin{cases} 1 & \text{if } j_t \in \mathcal{N}(P') \\ 0 & \text{otherwise} \end{cases} \qquad \text{for } k+1 = n \tag{5b}$$

$$C' = C - \mu_{j_k} - p_{j_k}\rho_i \tag{5c}$$

The extension of path $P = (j_0, \ldots, j_k)$ along arc $(j_k, j_{k+1})^0 \in \mathcal{A}$ is always feasible. The attributes of the label $L(P')$ are defined as follows:

$$R_t^{set'} = \begin{cases} 1 & \text{if } t = k+1 \\ R_t^{set} & \text{otherwise} \end{cases} \qquad \text{for } k+1 < n \tag{5d}$$

$$R_t^{set'} = \begin{cases} 1 & \text{if } j_t \in \mathcal{N}(P') \\ 0 & \text{otherwise} \end{cases} \qquad \text{for } k+1 = n \tag{5e}$$

$$C' = C \tag{5f}$$

As for dominance, let $L' = (j_k, \overline{R}^{set'}, C')$ and $L'' = (j_k, \overline{R}^{set''}, C'')$ be two labels representing two different paths ending at the same vertex $j_k \in \mathcal{V}$. Label L' dominates label L'' if both the following conditions are satisfied:

$$R_k^{set'} \leq R_k^{set''} \qquad j_k \in \mathcal{N} \tag{6a}$$

$$C' \leq C'' \tag{6b}$$

Note that, in (5b) and (5e), $\overline{R}^{set'}$ is set to the incidence vector of $\mathcal{N}(P')$ only to eventually increase the cardinality of the pareto-optimal path set associated with vertex j_n.

Maximum makespan constraint. Whenever an upper bound $\overline{C}^1$ is available for C^1, it is possible to easily integrate the corresponding maximum makespan constraint into the subproblems as follows. An additional component R^C (with values in $\mathbb{Z}_+$) is included into the vector $\overline{R}$ of each label $L = (j_i, \overline{R}, C)$. Initially, the component is set to 0. An extension along arcs $(j_k, j_{k+1})^1 \in A^i$ increases the component value by $p_{j_{k+1}}$, while the component value is simply copied in the new label in the other cases. The extension is feasible if the resulting value of the component R^C is less than, or equal to, $\overline{C}^1 - 1$. Dominance between two labels $L' = (j_k, \overline{R}', C')$ and $L'' = (j_k, \overline{R}'', C'')$ has to additionally check $R^{C'} \leq R^{C''}$.

Additionally, we sort the jobs in $\mathcal{N}$ in non-decreasing order according to their processing times. Thus, whenever path $P = (j_0, \ldots, j_k)$ cannot be feasibly extended along arc $(j_k, j_{k+1})^1 \in \mathcal{A}$, with $L(P) = (j_k, \overline{R}^{set}, R^C, C)$, $R^{set}_{k+1} = 0$, and $R^C + p_{j_{k+1}} > \overline{C}^1 - 1$, we directly extend the path towards vertex j_n along a fictitious arc $(j_k, j_n)^0$.

2.2.2. Column generation iteration

Let $\mathcal{M} = \{i_1, \ldots, i_m\}$ be the set of machines sorted in non-increasing order w.r.t. the corresponding ρ_{i_k} values. At each column generation iteration, the subproblems are considered sequentially starting from the problem related to machine i_1. Let $\mathcal{S}^{i_k}$ be the set of pareto-optimal solutions for the subproblem associated with machine $i_k \in \mathcal{M}$. For each solution in $\mathcal{S}^{i_k}$, we check if it represents a negative reduced cost variable (column) for any of the machines. Doing this, given that all subproblems share the same feasible region, we avoid to solve subsequent subproblems corresponding to machines i_t, $t > k$, such that $|\rho_{i_k} - \rho_{i_t}| < \epsilon$, with $\epsilon \to 0^+$. (This applies when the subproblems are addressed whether by means of the dynamic programming algorithm or by means of a MIP-solver that directly solve formulation (4)). The iteration terminates as soon as negative reduced cost columns are found after the resolution of a subproblem.

2.2.3. Partial pricing

At each column generation iteration, two heuristic algorithms are considered in sequence before solving the subproblems to optimality. They both consist in applying an heuristic version of the dynamic programming algorithm outlined in Section 2.2.1. The first heuristic applies the dominance rules by ignoring condition (6a). The second heuristic is run only if the first does not succeed in finding negative reduced cost columns. In particular, it extends at most $\overline{L}$ labels along each arc $(j_k, j_{k+1})^1 \in \mathcal{A}$, $0 \leq k < n$, the $\overline{L}$ labels associated with the smallest costs. When both heuristics fail, the subproblems are solved to optimality.

2.2.4. Lower bounds

Lower bounds for $P||C_{max}$ are also valid for PMC. Before starting column generation, we compute the following lower bounds proposed in Dell'Amico and Martello (1995):

$$C^1_0 = \left\lceil \frac{\sum_{j_i \in \widetilde{\mathcal{N}}} p_{j_i}}{|\widetilde{\mathcal{N}}|} \right\rceil \tag{7a}$$

$$C^1_1 = \max\{C^1_0, p_{j_{|\widetilde{\mathcal{N}}|}}\} \tag{7b}$$

$$C^1_2 = \max\{C^1_1, p_{j_{|\widetilde{\mathcal{N}}|-m}} + p_{j_{|\widetilde{\mathcal{N}}|-(m+1)}}\} \tag{7c}$$

where $\widetilde{\mathcal{N}}$ is the set of jobs sorted in non-decreasing order according to their processing times, and $p_{j_{|\widetilde{\mathcal{N}}|-m}}$ and $p_{j_{|\widetilde{\mathcal{N}}|-(m+1)}}$ are the m-th and m+1-st largest processing times, respectively.

Whenever an upper bound $\overline{C}^1$ is available for C^1, if $C^1_2 > \overline{C}^1$ the current branch-and-bound node can be fathomed without generating any column. Otherwise, constraint (1g) for C^1 is substituted by $C^1 \geq C^1_2$. From preliminary experimental results, this allows column generation to converge faster in finding the optimal solution.

As will become clear in Section 2.3, the set of jobs may differ from one node to another due to branching. This is the reason why lower bounds computation is performed before each column generation run.

2.2.5. *Restricted master heuristic*

In order to speed up the BP algorithm, we embed into column generation a *restricted master heuristic* as defined in (Joncour *et al.*, 2010, p. 697). The basic idea behind restricted master heuristics is to solve, by means of a general mixed integer linear programming (MILP) solver, the master problem (1) restricted to a subset of the generated columns.

We run the restricted master heuristic each time we compute an optimal solution for the linear relaxation of (1). More precisely: *(i)* We consider the set $\bar{S}$ of the schedules associated with the columns λ_i^s. *(ii)* The schedules in $\bar{S}$ are then used to initialize an integer program slightly different from (1), using binary variables λ^s no more indexed by machine and assuming value 1 if schedule s is selected to be assigned to one of the machines:

$$\min \; C \tag{8a}$$
$$s.t. \; \sum_{s\in S} a_j^s \lambda^s \geq 1 \qquad j \in N \tag{8b}$$
$$\sum_{s\in S} \lambda^s \leq m \tag{8c}$$
$$c^s\lambda^s \leq C \qquad s \in \bar{S} \tag{8d}$$
$$\lambda^s \in \{0,1\} \qquad s \in \bar{S} \tag{8e}$$
$$C \geq 0 \tag{8f}$$

This allows to avoid symmetries in the solution space at the expense of an increase in the number of constraints (8d) required to define the maximum makespan. *(iii)* Finally, we solve model (8) by means of a general MILP solver.

Whenever an optimal solution to (8) is found, a new upper bound $\overline{C}^1$ (for C^1) becomes available. Thus, column generation is reiterated to solve the linear relaxation of (1) by considering only machine schedules with a makespan smaller than, or equal to, $\overline{C}^1 - 1$.

2.3. *Branching rules*

When the optimal solution $(\widehat{\boldsymbol{\lambda}}, \widehat{\mathbf{C}})$ of the current linear relaxation of (1) is fractional, we apply a three-level hierarchical branching scheme. The rules are presented in the following, in order of priority.

