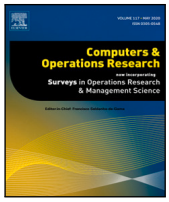
Computers & Operations Research 163 (2024) 106484

Contents lists available at ScienceDirect

Computers and Operations Research

journal homepage: www.elsevier.com/locate/cor



Exact algorithms for a parallel machine scheduling problem with workforce and contiguity constraints

Giulia Caselli [a], Maxence Delorme [b,*], Manuel Iori [c], Carlo Alberto Magni [a]

[a] School of Doctorate E4E (Engineering for Economics - Economics for Engineering), University of Modena and Reggio Emilia, Largo Marco Biagi 10, 41121 Modena, Italy
[b] Department of Econometrics and Operations Research, Tilburg University, Warandelaan 2, 5037 AB, Tilburg, The Netherlands
[c] Department of Sciences and Methods for Engineering, University of Modena and Reggio Emilia, Via Amendola 2, 42122 Reggio Emilia, Italy

ARTICLE INFO

Dataset link: https://github.com/regor-unimore/Parallel-Machine-Scheduling-with-Contiguity

Keywords:
Scheduling
Integer linear programming
Constraint programming
Combinatorial Benders' decomposition
Resource constraints

ABSTRACT

We address a real-world scheduling problem where the objective is to allocate a set of tasks to a set of machines and to a set of workers in such a way that the total weighted tardiness is minimized. Our case study encompasses four types of constraints: precedence, resource, eligibility, and contiguity. While the first three constraints are common in the scheduling literature, contiguity constraints, which can be defined as a form of precedence constraints that requires both a predecessor and its successor to be processed on the same machine with no intermediate jobs in-between (but idle time is allowed), have never been studied in the literature. We present four exact methods to solve the problem: two methods use integer linear programming, one uses constraint programming, and one uses a combinatorial Benders' decomposition. We introduce method-specific strategies to model the contiguity constraints for each of the proposed methods. We empirically evaluate, through an extensive set of computational experiments, the performance of the four methods on a heterogeneous dataset composed of real, realistic, and random instances, and outline that every method offers a competitive advantage on a targeted subset of instances. We also show that our algorithms can be generalized to solve related scheduling problems with contiguity constraints.

1. Introduction

Owing to the numerous practical applications and theoretical properties, the field of Parallel Machines Scheduling Problems (PMSP) is one of the most studied in combinatorial optimization. In a PMSP, one wants to allocate a set of jobs to a set of machines, and determine, for each machine, the order in which the jobs should be processed (see, e.g., Baker and Trietsch, 2019). Some PMSPs can be solved in polynomial time, but most of them are $\mathcal{NP}$-hard (see Pinedo, 2016). Over the years, many additional features have been introduced and studied in PMSPs, by considering the characteristics of the machines (e.g., machine speed), of the jobs (e.g., precedence constraints), and of the objective function (e.g., sum of completion time minimization). A particularly important feature in real-world applications is resource consumption. In a Resource-Constrained Parallel Machine Scheduling Problem (RCPMSP), one also considers a set of resources that must be used in order to process a job on a machine.

Even though RCPMSPs have been extensively studied in the literature (see Edis et al., 2013 for a complete survey; see also Section 2 below), there are practical applications that have not yet been modeled in the literature because they either involve a novel combination of features or because they require a type of constraints that was never studied before. Our work deals with the latter case and originates from a collaboration with the engineering test laboratory of Dana Inc., a company working in the hydraulic automation industry, where complex and customized components of motion systems (such as pumps, motors, and valves) require several tests (e.g., performance or endurance tests) before being delivered to the customers.

The decision problem faced by the company can be modeled as an RCPMSP where the tests (called jobs in the remainder of the paper to stick with the most common scheduling notation) are assigned to machines and workers by meeting a number of operational constraints. The job occupies the machine entirely for a fixed number of days, and for each day the worker is required to process the job for a few hours. A limited number of workers is available every day; each of them can process one or more jobs, without exceeding their daily maximum number of hours.