We consider first an adaptation of the branching rule proposed in Ryan and Foster (2003). For each pair of jobs j' and j'', we define $b_{j'j''} = \sum_{i\in M}\sum_{s\in S} a_{j'}^s a_{j''}^s \hat{\lambda}_i^s$ as the sum of the λ_i^s variable values associated with schedules that include both job j' and job j''. We select the fractional value $b_{j'j''}^*$, $0 < b_{j'j''}^* < 1$, closest to 0.5. On one branch we set $b_{j'j''}^* = 0$, meaning that the two jobs must be included into different schedules, while on the other branch we set $b_{j'j''}^* = 1$, meaning that the two jobs must be included into the same schedule. For $b_{j'j''}^* = 0$, it is enough to modify the conflict graph by including the new edge (j', j''). Whereas, for $b_{j'j''}^* = 1$, we merge together jobs j' and j'' into one new job (reducing by 1 the cardinality of set N) and update the conflict graph accordingly.

At the second level, we branch on the number of inclusions of a job into the schedules, i.e., $a_j = \sum_{i\in M}\sum_{s\in S} a_j^s \hat{\lambda}_i^s$. We select a_j^*, $a_j^* > 1$, such that the value $a_j^* - \lfloor a_j^* \rfloor$ is the closest to 0.5. Hence, we create just one child node in which the set-covering constraint (1b) for the job j such that $a_j = a_j^*$ is formulated as a set-partitioning constraint.

When none of the previous branching rules applies, it means that at most $|M|$ distinct schedules are selected in the optimal solution $(\widehat{\boldsymbol{\lambda}}, \widehat{\mathbf{C}})$. Each such distinct schedule $\bar{s}$ may be fractional assigned to different machines, but $\sum_{i\in M} \hat{\lambda}_i^{\bar{s}} = 1$. (This follows from Proposition 1 in Barnhart *et al.* (1998) and contraints (1c); see also (Wolsey, 1998, Section 11.6)). Furthermore, recall that every λ_i^s variable in the RMP represent a schedule s (to be assigned to machine i) which has to be feasible w.r.t. the current upper bound $\overline{C}^1$ available for C^1, meaning that $c^s < \overline{C}^1$. Thus, the optimal solution $(\widehat{\boldsymbol{\lambda}}, \widehat{\mathbf{C}})$ can be converted into a feasible solution to (1) whose value improves the current upper bound $\overline{C}^1$. We give a concrete numerical example to better clarify.

Example 1. *Consider an instance with 10 jobs, associated with processing times 8, 2, 7, 3, 3, 5, 8, 2, 9, and 6, to be scheduled on 5 machines. Assume that the current value of* $\overline{C}^1$ *is 361. Furthermore, consider an optimal fractional solution such that* $s' = \{3\}$, $s'' = \{6, 1, 4, 9, 10, 7\}$ *and* $s''' = \{8, 2, 5\}$ *are the 3 schedules,* *characterized respectively by makespan 97, 264, and 92, associated with the* $\widehat{\boldsymbol{\lambda}}$ *variables. Note that all the makespan are less than* $\overline{C}^1$. *Let the values of the* $\widehat{\boldsymbol{\lambda}}$ *variables be the following:* $\hat{\lambda}_3^{s'} = 0.278041$, $\hat{\lambda}_5^{s'} = 0.721959$, $\hat{\lambda}_3^{s''} = 0.265228$, $\hat{\lambda}_2^{s''} = 0.367386$, $\hat{\lambda}_1^{s''} = 0.367386$, $\hat{\lambda}_4^{s'''} = 1$. *Then,* $\sum_{i\in M} \hat{\lambda}_i^{s'} = \sum_{i\in M} \hat{\lambda}_i^{s''} = \sum_{i\in M} \hat{\lambda}_i^{s'''} = 1$ *and the optimal fractional solution can be converted into a feasible solution to* (1) *with value* $264 < \overline{C}^1 = 361$.

Let $\overline{C'}^1$ be the new upper bound value for C^1. Then, define $\underline{C}^1$ as the lower bound value for C^1 at the current node, i.e., the value of the optimal fractional solution $(\widehat{\boldsymbol{\lambda}}, \widehat{\mathbf{C}})$. When $\overline{C'}^1 - \underline{C}^1 < 1$, the current node can be fathomed. Conversely, whenever $\overline{C'}^1 - \underline{C}^1 \geq 1$, it is necessary to continue exploring the subtree rooted in the current node in order not to lose any feasible solution to (1), i.e., solutions with values $\widetilde{C}^1$ such that $\underline{C}^1 \leq \widetilde{C}^1 < \overline{C'}^1$.

Example 2. *(Continued from Example 1). With respect to the optimal fractional solution described in Example 1,* $\overline{C'}^1 = 264$ *and* $\underline{C}^1 = 264\bar{\lambda}_1^{s''} = \max\{264\bar{\lambda}_1^{s''}, 264\bar{\lambda}_2^{s''}, 97\bar{\lambda}_3^{s'} + 264\bar{\lambda}_3^{s''}, 92\bar{\lambda}_4^{s'''}, 97\bar{\lambda}_5^{s'}\} \approx 96.9899$. *The current node cannot be fathomed as* $\overline{C'}^1 - \underline{C}^1 \geq 1$.

Hence, we create one child node associated with a problem identical to the one associated with the current (father) node, but that will be solved by considering only machine schedules with a makespan smaller than, or equal to, $\overline{C'}^1 - 1$ (see Section 2.2).

These branching rules do not introduce symmetries in the solution space, do not alter the structure of the column generation subproblems, and allow these latter to share the same feasible region at every node of the branch-and-bound tree.

3. Experimental results

All the algorithms have been implemented in C++ using CPLEX 12.7.0 with Concert Technology, and compiled in release mode with MS Visual Studio Professional 2017. The experiments have been performed on a 64-bit Windows 10 PC equipped with an Intel processor i7-6700K clocked at 4.00 GHz and with 32 GB of RAM. As for the parameters mentioned in the description of the column generation iteration and of the partial pricing (see the correponding sections 2.2.2 and 2.2.3), ϵ and $\overline{L}$ have been set equal to 1.0E-9 and 200, respectively. We allowed CPLEX to use at most 6 threads for each run by setting `IloCplex::Param::Threads=6`. We set `IloCplex::ParallelMode=1` in order to force CPLEX to always use deterministic algorithms. CPLEX's default values were kept for all remaining parameters.

In order to finalize the design of the BP algorithm, we run some preliminary experiments on a subset of the 2500 PMC instances used in Kowalczyk and Leus (2017). Then, the performance of the resulting BP algorithm has been assessed on the full set of benchmark instances for PMC and on the 3500 instances for $P||C_{max}$ presented in Mrad and Souayah (2018). The description of the benchmark sets along with the analysis of the results obtained is reported thereafter.

3.1. Instances for PMC

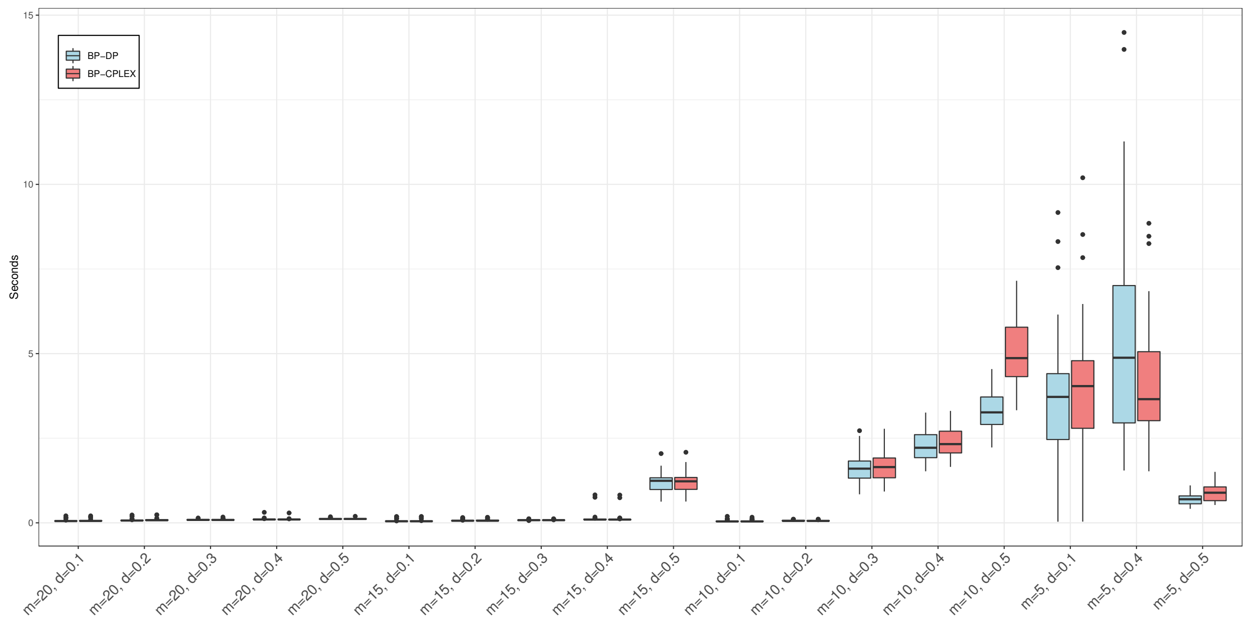
The features of the instances considered in Kowalczyk and Leus (2017) are as follows. The number of machines (m) and jobs (n) take values in sets $\{5, 10, 15, 20\}$ and $\{10, 25, 50, 75, 100\}$, respectively. The processing time of the jobs are all integers and randomly generated taking into account three different *ranges*: (a) = [1, 10], (b) = [1, 50], and (c) = [1, 100]. Finally, the edge set E of the conflict graph $G = (N, E)$ is randomly generated selecting each of the $\frac{n(n-1)}{2}$ possible edges with *density* probability $d \in \{0.1, 0.2, 0.3, 0.4, 0.5\}$. For each combination of n, m, *range* and d, 10 instances have been generated, for a total of 2550 instances.

3.1.1. Preliminary experiments

In order to finalize the design of the BP algorithm, we compared the performance of the proposed dynamic programming (DP) algorithm (see Section 2.2.1) against those of CPLEX solving directly formulation (4).

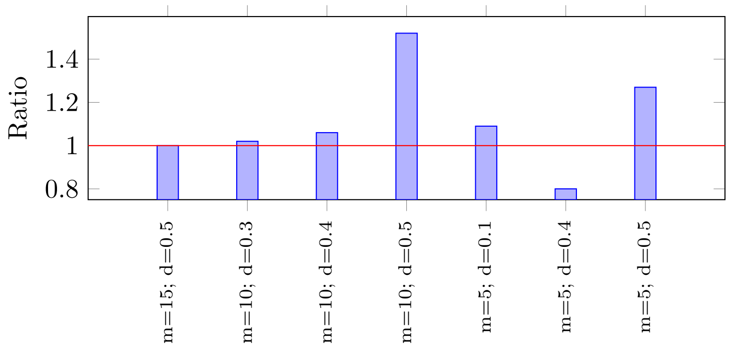
To this aim we considered two fully-fledged versions of the BP algorithm, `BP-DP` and `BP-CPLEX`, that differentiate each other only for the algorithm used to optimally solve the subproblems (the DP algorithm is used in `BP-DP`, and CPLEX solving directly formulation (4) is used in `BP-CPLEX`), and evaluated their performance while solving the linear relaxation of model (1) at the root node of the branch-and-bound tree. In order to perform a meaningful comparison, we concentrated on the instances with $n = 75$ jobs. We set the time limit of each run to 150 seconds.

Figure 2: Solution times achieved by means of `BP-DP` and `BP-CPLEX` while solving the linear relaxation of (1) at the root node of the branch-and-bound tree - Subsets of instances with $n = 75$ jobs

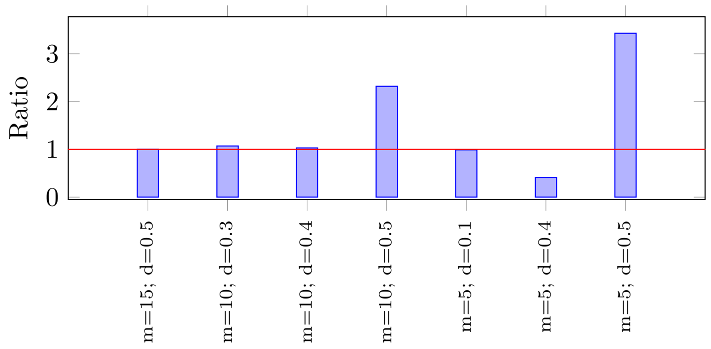


By grouping the instances with same values of m and d, we get 20 groups of 30 instances each. `BP-CPLEX` was able to solve the linear relaxation of model (1) for all the instances in each group, whereas `BP-DP` was not for the groups $m = 5$; $d = 0.2$ and $m = 5$; $d = 0.3$. In particular, `BP-CPLEX` proved that all the instances in these two groups are all infeasible, so as all the instances in groups $m = 10$; $d = 0.5$, $m = 5$; $d = 0.4$, and $m = 5$; $d = 0.5$. All the remaining instances are feasible. In Figure 2, we reported the box-plot concerning the solution times of `BP-DP` and `BP-CPLEX` for all the subsets for which both the algorithms have been able to solve all the instances. Clearly, the solution times for the subsets of infeasible instances $m = 5$; $d = 0.2$ and $m = 5$; $d = 0.3$ are not considered. The results for the subsets of instances whose median of solution times is greater than 0.25 show that `BP-DP` is better than `BP-CPLEX` as far as feasible instances are considered. This is true also for some subsets of infeasible instances, namely $m = 10$; $d = 0.5$ and $m = 5$; $d = 0.5$. In fact, except for the subset of instances $m = 5$; $d = 0.4$, each of the five statistics concerning the solution times of `BP-DP` dominates the corresponding statistic regarding the solution times of `BP-CPLEX`. Nevertheless, `BP-CPLEX` seems to be more stable than `BP-DP` in addressing infeasible instances, being able to solve the linear relaxation of model (1) for all of them. In order to detail further the analysis, for the subsets of instances whose median of solution times is greater than 0.25, we reported in Figure 3-(a) the geometric means of the ratios of solution time taken over the 30 instances of each subset. For a given subset, we compute instance by instance the ratio $Sol.\ time^{\texttt{BP-CPLEX}}/Sol.\ time^{\texttt{BP-DP}}$ and provide the graphical representation of the geometric mean over the 30 instances of the subset. For example, the fourth bar associated with an average ratio $Sol.\ time^{\texttt{BP-CPLEX}}/Sol.\ time^{\texttt{BP-DP}}$ of about 1.5, above 1, means that solution times of `BP-CPLEX` are consistently longer (by a factor 1.5 on average) than those of `BP-DP` for the subset of instances $m = 10$; $d = 0.5$. Then, in Figure 3-(b) we show the geometric means of the ratios of computing time needed by `BP-CPLEX` and `BP-DP` to perform the last column generation iteration, i.e., the column generation iteration consisting in solving the subproblems when the dual variables assume a given set of optimal values, and thus resulting in no new column generated. Again, as already shown by results represented in Figure 2, the use of CPLEX solving directly formulation (4) is to be preferred only for the infeasible instances of subset $m = 5$; $d = 0.4$. Finally, to the sake of completeness, we report that the average number of calls to the algorithm used to optimally solve the subproblems has been greater than 2 only for subsets of infeasible instances $m = 10$; $d = 0.5$, $m = 5$; $d = 0.2$, $m = 5$; $d = 0.3$, and $m = 5$; $d = 0.4$. In particular, for the instances in subsets $m = 10$; $d = 0.5$ and $m = 5$; $d = 0.4$, Table 1 reports for `BP-DP` and `BP-CPLEX` the average number of columns generated by the corresponding algorithm used to optimally solve the subproblems (*Columns*) and the average number of columns generated in total to solve an instance (*Tot. columns*). Numbers are similar and show how the different performance of `BP-DP`

Figure 3: Root node of the branch-and-bound tree - Subsets of instances with $n = 75$ jobs



(a) Geometric means of ratios of solution time *Sol. time*$^{\texttt{BP-CPLEX}}$/*Sol. time*$^{\texttt{BP-DP}}$ concerning the solution of the linear relaxation of (1)



(b) Geometric means of ratios of computing time *Comp. time*$^{\texttt{BP-CPLEX}}$/*Comp. time*$^{\texttt{BP-DP}}$ needed to perform the last column generation iteration to optimally solve the linear relaxation of (1)

and `BP-CPLEX` do not depend on the number of columns generated by the corresponding algorithm used to solve the subproblems to optimality. Difference in performance that we attribute to the effectiveness of CPLEX in solving the subproblems arising for not clearly infeasible instances by means of, among others, preprocessing and probing techniques.