* Corresponding author.
E-mail addresses: giulia.caselli@unimore.it (G. Caselli), m.delorme@tilburguniversity.edu (M. Delorme), manuel.iori@unimore.it (M. Iori), magni@unimore.it (C.A. Magni).

https://doi.org/10.1016/j.cor.2023.106484
Received 13 December 2022; Received in revised form 5 October 2023; Accepted 17 November 2023
Available online 20 November 2023

The problem also requires to consider three sets of eligibility constraints: (i) a job can only be performed by a subset of workers (for example, a test on a valve must be done by a worker who is an expert in valves); (ii) a job can only be performed by a subset of machines (for example, a performance test on a valve must be done on a machine dedicated to valves performance tests); (iii) a worker can only manipulate a subset of machines (in our case, a worker can only use a machine for which they have a skill certification). In addition, every job has a release date, a duration, a desired due date, and a weight (expressing the priority of the job). The objective is to minimize the total weighted tardiness.

The main novelty of our application comes from the relations existing between pairs of jobs. We consider (i) standard precedence constraints that require a predecessor to be finished before its successor starts and (ii) contiguity constraints, which are precedence constraints that additionally require both a predecessor and its successor to be processed on the same machine with no intermediate jobs in-between (but idle time is allowed). As defined by the company, precedence relations occur between two types of jobs that must be performed on the same product (such as a short performance test followed by a long endurance test on the same hydraulic component), whereas contiguity relations take place on a chain of similar jobs that must be performed on the same product and on the same machine (such as consecutive performance tests on the same hydraulic component with different system conditions, like the oil pressure level for example), but not necessarily by the same worker. To the best of our knowledge, contiguity constraints were never studied in the RCPMSP literature. Indeed, as reported by Khatami et al. (2020), there is a research gap in the area of RCPMSPs with non-ordinary precedence constraints (such as "coupled tasks" in Khatami et al.'s paper or contiguous tasks in our case).

In this work, our first objective is to provide a set of exact algorithms able to solve real instances provided by the company. Inspired by the recent trends in the scheduling literature (see Koné et al., 2011; Lovato et al., 2023; Fang et al., 2021), we propose (i) a descriptive Mixed Integer Linear Programming (MILP) formulation, (ii) an enhanced MILP model, (iii) a new Constraint Programming (CP) formulation, and (iv) a new combinatorial Benders' decomposition that first determines the starting date of every job and then finds a suitable machine and a suitable worker to process each job at the scheduled time. For each method, we also introduce an innovative strategy to model contiguity constraints. Another objective is to determine the effectiveness of our techniques on large size realistic instances (i.e., randomly generated instances aimed at mimicking the dataset provided by the company) and on artificial instances studying the impact of key features such as the presence of precedence constraints, the presence of eligibility constraints, and various worker/machine/job ratios. These experiments show that the effectiveness of the proposed approaches depends on the instance features and, therefore, that no method consistently outperforms the others. Our contribution is not only limited to the problem considered in this work: we show that our algorithms can easily be generalized to model other scheduling problems with contiguity constraints. This can be useful if, for example, Dana Inc. prefers minimizing the total weighted completion time or the total weighted flow time in the future.

The rest of the paper is organized as follows. In Section 2, we provide a review of the RCPMSP literature related to our problem. In Section 3, we give a detailed description of our RCPMSP and introduce the descriptive MILP formulation. The enhanced MILP formulation, the CP model, and the decomposition approach are described in Section 4. Section 5 gives a summary of the results obtained by an extensive set of computational experiments. Finally, concluding remarks are provided in Section 6. A preliminary version of this work in which an MILP formulation was used to solve a single instance with unrelated parallel machines was presented as Caselli et al. (2022).

2. Literature review

Scheduling problems on parallel machines have been extensively studied since the early fifties. We refer to Mokotoff (2001) for a survey on PMSPs, to Pinedo (2016) for a comprehensive review of the scheduling theory, and to Fuchigami and Rangel (2018) for a discussion on analysis and perspectives on scheduling case studies. Following the three-field notation by Graham et al. (1979), our problem can be denoted as $P|res1, \mathcal{M}_j, r_j, d_j, prec, cont|\sum w_j T_j$. Specifically, we aim to minimize the total weighted tardiness ("$\sum w_j T_j$") on identical parallel machines ("P"), with an additional resource ("$res1$"), eligibility constraints ("$\mathcal{M}_j$"), release dates ("r_j"), due dates ("d_j"), and precedence ("$prec$") and contiguity ("$cont$") constraints. To the extent of our knowledge, characteristic "$cont$" has never been formally defined in the three-field notation. In the following, we briefly review closely related scheduling problems and their applications.