Table 1: Average number of columns generated

	`BP-DP`		`BP-CPLEX`	
Subset	Columns	Tot. columns	Columns	Tot. columns
$m = 10$; $d = 0.5$	108.1	4790.1	74.7	4781.1
$m = 5$; $d = 0.4$	61.5	860.1	41.3	850.8

Hence, considering all the observations made herein-above, we decided to finalize the design of the BP algorithm as follows. When the partial pricing fails (see Section 2.2.3), the subproblems are solved by CPLEX addressing directly formulation (4) only if the current RMP has an infeasible base, otherwise the subproblems are solved by the DP algorithm. We named this finalized version of the BP algorithm as `BP`.

Then, in order to assess the impact of the main components/features of `BP`, we derived different variants from the algorithm by alternatively switching off:

- the possibility, at each column generation iteration, to avoid solving the subproblem corresponding to machine i_t after having addressed the subproblem for machine i_k, $k \neq t$, such that $|\rho_{i_k} - \rho_{i_t}| < \epsilon$ ($\texttt{BP-}\overline{\texttt{SKIP}}$); this implies that, given the set $\mathcal{S}^{i_k}$ of pareto-optimal solutions for the subproblem associated with machine i_k, no check is made to see if the solutions in $\mathcal{S}^{i_k}$ represent negative reduced cost variables (columns) associated with other machines;
- the possibility, at each column generation iteration, to perform partial pricing ($\texttt{BP-}\overline{\texttt{PP}}$);
- the possibility to run the restricted master heuristic ($\texttt{BP-}\overline{\texttt{RMH}}$).

For each variant, including BP, we solved the 600 benchmark instances with $n = 75$ jobs. Again, the time limit (TL) for each run was set to 150 seconds.

Table 2 summarizes the results obtained. For each subset of instances and each of the four algorithms, we report the average time spent while solving (the linear relaxation of model (1) at) the root node of the branch-and-bound tree (*Time (root)*), the number of instances solved to optimality (*Opt*), and the average solution time (*Time*). Additionally, we report for BP the average number of nodes explored (*Nodes*). Times are given in seconds. A star (*) appears next to the name of the subsets including infeasible instances.

According to the *Average*/Total results reported in Table 2, we can see how, by avoiding to solve all the subproblems associated with machines $i_r \in \mathcal{M}$ sharing an almost identical value of ρ_{i_r}, BP is on average at least 5 times faster than BP-$\overline{\text{SKIP}}$ in solving the root node of the branch-and-bound tree. In particular, this design feature is more relevant than the possibility to perform partial pricing. Actually, BP is on average 'only' 2 times faster than BP-$\overline{\text{PP}}$ in solving the root node.

Looking at the average times to solve the root node per subset, only for subsets $m = 5$; $d = 0.2$ and $m = 5$; $d = 0.3$, including infeasible instances, BP does not dominate both BP-$\overline{\text{SKIP}}$ and BP-$\overline{\text{PP}}$. As for the remaining subsets, the ratio between the average times is in favour of BP, especially as far as feasible instances are considered. For subsets $m = 15$; $d = 0.1$ and $m = 10$; $d = 0.1$, BP is on average 2 orders of magnitude faster than both BP-$\overline{\text{SKIP}}$ and BP-$\overline{\text{PP}}$. For all the subsets with $m = 20$ and $m = 15$, with the exception of $m = 15$; $d = 0.5$, BP is on average 1 order of magnitude faster than BP-$\overline{\text{SKIP}}$. These different speeds of convergence of the column generation algorithm have relevant impacts on the whole solution process, allowing BP to solve about 30% and 12% more instances to optimality than BP-$\overline{\text{SKIP}}$ and BP-$\overline{\text{PP}}$, respectively. Also, always regarding the number of instances solved to optimality, and surprisingly enough, BP-$\overline{\text{RMH}}$ is able to solve 485 instances against the 484 solved by BP. However, the average solution time of BP is lower than the average solution time BP-$\overline{\text{RMH}}$ for all the subsets but $m = 10$; $d = 0.3$. In particular, the time reduction occurs even if BP solves less instances in the subset (see, i.e., $m = 10$; $d = 0.2$) and is around 30% for subsets $m = 15$; $d = 0.2$, $m = 15$; $d = 0.3$, and all those associated with $m = 20$. Thus, when BP-$\overline{\text{RMH}}$ is able to solve an instance to optimality and BP is not, this substantially depends on the different branch-and-bound trees explored by the two algorithms. Finally, let us consider the results reported for the subset of feasible instances solved by BP, focusing in particular on the average time spent to solve the root node and the number of nodes explored. For a given subset, when *Time (root)* is greater than 0.15, the average time spent per explored node (150/*Nodes*) is much less than *Time (root)*. Actually, the average time spent per explored node is less than 0.3 for $m = 15$; $d = 0.5$, $m = 10$; $d = 0.3$, and $m = 10$; $d = 0.4$, and is equal to 1 for $m = 5$; $d = 0.1$ (0.7 considering only the instances in the subset not solved to optimality). Thus, the negative impacts of the so-called tailing-off effect of the column generation (see Lübbecke and Desrosiers (2005)), if any, are very limited and ad-hoc stabilization techniques are not required.

As a consequence of all the analysis outlined in this section, we performed all following experiments by using BP as final version of the BP algorithm.

3.1.2. Comparison with state-of-the-art exact algorithms

As mentioned in the Section 1, the algorithm presented in Kowalczyk and Leus (2017) is the best known exact solution algorithm proposed so far in the literature for PMC. Hereafter, we will refer to this algorithm as KL2017. We compared BP against KL2017 on the full set of PMC benchmark instances. KL2017 was implemented in C, compiled by means of gcc (version 4.8.2) with full optimization pack -O3, and run on one core of a system with an Intel Core i7-3770 processor, clocked at 3.4 GHz, 8 GB of RAM, and Linux as OS. The MIP solver Gurobi 6.0.0 was used within the algorithm as linear and integer solver. In Kowalczyk and Leus (2017), the authors set a time limit of 900 seconds for KL2017. However, considering the different computational setting, to provide a fair comparison we set a time limit for BP of 720 seconds. This value has been computed considering the difference between the CPUs as reported in PassMark (2021). The results obtained are summarized in Tables 3 and 4, having structures similar to the tables given in Kowalczyk and Leus (2017). Complete detailed results for all instances are publicly available at https://github.com/Treema81/PMC_Data.

For each range of processing times, and each combination of number of machines (m) and jobs (n), we report in Table 3 the average solution time (*Time*) in seconds, the average number of branch-and-bound nodes explored (*Nodes*), the number of instances proved to be infeasible (*Infeasible*), and the number of feasible instances solved to the optimality (*Optimal*). Then, for the feasible instances for which valid lower and upper bounds have been found, we report the average percentage optimality gap (*Gap (%)*) and, in brackets, the number of instances over which the average gap is computed. Note that, for a given range of processing times, each row of the table reports the aggregate results for the 50 instances associated with the

Table 2: Results assessing the impact of the main components of BP - Subsets of instances with $n = 75$ jobs

	BP-$\overline{\text{SKIP}}$			BP-$\overline{\text{PP}}$			BP-$\overline{\text{RMH}}$			BP			
Subset	Time (root)	Opt	Time	Time (root)	Opt	Time	Time (root)	Opt	Time	Time (root)	Opt	Time	Nodes
$m = 20$; $d = 0.1$	8.34	29	48.3	0.98	24	77.3	0.07	30	13.0	0.07	30	8.4	101.0
$m = 20$; $d = 0.2$	8.93	27	49.7	0.46	30	32.4	0.08	30	15.1	0.08	30	10.1	98.4
$m = 20$; $d = 0.3$	7.52	27	51.2	0.70	30	22.5	0.09	30	18.4	0.09	30	11.6	104.0
$m = 20$; $d = 0.4$	8.49	21	79.6	0.86	30	35.3	0.11	30	30.0	0.11	30	20.5	143.4
$m = 20$; $d = 0.5$	9.97	16	91.3	1.00	25	70.9	0.12	26	56.7	0.12	27	41.8	274.4
$m = 15$; $d = 0.1$	6.79	30	36.9	13.67	15	110.1	0.06	30	10.3	0.06	30	9.7	123.0
$m = 15$; $d = 0.2$	6.85	23	73.0	0.50	30	48.1	0.08	30	22.4	0.08	30	15.9	136.6
$m = 15$; $d = 0.3$	6.55	19	86.8	0.76	29	42.4	0.09	28	39.5	0.09	30	27.3	181.0
$m = 15$; $d = 0.4$	8.17	11	106.8	0.93	15	103.6	0.15	17	90.6	0.15	16	88.6	471.0
$m = 15$; $d = 0.5$	10.21	7	137.4	1.28	5	137.9	1.19	9	121.5	1.19	10	119.5	931.6
$m = 10$; $d = 0.1$	5.97	28	64.1	10.49	21	101.8	0.06	30	14.2	0.06	30	13.6	146.7
$m = 10$; $d = 0.2$	5.57	14	102.9	0.90	25	65.6	0.07	25	66.9	0.07	22	62.7	292.1
$m = 10$; $d = 0.3$	5.78	4	139.5	1.75	5	139.1	1.62	7	128.5	1.62	7	129.6	644.9
$m = 10$; $d = 0.4$	12.77	0	TL	8.03	0	TL	2.28	0	TL	2.28	0	TL	590.8
$m = 10$; $d = 0.5$*	25.92	30	25.9	17.33	30	17.3	4.71	30	4.7	4.71	30	4.7	1.0
$m = 5$; $d = 0.1$	4.40	4	140.0	81.53	0	TL	3.83	13	110.9	3.83	12	109.7	155.5
$m = 5$; $d = 0.2$*	TL	0	TL	29.17	30	29.2	53.41	30	53.4	53.41	30	53.4	1.0
$m = 5$; $d = 0.3$*	132.32	20	132.5	13.30	30	13.3	17.61	30	17.6	17.61	30	17.6	1.0
$m = 5$; $d = 0.4$*	30.09	30	30.1	6.12	30	6.1	4.39	30	4.4	4.39	30	4.4	1.0
$m = 5$; $d = 0.5$*	7.50	30	7.5	3.28	30	3.3	1.01	30	1.0	1.01	30	1.0	1.0
Average/Total	*23.11*	370	*85.2*	*9.65*	434	*67.8*	*4.55*	485	*48.6*	*4.55*	484	*45.0*	*220.0*

same values of m and n, independently of the value of the density d. Table 4 reports the same information for each value of the density probability d and each combination of number of machines (m) and jobs (n). Each row of the table reports the aggregate results for the 30 instances associated with the same values of n and m, regardless of the of processing time values.

BP optimally solves 2296 out of the 2550 instances. In particular, BP proves 471 instances to be infeasible and achieves the optimal solution for the other 1825 instances. Considering the remaining 254 instances not solved to the optimality, BP computes a non-optimal feasible solution for 170 further instances, whereas it is neither able to find any solution nor to prove the infeasibility within the imposed time limit (TL) for 84 instances.

Looking at the tables in more details, especially at the average solution times and the number of instances solved, we can observe that the efficiency of BP clearly declines with the increase in the number of jobs n. In fact, all instances with 10 and 25 jobs, and almost all with 50 jobs, have been solved to optimality, while in only 87.5% of the instances with $n = 75$ and 71.33% of the instances with 100 jobs BP is able to achieve the optimal solution.

Moreover, Table 3 shows that the number of instances solved to optimality decrease steadily with the increase in *range* from 802 with *range* = [1, 10] to 737 with *range* = [1, 100]. The average computational time grows accordingly, instances with *range* = [1, 100] take more than twice the computational time of the instances with *range* = [1, 10], 131.3 seconds vs 59.1. In all ranges there are three classes of instances that prove to be the most difficult to solve for BP ($n = 50$, $m = 5$), ($n = 75$, $m = 10$) and ($n = 100$, $m = 15$). These instances takes always the longest computational time to be solved among all instances with the same *range* and number of jobs n. This is in line with the results reported in Kowalczyk and Leus (2017) and show that both BP and KL2017 encounter convergence difficulties on the same classes of instances.

Additionally, according to Table 4, increasing the number of conflicts makes, on average, the problem more difficult. In fact, the average computational time rises from 70.7 seconds with $d = 0.1$ to 108.6 seconds with $d = 0.5$. However, this growth is not monotone as for the *range*. Looking more closely, we can see that the computational time increases with d for feasible instances, whereas it decreases at the increase of the number of conflicts as far as infeasible instances are concerned. In more details, we observed three different behaviours. Let us consider highly infeasible instances as the ones with $m = 5$ and $n = 100$. These instances, as reported in Table 4, are the most difficult instances with $d = 0.1$ and are all already infeasible with $d = 0.2$. Increasing the number of conflicts cannot make them any more feasible but makes easier for BP to prove their infeasibility. In fact, the average computational time to prove them infeasible decreases monotonically from 188.8 seconds with $d = 0.2$ to only 4.9 seconds with $d = 0.5$. Now, consider feasible instances like the ones with with $m = 15$ and $n = 75$. They are always feasible with any value of d and the required computational time increases from an average of 10.3 seconds with $d = 0.1$ to 470.4 seconds with $d = 0.5$. Finally, let us take into account a more general case as the ones represented by instances with $m = 10$ and $n = 75$. They show a quasi-parabolic trend. The computational time increases as long as they are feasible reaching a peak with $d = 0.4$. Then the computational time drops to less than 5 seconds with $d = 0.5$ when all instances become infeasible. The vertex of this hypothetical parabola, between $d = 0.4$ and $d = 0.5$ in this case, represents the most difficult instances and corresponds to instances in which the *chromatic number* of the underlying conflict graph is very close to m. These results follow the ones reported in Kowalczyk and Leus (2017) and show that both `BP` and `KL2017` have very similar behaviour.

The performance of `BP` are assessed against those of `KL2017` in the following. In a private communication, Kowalczyk and Leus (2017) sent us the detailed results for the 2550 benchmark instances. According to their results, we have been able to see that 471 instances have been found infeasible and that the number of feasible instances ranges between 2076 and 2079. Overall `BP` solved to optimality 2297 instances, while `KL2017` is claimed to reach the optimal solution for 2436 instances. In details, both algorithms proved the same 471 instances to be infeasible and, as for feasible instances, `BP` and `KL2017` computed the optimal solution in 1826 and 1965 cases, respectively. However, comparing the results instance by instance, we noticed that for 87 feasible instances, claimed to be solved to optimality by `KL2017`, we are able to achieve a better solution. In other words, the authors of Kowalczyk and Leus (2017) assert to have found 87 optimal solutions that are not actually optimal. The differences, in terms of value of the objective function, with the results obtained by `BP` are not negligible and range between 1 to 8. This means that `KL2017` is able to solve *at most* 52 more instances, out of 2079, than `BP`. Hence, the number of instances solved by both algorithms is comparable. Moreover, over the 170 feasible instances for which valid lower and upper bounds have been computed, the absolute difference between the bounds is equal to 1 in 47 cases, and to 1 or 2 in 64 cases. This shows how, even without making use of ad-hoc primal heuristics and feasibility check algorithms, `BP` is competitive with `KL2017`.

Table 3: Computational results on instances with conflicts in function of range

		Solution process		Instances			
				Solved		Feasible with LB/UB values	
m	n	Time	Nodes	Infeasible	Optimal	Gap (%)	
Range $(a) = [1, 10]$							
5	10	0.0	4.7	0	50	-	
20	25	0.1	11.3	0	50	-	
15	25	0.2	11.4	0	50	-	
10	25	0.2	9.4	0	50	-	
5	25	0.3	14.1	16	34	-	
20	50	1.5	22.7	0	50	-	
15	50	1.4	34.3	0	50	-	
10	50	4.2	102.6	0	50	-	
5	50	36.3	97.4	31	19	-	
20	75	4.8	67.5	0	50	-	
15	75	18.0	185.5	0	50	-	
10	75	185.4	859.0	10	29	15.9	(7)
5	75	27.2	25.6	40	10	-	
20	100	50.7	249.9	0	49	3.7	(1)
15	100	265.6	1144.1	0	35	10.2	(8)
10	100	212.2	432.4	20	19	26.4	(4)
5	100	196.9	8.9	40	0	-	
Average/Total		*59.1*	*193.0*	157	645	*15.1*	(20)
Range $(b) = [1, 50]$							
5	10	0.1	5.1	0	50	-	
20	25	0.2	11.7	0	50	-	
15	25	0.2	13.5	0	50	-	
10	25	0.4	13.5	0	50	-	
5	25	0.6	23.2	16	34	-	
20	50	2.5	37.4	0	50	-	
15	50	3.3	60.4	0	50	-	
10	50	12.5	188.7	0	50	-	
5	50	117.4	216.6	31	15	2.3	(4)
20	75	16.5	138.6	0	50	-	
15	75	167.8	938.6	0	42	1.7	(8)
10	75	299.5	1114.7	10	23	16.3	(14)
5	75	74.0	69.9	40	9	0.2	(1)
20	100	245.9	817.2	0	39	1.2	(11)
15	100	399.8	1555.3	0	26	8.1	(18)
10	100	316.3	673.1	20	12	6.7	(10)
5	100	202.5	6.9	40	0	-	
Average/Total		*109.4*	*346.1*	157	600	*7.2*	(66)
Range $(c) = [1, 100]$							
5	10	0.1	5.4	0	50	-	
20	25	0.2	12.5	0	50	-	
15	25	0.3	13.7	0	50	-	
10	25	0.7	17.1	0	50	-	
5	25	0.7	27.3	16	34	-	
20	50	7.2	43.1	0	50	-	
15	50	6.1	86.6	0	50	-	
10	50	21.1	272.8	0	50	-	
5	50	107.8	235.7	31	16	1.9	(3)
20	75	42.0	245.2	0	50	-	
15	75	253.1	1313.6	0	38	1.5	(12)
10	75	360.2	1439.5	10	18	10.1	(19)
5	75	124.3	90.8	40	6	0.4	(4)
20	100	309.8	1129.4	0	36	1.7	(14)
15	100	462.2	1821.3	0	22	5.8	(20)
10	100	333.2	722.1	20	10	3.3	(12)
5	100	203.0	5.8	40	0	-	
Average/Total		*131.3*	*440.1*	157	580	*4.7*	(84)