Additional resources ($res1$). As stressed by Blazewicz et al. (1983), most RCPMSPs are $\mathcal{NP}$-hard, even though some of them with unit-length jobs are polynomially solvable (see also Lawler et al., 1993; Ventura and Kim, 2000). Various kinds of additional resources were considered in the literature, including human workforce (Seifi et al., 2021; Edis and Ozkarahan, 2011), tools (Ventura and Kim, 2000; Hall et al., 2000), and automated guided vehicles (Ulusoy et al., 1997; Reddy et al., 2022). For a complete survey on PMSPs with additional resources, we refer the reader to Edis et al. (2013). We only mention the work of Edis and Ozkarahan (2012), who used solution methods similar to the ones we propose in this paper. The authors solved a real-world RCPMSP with identical machines by using a two-phase algorithm that first assigns the jobs to the machines and then finds the optimal sequence of jobs on every machine. The first phase was solved with a MILP formulation while both CP and MILP were tested to solve the second phase. The two resulting algorithms were run on realistic instances with up to 100 jobs, 36 machines, and 12 workers, and consistently outperformed the original MILP model.

We also point out that in some RCPMSPs, known as parallel machine flexible-resource scheduling problems, resources can be used to speed-up the processing time of a job (see Daniels et al., 1997; Chen, 2004). This is not the case in our problem because assigning two workers to perform a job does not shorten the job processing time.

Eligibility constraints ($\mathcal{M}_j$). Eligibility constraints are used in many practical applications where jobs must be processed on a specific subset of machines. For a complete survey on PMSPs with eligibility constraints, we refer the reader to Leung and Li (2016). Problems closely related to ours were studied by Afzalirad and Rezaeian (2016) and Edis and Ozkarahan (2011). The former introduced metaheuristics to solve a PMSP with unrelated machines, several additional resources, eligibility constraints, and precedence constraints. The latter used a combination of CP and MILP to solve a PMSP with identical machines, one additional resource, and eligibility constraints. To the best of our knowledge, our work is the first in the RCPMSP literature dealing with multiple eligibility constraints simultaneously: worker/machine, worker/job, and job/machine. We point out, however, that such multiple eligibility constraints could be modeled within the framework of the multi-mode resource-constrained project scheduling problem, with one resource per machine and per worker, and, for each job, one mode per compatible (i.e., eligible) machine/worker pair. For an extensive literature review on multi-mode resource-constrained project scheduling problems, we refer the reader to Section 2.4 of the recent survey by Hartmann and Briskorn (2022) focusing on resource-constrained project scheduling problem extensions, and to Section 5 of the survey by Węglarz et al. (2011) focusing on multi-mode project scheduling problems.

Table 1
Categorization of the recent related PMSP literature.

Reference	Constraints						Approach			
	$res1$	$\mathcal{M}_j$	r_j	d_j	$prec$	$cont$	HM	MILP	CP	DA
Afzalirad and Rezaeian (2016)	✓	✓	✓		✓		✓	✓		
Chen (2004)	✓			✓						✓
Chudak and Shmoys (1999)					✓		✓			
Daniels et al. (1997)	✓						✓	✓		
Edis and Ozkarahan (2011)	✓	✓						✓	✓	
Edis and Ozkarahan (2012)	✓	✓						✓	✓	✓
Fang et al. (2021)	✓						✓	✓		✓
Fleszar and Hindi (2018)	✓							✓	✓	✓
Gökgür et al. (2018)	✓								✓	
Hooker (2006)		✓	✓	✓	✓			✓	✓	✓
Hu et al. (2010)		✓			✓		✓			
Kramer et al. (2019)				✓				✓		
Li et al. (2022)			✓				✓	✓		
Mischek and Musliu (2021)	✓	✓	✓	✓	✓		✓		✓	
Su et al. (2017)		✓		✓			✓			
Ventura and Kim (2000)	✓			✓				✓		
Caselli et al. (2022)	✓	✓	✓	✓	✓	✓		✓		
This paper	✓	✓	✓	✓	✓	✓		✓	✓	✓