Table 4: Computational results on instances with conflicts in function of density

				Instances			
		Solution process		Solved		Feasible with LB/UB values	
m	n	Time	Nodes	Infeasible	Optimal	Gap (%)	
Density $d = 0.1$							
5	10	0.1	6.9	0	30	-	
20	25	0.2	14.6	0	30	-	
15	25	0.3	15.5	0	30	-	
10	25	0.6	18.7	0	30	-	
5	25	0.7	36.2	0	30	-	
20	50	10.0	46.3	0	30	-	
15	50	3.2	55.2	0	30	-	
10	50	3.6	79.5	0	30	-	
5	50	8.5	92.0	0	30	-	
20	75	9.0	100.1	0	30	-	
15	75	10.3	122.7	0	30	-	
10	75	14.7	146.9	0	30	-	
5	75	298.6	306.5	0	25	0.4	(5)
20	100	44.9	158.8	0	29	0.7	(1)
15	100	25.1	179.2	0	30	-	
10	100	52.4	264.3	0	30	-	
5	100	TL	31.9	0	0	-	
Average/Total		*70.7*	*98.5*	0	474	*0.4*	(6)
Density $d = 0.2$							
5	10	0.1	5.5	0	30	-	
20	25	0.2	12.9	0	30	-	
15	25	0.2	14.6	0	30	-	
10	25	0.5	14.7	0	30	-	
5	25	0.8	34.3	0	30	-	
20	50	2.3	38.9	0	30	-	
15	50	3.2	52.2	0	30	-	
10	50	5.7	87.1	0	30	-	
5	50	422.9	821.1	3	20	2.1	(7)
20	75	10.9	98.2	0	30	-	
15	75	17.1	136.5	0	30	-	
10	75	133.3	483.8	0	28	0.2	(2)
5	75	54.7	1.0	30	0	-	
20	100	40.9	182.5	0	30	-	
15	100	86.6	330.9	0	30	-	
10	100	529.7	1466.9	0	11	0.8	(19)
5	100	188.8	1.0	30	0	-	
Average/Total		*88.1*	*222.5*	63	419	*1.1*	(28)
Density $d = 0.3$							
5	10	0.0	4.9	0	30	-	
20	25	0.2	11.9	0	30	-	
15	25	0.2	12.8	0	30	-	
10	25	0.4	12.2	0	30	-	
5	25	0.8	32.7	0	30	-	
20	50	2.2	35.0	0	30	-	
15	50	3.3	55.9	0	30	-	
10	50	10.8	123.0	0	30	-	
5	50	3.2	1.0	30	0	-	
20	75	13.0	102.4	0	30	-	
15	75	28.8	180.3	0	30	-	
10	75	535.6	2131.1	0	12	0.9	(18)
5	75	17.3	1.0	30	0	-	
20	100	101.6	316.5	0	30	-	
15	100	385.4	1292.8	0	18	0.9	(12)
10	100	TL	1312.8	0	0	28.2	(7)
5	100	66.4	1.0	30	0	-	
Average/Total		*111.1*	*331.0*	90	360	*6.1*	(37)

Density $d = 0.4$							
5	10	0.0	4.2	0	30	-	
20	25	0.1	10.7	0	30	-	
15	25	0.2	12.1	0	30	-	
10	25	0.4	10.7	0	30	-	
5	25	0.1	3.5	18	12	-	
20	50	2.0	27.8	0	30	-	
15	50	3.9	63.0	0	30	-	
10	50	23.8	291.8	0	30	-	
5	50	0.9	1.0	30	0	-	
20	75	21.7	143.3	0	30	-	
15	75	204.9	878.5	0	28	0.4	(2)
10	75	TL	2925.7	0	0	25.7	(20)
5	75	4.3	1.0	30	0	-	
20	100	284.3	823.6	0	26	0.6	(4)
15	100	662.2	2977.8	0	5	4.5	(25)
10	100	104.5	1.0	30	0	-	
5	100	23.8	1.0	30	0	-	
Average/Total		*121.0*	*481.0*	138	311	*12.3*	(51)
Density $d = 0.5$							
5	10	0.0	3.8	0	30	-	
20	25	0.1	9.2	0	30	-	
15	25	0.2	9.4	0	30	-	
10	25	0.3	10.4	0	30	-	
5	25	0.1	1.0	30	0	-	
20	50	2.1	24.1	0	30	-	
15	50	4.4	75.9	0	30	-	
10	50	19.0	358.9	0	30	-	
5	50	0.2	1.0	30	0	-	
20	75	50.8	308.1	0	30	-	
15	75	470.4	2744.9	0	12	1.7	(18)
10	75	4.9	1.0	30	0	-	
5	75	1.0	1.0	30	0	-	
20	100	538.9	2179.5	0	9	1.8	(21)
15	100	TL	2753.9	0	0	24.4	(9)
10	100	29.5	1.0	30	0	-	
5	100	4.9	1.0	30	0	-	
Average/Total		*108.6*	*499.1*	180	261	*6.0*	(48)

3.2. Instances for $P||C_{max}$

As PMC generalizes $P||C_{max}$, which has been extensively studied in the literature, we decided to compare `BP` against state-of-the-art exact algorithms for the latter problem, such as the `DIMM` algorithm by (Dell'Amico *et al.*, 2008), the branch-and-bound algorithms by Haouari and Jemmali (2008) and Lawrinenko (2017), or the exact approach by Mrad and Souayah (2018) consisting of solving an an-arc flow model for the problem by means of a commercial ILP solver.

As expected, preliminary experiments showed that, in general, `BP` cannot compete with state-of-the-art exact algorithms for $P||C_{max}$. However, `BP` was quite effective in solving instances for which the average number of jobs per machine (n/m) is relatively small. A similar behaviour characterizes other branch-and-price algorithms proposed in the literature to address scheduling problems (see, i.e., Kowalczyk and Leus (2018)). This kind of instances have been introduced by Haouari and Jemmali (2008) and classified by the authors as hard instances. Except for small-sized hard instances ($n \leq 20$), the performance of their algorithm was deteriorating for increasing values of $n \leq 200$ (Haouari and Jemmali, 2008, p. 33). These benchmark instances have been addressed later in Lawrinenko (2017) and Mrad and Souayah (2018). In particular, the exact approach proposed in the latter paper has been able to solve all the hard instances with an average running time smaller than 8 seconds. Thus, to further assess the performance of their algorithm, the authors introduced an additional set of hard instances. Such set includes instances with ratios $\frac{n}{m} \in \{2.0, 2.25, 2.5, 2.75, 3.0\}$. For each ratio, seven different classes are considered for the random generation of the processing times:

- Class 1: Discrete uniform distribution in [1, 100]
- Class 2: Discrete uniform distribution in [20, 100]
- Class 3: Discrete uniform distribution in [50, 100]

- Class 4: Normal distribution with $\mu = 100$ and $\sigma = 20$
- Class 5: Normal distribution with $\mu = 100$ and $\sigma = 50$
- Class 6: Discrete uniform distribution in $[n, 4n]$
- Class 7: Normal distribution with $\mu = 4n$ and $\sigma = n$

Each class considers then pairs (n, m) such that:

- $n \in \{20, 40, 60, 80, 100, 120, 140, 160, 180, 200\}$ if $n = m \in \{2, 2.5\}$
- $n \in \{36, 54, 72, 90, 108, 126, 144, 162, 180, 198\}$ if $n = m \in \{2.25, 3\}$
- $n \in \{22, 44, 66, 88, 110, 132, 154, 176, 198, 220\}$ if $n = m \in \{2.75\}$

10 instances are generated for each pair (n, m), for a total of 3500 instances.