Due dates and total weighted tardiness minimization ($d_j|\sum w_jT_j$). Total weighted tardiness is one of the most common objective functions in the scheduling literature, because it models many practical problems in industrial engineering and production management (see, e.g., Cheng and Sin, 1990). We refer the reader to Koulamas (1994) for a survey focused on total tardiness minimization in scheduling problems and to Janiak et al. (2015) for a survey on scheduling problems with due windows. We also mention the relevant work of Su et al. (2017), who studied a PMSP with eligibility constraints where the objective was to minimize the total weighted tardiness, and the recent work of Kramer et al. (2019), who proposed MILP formulations for a PMSP where the objective is to minimize the total weighted completion time (equivalent to the total weighted tardiness if all due dates were set to 0).

Release dates (r_j). The addition of release dates for jobs tends to make PMSPs harder to solve. Graham et al. (1979) showed that $1|r_j,d_j|\sum w_jT_j$ is $\mathcal{NP}$-hard while $1|d_j|\sum w_jT_j$ can be solved in polynomial time. We refer the reader to Lee (2004) for a survey on machine scheduling problems with availability constraints. Several recent works include release dates in their PMSPs (see, e.g., Afzalirad and Rezaeian, 2016; Li et al., 2022).

Precedence relations between jobs ($prec$). The complexity of scheduling problems with precedence constraints has been studied since the seventies (see Lenstra and Rinnooy Kan, 1978). Chudak and Shmoys (1999) introduced approximation algorithms for the PMSP with precedence constraints where each machine has its own speed. Hu et al. (2010) proposed a heuristic for a realistic PMSP with precedence and eligibility constraints. An extended literature on precedence constraints is also available in the area of assembly line balancing problems (Becker and Scholl, 2006), bin packing (Kramer et al., 2017), and (resource-constrained) project scheduling problems (Hartmann and Briskorn, 2022).

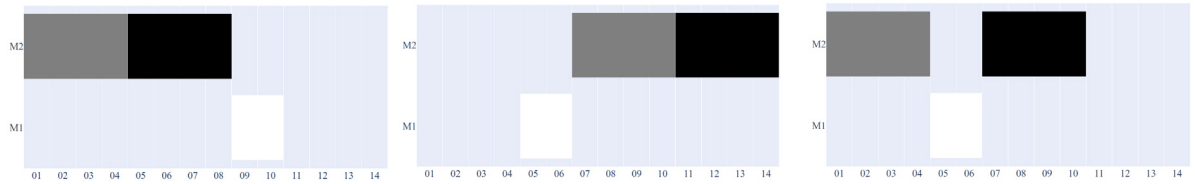
Contiguity constraints ($cont$). Contiguity constraints can be seen as a special form of precedence constraints where the successor of a job must be processed on the same machine as its predecessor and without any intermediate jobs in-between (but idle time is allowed). The roots of contiguity constraints can be found in batch scheduling problems (see Potts and Kovalyov, 2000 for an extensive survey) and scheduling problems with set-up times and changeover costs (see Bruno and Downey, 1978), where the concept of families of jobs that must be performed together was introduced. In recent years, concepts that are related to contiguity constraints have been introduced. We mention the recent work of Mischek and Musliu (2021), who introduced the test laboratory scheduling problem, an extension of the resource-constrained project scheduling problem inspired by a real-world problem similar to ours. They introduced the notion of "link" between two jobs to indicate that they need to be performed by the same employee. In contrast, we originally consider the interactions between contiguity constraints and workforce. Khatami et al. (2020) surveyed coupled task scheduling problems, where it is required to have an exact time interval between the two jobs of a pair and this time can be used to process other jobs (see also Khatami and Salehipour, 2021a,b for other recent works). While the definition of contiguity in our problem has a few similarities with the aforementioned constraints, it cannot exactly be considered just a combination of existing concepts, because we formalize the notion of contiguity constraints between pairs of jobs, which enriches the RCPMSP literature. We mention that a different definition of contiguity in which one requires a pair of jobs to be scheduled in consecutive machines exists in the cutting and packing literature. To the best of our knowledge, that definition was only used to introduce $P|cont|C_{max}$, a relaxation of the well-known strip packing problem (see Côté et al., 2014).

We categorize in Table 1 the relevant PMSP literature discussed in this work and identify, for each paper, the types of constraints considered (among those studied in our problem) and the type of approaches used: heuristics and metaheuristics (HM), MILP models, CP models, or decomposition approaches (DA).