We decided to thoroughly test BP over the set of benchmark instances introduced by Mrad and Souayah (2018), and that the authors kindly provided us. (All but the 70 instances for $n = 220$, $m = 80$, and $n/m = 2.75$. We replaced them with the 70 instances associated with $n = 187$, $m = 68$, and $n/m = 2.75$, that were included in the set the authors provided us.) In the remainder, we will refer to the exact approach designed by Mrad and Souayah (2018) as to MS2018.

MS2018 was implemented in C++ (by making use of CPLEX 12.6 Concert Technology) and run on an Intel Core i7-2600 (3.4 GHz) machine, with 16 GB RAM and Windows 7 Professional (64-bit) as OS. In Mrad and Souayah (2018) the authors set a time limit of 1200 seconds for MS2018. However their setting, in terms of computing power, is different from ours. For this reason, in order to provide a fair comparison, considering the performance difference between the two processors as reported in the benchmark by PassMark (2021), we set a time limit for BP of 850 seconds. Moreover, we initialized BP with a primal solution computed by means of the same heuristic as that used in Mrad and Souayah (2018). Instance-wise detailed results are available upon request. Table 5 shows the results computed by BP, grouped by ratio $\frac{n}{m}$ and by class of distribution. Each row of the table reports, for each class of distribution, the average execution time (*Time*) and the number of instances not solved to the optimality (*NS*) w.r.t. the 10 instances associated with the considered values of n and m.

BP is able to find the optimal solution in 3358 instances over 3500 with an average execution time of 97.51 seconds. In 111 of the 142 instances not solved to optimality the average residual gap is less than 1%. Instances not solved to optimality are not spread equally among ratios $\frac{n}{m}$ and classes of distributions. The higher are the ratio and the class identifier, the more difficult is to solve the instances to optimality. In particular, almost 80% of the instances not solved belongs to classes 6 and 7 with ratio ≥ 2.5.

In order to compare the performance of BP and MS2018we only considered the 3430 commonly addressed by both algorithms. Table 6 reports, grouped by ratio $\frac{n}{m}$ and by class of distribution, the number of instances not solved to optimality by BP and MS2018. The overall performance of the two algorithms are similar. However, instances not solved by BP are more spread among different ratios and classes while all instances not solved by MS2018 belong to classes 6 and 7 with ratio $\geq$ 2.25. Then, MS2018 seems to be slightly more effective than BP while addressing instances in classes 1-6, whereas BP clearly outperforms MS2018 in solving instances of Class 7. Overall, BP solves 1 more instance than MS2018. These results demonstrate that, although BP was not specifically designed for $P||C_{max}$, it is in line with state-of-the-art exact methods for this problem while addressing instances with a relatively small average number of jobs per machine.

Table 5: Computational results on $P||C_{max}$ instances

		Class 1		Class 2		Class 3		Class 4		Class 5		Class 6		Class 7	
n	m	Time	NS	Time	NS	Time	NS	Time	NS	Time	NS	Time	NS	Time	NS
Ratio $n/m = 2.00$															
20	10	0.1	0	0.1	0	0.1	0	0.1	0	0.1	0	0.1	0	0.1	0
40	20	0.1	0	0.1	0	0.1	0	0.1	0	0.1	0	0.1	0	0.2	0
60	30	0.3	0	0.3	0	0.7	0	0.6	0	1.0	0	0.7	0	0.9	0
80	40	0.5	0	1.3	0	1.1	0	2.2	0	0.3	0	1.7	0	2.0	0
100	50	2.7	0	3.7	0	2.2	0	4.5	0	1.6	0	1.6	0	4.6	0
120	60	7.9	0	8.3	0	2.5	0	5.3	0	1.2	0	6.3	0	7.7	0
140	70	12.2	0	12.8	0	4.2	0	9.7	0	1.7	0	16.0	0	11.5	0
160	80	11.2	0	13.3	0	7.4	0	24.9	0	1.7	0	11.9	0	12.4	0
180	90	19.0	0	13.8	0	24.3	0	43.8	0	1.2	0	23.5	0	40.2	0
200	100	20.8	0	24.0	0	7.8	0	45.7	0	0.0	0	26.6	0	39.3	0
Average/Total		*7.5*	0	*7.8*	0	*5*	0	*13.7*	0	*0.9*	0	*8.8*	0	*11.9*	0
Ratio $n/m = 2.25$															
20	8	0.1	0	0.5	0	0.8	0	0.8	0	0.4	0	0.7	0	0.8	0
40	16	0.7	0	1.7	0	2.7	0	2.3	0	1.1	0	2	0	2.6	0
60	24	2.0	0	3.6	0	5.7	0	5.3	0	3.1	0	4.8	0	6.5	0
80	32	3.6	0	8.9	0	10.9	0	9.7	0	8.5	0	11.3	0	15.7	0
100	40	10.5	0	99.9	1	21.0	0	21.4	0	10.2	0	23.9	0	29.5	0
120	48	17.2	0	25.8	0	41.5	0	38.9	0	106.8	1	43.6	0	129.8	1
140	56	27.1	0	123.9	1	63.0	0	58.6	0	34.3	0	152.8	1	98.9	0
160	64	45.7	0	68.5	0	97.5	0	87.5	0	34.6	0	214.3	1	155.9	0
180	72	39.9	0	164.5	1	134.9	0	120.7	0	57.0	0	394.6	3	319.5	1
200	80	70.3	0	113.2	0	198.1	0	186.9	0	107.5	0	421.9	2	371.5	0
Average/Total		*21.7*	0	*61.0*	3	*57.6*	0	*53.2*	0	*36.4*	1	*127.0*	7	*113.1*	2
Ratio $n/m = 2.50$															
20	8	0.1	0	0.2	0	0.3	0	0.3	0	0.2	0	0.2	0	0.2	0
40	16	0.5	0	1.1	0	1.3	0	1.5	0	1.2	0	1.5	0	1.8	0
60	24	2.3	0	3.6	0	4.7	0	4.2	0	3.8	0	4.7	0	5.4	0
80	32	4.6	0	8.1	0	11.0	0	9.8	0	10.1	0	12.8	0	15.3	0
100	40	13.3	0	16.5	0	23.6	0	23.9	0	19.7	0	29.3	0	64.8	0
120	48	107.0	1	34.2	0	51.8	0	49.0	0	34.8	0	72.9	0	130.6	0
140	56	122.1	1	59.0	0	89.4	0	85.4	0	72.3	0	203.0	1	256.1	0
160	64	135.3	1	89.9	0	148.5	0	142.4	0	92.6	0	476.9	3	504.2	2
180	72	87.7	0	129.7	0	237.2	0	220.5	0	158.6	0	690.7	7	806.4	7
200	80	103.1	0	258.0	1	359.9	0	314.1	0	220.1	1	765.8	7	838.3	9
Average/Total		*57.6*	3	*60.0*	1	*92.8*	0	*85.1*	0	*61.3*	1	*225.8*	18	*262.3*	18
Ratio $n/m = 2.75$															
22	8	0.3	0	0.4	0	0.3	0	0.3	0	0.3	0	0.3	0	0.3	0
44	16	1.4	0	1.8	0	1.8	0	2.1	0	1.8	0	2.0	0	2.5	0
66	24	5.1	0	5.8	0	5.9	0	8.3	0	6.7	0	7.7	0	10.8	0
88	32	12.8	0	12.1	0	15.3	0	21.5	0	29.2	0	108.5	1	39.2	0
110	40	106.5	1	106.6	1	41.5	0	52.8	0	31.9	0	65.2	0	110.5	0
132	48	37.7	0	56.2	0	86.8	0	103.5	0	58.6	0	286.5	1	391.0	1
154	56	67.2	0	74.9	0	169.1	0	160.6	0	240.6	2	404.3	2	645.0	3
176	64	161.3	1	133.0	0	252.0	0	239.4	0	112.8	0	564.1	1	845.5	9
187	68	99.1	0	123.1	0	364.8	0	320.9	0	208.3	1	770.3	6	TL	10
198	72	187.4	1	196.2	0	384.6	0	372.8	0	322.5	2	791.7	6	TL	10
Average/Total		*67.9*	3	*71.0*	1	*132.2*	0	*128.2*	0	*101.3*	5	*300.1*	17	*374.5*	33
Ratio $n/m = 3.00$															
36	12	0.9	0	1.1	0	0.8	0	0.8	0	1.2	0	1.2	0	1.1	0
54	18	3.4	0	3.1	0	1.8	0	2.1	0	4.0	0	3.9	0	4.3	0
72	24	6.7	0	7.4	0	5.1	0	6.6	0	9.6	0	10.0	0	17.1	0
90	30	13.4	0	12.1	0	11.5	0	15.6	0	16.3	0	27.5	0	33.4	0
108	36	21.8	0	21.4	0	16.7	0	22.9	0	27.8	0	53.0	0	191.2	1
126	42	113.8	0	45.8	0	33.9	0	42.3	0	52.3	0	100.8	0	140.3	0
144	48	127.7	1	139.4	1	49.9	0	49.6	0	241.9	2	282.7	1	252.8	0
162	54	65.9	0	79.2	0	68.8	0	104.9	0	348.8	3	428.7	2	576.9	0
180	60	171.9	1	193.3	1	94.2	0	119.4	0	134.9	0	660.6	5	839.0	7
198	66	104.1	0	217.3	1	212.8	1	131.2	0	169.2	0	844.5	9	TL	10
Average/Total		*63.0*	2	*72.0*	3	*49.5*	1	*49.5*	0	*100.6*	5	*241.3*	17	*290.6*	18

Table 6: Comparison between `BP` and `MS2018` - Benchmark instance by Mrad and Souayah (2018)

	Instances not solved by											
	`BP`						`MS2018`					
	For n/m equals to						For n/m equals to					
Class	2.00	2.25	2.50	2.75	3.00	Total	2.00	2.25	2.50	2.75	3.00	Total
1	0	0	3	3	2	8	0	0	0	0	0	0
2	0	3	1	1	3	8	0	0	0	0	0	0
3	0	0	0	0	1	1	0	0	0	0	0	0
4	0	0	0	0	0	1	0	0	0	0	0	0
5	0	1	1	4	5	11	0	0	0	0	0	0
6	0	7	18	11	17	53	0	2	17	14	13	46
7	0	2	18	23	18	61	0	20	26	27	24	97
Total	0	13	41	42	46	142	0	22	43	41	37	143

4. Conclusions

In this work, we presented the first stand-alone branch-and-price (BP) algorithm for solving directly the parallel machine scheduling problem with conflicts (PMC). The problem is NP-hard as it generalizes, among others, the $P||C_{max}$ scheduling problem. Differently from state-of-the-art algorithms addressing PMC and the $P||C_{max}$ scheduling problem, the BP algorithm does not make use of ad-hoc primal heuristics and feasibility check algorithms, it incorporates just an optional, straightforward, restricted master heuristic. Dispite of this, by means of comprehensive computational experiments with benchmark instances, we prove that the new BP algorithm is competitive with the best exact solution algorithm proposed so far in the literature for PMC. Moreover, we additionally show that the BP algorithm is in line with state-of-the-art exact methods for $P||C_{max}$ while addressing instances with a relatively small average number of jobs per machine. The main component of the BP algorithm that contributed to the overall effectiveness of the approach is the branching scheme we propose, that does not introduce symmetries in the solution space and allows the column generation subproblems to share the same feasible region at every node of the branch-and-bound tree. In turn, this allows to design the column generation iteration to potentially avoid solving some of the subproblems.

Acknowledgement. This research was funded by Regione Lombardia, grant agreement n. E97F17000000009, Project AD-COM.

References

Barnhart, C., Johnson, E. L., Nemhauser, G. L., Savelsbergh, M. W. P., and Vance, P. H. (1998). Branch-and-price: column generation for solving huge integer programs. *Operations Research*, **46**, 316–329.

Berghman, L., Leus, R., and Spieksma, F. C. R. (2014). Optimal solutions for a dock assignment problem with trailer transportation. *Annals of Operations Research*, **213**(1), 3–25.

Bodlaender, H. L., Jansen, K., and Woeginger, G. J. (1994). Scheduling with incompatible jobs. *Discrete Applied Mathematics*, **55**(3), 219 – 232.

Dell'Amico, M. and Martello, S. (1995). Optimal scheduling of tasks on identical parallel processors. *ORSA Journal on Computing*, **7**(2), 191–200.

Dell'Amico, M., Iori, M., Martello, S., and Monaci, M. (2008). Heuristic and exact algorithms for the identical parallel machine scheduling problem. *INFORMS Journal on Computing*, **20**(3), 333–344.

Desaulniers, G., Desrosiers, J., and Solomon, M., editors (2005). *Column Generation*. Springer, New York.

Even, G., Halldórsson, M. M., Kaplan, L., and Ron, D. (2009). Scheduling with conflicts: online and offline algorithms. *Journal of Scheduling*, **12**(2), 199–224.

Garey, M. R. and Johnson, D. S. (1979). *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., USA.

Graham, R., Lawler, E., Lenstra, J., and Kan, A. (1979). Optimization and approximation in deterministic sequencing and scheduling: a survey. In P. Hammer, E. Johnson, and B. Korte, editors, *Discrete Optimization II*, volume 5 of *Annals of Discrete Mathematics*, pages 287 – 326. Elsevier.

Haouari, M. and Jemmali, M. (2008). Tight bounds for the identical parallel machine-scheduling problem: Part ii. *International Transactions in Operational Research*, **15**(1), 19–34.

Hong, H.-C. and Lin, B. M. (2018). Parallel dedicated machine scheduling with conflict graphs. *Computers & Industrial Engineering*, **124**, 316 – 321.

Irnich, S. and Desaulniers, G. (2005). *Shortest Path Problems with Resource Constraints*, pages 33–65. Springer US, Boston, MA.

Joncour, C., Michel, S., Sadykov, R., Sverdlov, D., and Vanderbeck, F. (2010). Column generation based primal heuristics. *Electronic Notes in Discrete Mathematics*, **36**, 695–702.

Kowalczyk, D. and Leus, R. (2015). An exact algorithm for parallel machine scheduling with conflicts. Technical Report KBI_1505, Department of Decision Sciences and Information Management, FEB, KU Leuven.

Kowalczyk, D. and Leus, R. (2017). An exact algorithm for parallel machine scheduling with conflicts. *Journal of Scheduling*, **20**(4), 355–372.

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

Lawrinenko, A. (2017). *Identical parallel machine scheduling problems: structural patterns, bounding techniques and solution procedures*. Ph.D. thesis, Friedrich-Schiller-Universität Jena, Jena, Germany. Chapter 3: Effective solution space limitation for the identical parallel machine scheduling problem.

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

Mallek, A., Bendraouche, M., and Boudhar, M. (2019). Scheduling identical jobs on uniform machines with a conflict graph. *Computers & Operations Research*, **111**, 357 – 366.

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

Page, D. R. and Solis-Oba, R. (2018). Makespan minimization on unrelated parallel machines with a few bags. In S. Tang, D.-Z. Du, D. Woodruff, and S. Butenko, editors, *Algorithmic Aspects in Information and Management*, pages 24–35, Cham. Springer International Publishing.

PassMark (2021). *CPU Benchmarks*. `https://www.cpubenchmark.net/` [Accessed: March 2021].

Pferschy, U. and Schauer, J. (2009). The knapsack problem with conflict graphs. *Journal of Graph Algorithms and Applications*, **13**, 233–249.

Ryan, D. M. and Foster, B. A. (2003). On the capacitated vehicle routing problem. *Mathematical Programmming*, **94**, 343–359.

Unlu, Y. and Mason, S. J. (2010). Evaluation of mixed integer programming formulations for non-preemptive parallel machine scheduling problems. *Computers & Industrial Engineering*, **58**(4), 785 – 800.

Wolsey, L. (1998). *Integer Programming*. Wiley Series in Discrete Mathematics and Optimization. Wiley.

Yu, S. and Hung, Y. (2016). Comparisons of three mixed integer programming models for parallel machine scheduling. In *2016 IEEE International Conference on Industrial Engineering and Engineering Management (IEEM)*, pages 917–921.