3. Problem statement and descriptive model

In our RCPMSP, we are given a set J of jobs that must be scheduled on a set M of machines by a set K of workers in the time horizon $\mathcal{T} = 0,\dots,t_{max}$ expressed in days. Every job $j \in J$ has a release date r_j, a due date d_j, a daily resource consumption q_j (number of working hours), a processing time p_j, and a weight w_j. The weighted tardiness of each job is computed by multiplying its weight by its delay with respect to the due date, if any. The aim of our RCPMSP is to minimize the total weighted tardiness (z). To this aim, each job j must be processed on a machine i that belongs to the subset $M_j \subseteq M$ of machines compatible with job j, and by a worker k that belongs to the subset $K_{ij} \subseteq K$ of workers compatible with both job j and machine i. We also define $K_j \subseteq K$ as the set of workers compatible with job j. We assume that every machine can be used every day of the time horizon, while every worker k is available for u_{kt} hours in day t ($k \in K, t \in \mathcal{T}$).

We are given a set P of precedence relations and a set Q of contiguity relations. A precedence relation $(j,l) \in P$ between job j and job l implies that the starting time of l must be greater than or equal to the completion time of j. A contiguity relation $(j,l) \in Q$ between job j and job l states that the starting time of l must be greater than or equal



(a) $z = 2$ (b) $z = 10$ (c) $z = 1$

Fig. 1. Numerical example on contiguity relation.

Table 2
Mathematical notation.

Notation	Definition
J	Set of jobs
M	Set of available machines
M_j	Set of machines compatible with job j
K	Set of available workers
K_j	Set of workers compatible with job j
K_{ij}	Set of workers compatible with job j and machine i
$\mathcal{T}$	Set of days in the time horizon ($\mathcal{T} = 0, \dots, t_{\max}$)
r_j	Release date of job j
d_j	Due date of job j
q_j	Daily resource consumption of job j (hours per day)
p_j	Processing time of job j (days)
w_j	Weight of job j
u_{kt}	Working availability of worker k in day t (hours)
P	Set of precedence relations
Q	Set of contiguity relations
S_{jt}	Set of starting times for which job j would not be finished by day t

to the completion time of j and that the two jobs must be processed on the same machine without any other jobs in-between (but idle time is allowed).

Interestingly, the notion of contiguity is only relevant in our problem because of the workers. One could think that the tasks involved in a contiguity constraint could simply be merged so that they are forced to be processed one after the other on the same machine. However, doing so would remove the possibility to slightly postpone the second task to free its associated worker so that they become available to perform another (more urgent) job. A numerical example illustrating this situation is given in Fig. 1. We consider three jobs $J = \{1, 2, 3\}$ (in white, gray, and black in the figure) with release dates $r_j = \{4, 0, 4\}$, due dates $d_j = \{8, 5, 9\}$, processing times $p_j = \{2, 4, 4\}$, and weights $w_j = \{1, 1, 1\}$. There is a contiguity relation between jobs 2 and 3 ($Q = \{(2, 3)\}$). We consider two machines $M = \{M1, M2\}$ with $M1$ compatible with job 1 and $M2$ compatible with jobs 2 and 3, and a single worker compatible with every job and every machine who can perform one job at a time. We show in Figs. 1(a) and 1(b) the two possible outcomes if contiguous jobs 2 and 3 are merged. In that case, the total weighted tardiness (z) is either equal to 2 if the merged jobs are scheduled before job 1 or it is equal to 10 if the merged jobs are scheduled after job 1. If a contiguity relation is considered between the two jobs instead, then a better solution with objective value 1 can be found as displayed in Fig. 1(c). We point out that if there is a contiguity relation (j, l) between two jobs j and l, then the sets of compatible machines M_j and M_l should be identical. If this is not the case, then a simple preprocessing step sets M_j and M_l to $M_j \cap M_l$ for every $(j, l) \in Q$.

For each job j and day t, we also define the set of starting times S_{jt} for which a machine would still be occupied at day t if job j were to start at any day τ that belongs to S_{jt}. Set S_{jt} is equal to $\{\emptyset\}$ if $r_j > t$, to $\{r_j, \dots, t\}$ if $t \geq r_j \geq t - p_j + 1$, and to $\{t - p_j + 1, \dots, t\}$ otherwise. The mathematical notation is summarized in Table 2.

Our RCPMSP can be modeled using a so-called discrete-time formulation (see Pritsker et al., 1969) where binary decision variable x_{ijkt} takes the value 1 if job j is processed on machine i by worker k on day t ($j \in J, i \in M_j, k \in K_{ij}, t \in \mathcal{T}$) and where continuous decision variables C_j and T_j indicate the completion time of job j and its tardiness (i.e., the maximum between 0 and $C_j - d_j$), respectively. We obtain the following MILP model:

$$\min \sum_{j \in J} w_j T_j \tag{1}$$

$$\text{s.t.} \sum_{i \in M_j} \sum_{k \in K_{ij}} \sum_{t \in \mathcal{T}} x_{ijkt} = 1 \qquad j \in J \tag{2}$$

$$\sum_{j \in J} \sum_{k \in K_{ij}} \sum_{\tau \in S_{jt}} x_{ijk\tau} \leq 1 \qquad i \in M, t \in \mathcal{T} \tag{3}$$

$$\sum_{j \in J} \sum_{i \in M_j : k \in K_{ij}} \sum_{\tau \in S_{jt}} q_j x_{ijk\tau} \leq u_{kt} \qquad k \in K, t \in \mathcal{T} \tag{4}$$

$$C_j = \sum_{i \in M_j} \sum_{k \in K_{ij}} \sum_{t \in \mathcal{T}} x_{ijkt}(t + p_j) \qquad j \in J \tag{5}$$

$$C_l - p_l \geq C_j \qquad (j, l) \in P \cup Q \tag{6}$$

$$\sum_{k \in K_{ij}} \sum_{t \in \mathcal{T}} x_{ijkt} = \sum_{k \in K_{il}} \sum_{t \in \mathcal{T}} x_{ilkt} \qquad (j, l) \in Q, i \in M_j \tag{7}$$

$$\sum_{j' \in J \setminus \{j,l\}} \sum_{k \in K_{ij'}} x_{ij'kt} \leq 1 - \sum_{k \in K_{ij}} \sum_{\tau = r_j}^{t} x_{ijk\tau} + \sum_{k \in K_{il}} \sum_{\tau = r_l}^{t} x_{ilk\tau} \qquad (j, l) \in Q, i \in M_j, t \in \mathcal{T} \tag{8}$$

$$T_j \geq C_j - d_j \qquad j \in J \tag{9}$$

$$x_{ijkt} = 0 \qquad j \in J, i \in M_j, k \in K_{ij}, t \in \mathcal{T} \setminus \{r_j, \dots, t_{max} - p_j\} \tag{10}$$

$$x_{ijkt} \in \{0, 1\} \qquad j \in J, i \in M_j, k \in K_{ij}, t \in \mathcal{T} \tag{11}$$

$$C_j, T_j \geq 0 \qquad j \in J \tag{12}$$

The objective function (1) minimizes the total weighted tardiness. Constraints (2) guarantee that each job is processed exactly once and is assigned to exactly one compatible machine and one compatible worker. Constraints (3) ensure that each machine processes at most one job at the same time (i.e., one job per day). Constraints (4) guarantee that no operator works more than the number of hours they are supposed to work in each day of the time horizon. Constraints (5) define the completion time of each job. Constraints (6) guarantee that the order between jobs defined by precedence and contiguity relations is respected. Constraints (7) make sure that two contiguous jobs are processed on the same machine. Constraints (8) forbid any job j' to be processed on a machine i at day t (i.e., $\sum_{j' \in J \setminus \{j,l\}} \sum_{k \in K_{ij'}} x_{ij'kt} = 1$) if that machine has processed a job j, the first element of a contiguous pair, before day t (i.e., $\sum_{k \in K_{ij}} \sum_{\tau = r_j}^{t} x_{ijk\tau} = 1$) but has not yet processed job l, the second element of that same contiguous pair, by day t (i.e., $\sum_{k \in K_{il}} \sum_{\tau = r_l}^{t} x_{ilk\tau} = 0$). Constraints (9) define the tardiness of the jobs, constraints (10) prevent any job from starting before its release date or finishing after the time horizon (note that in the model implementation the variables set to zero are not created), and constraints (11) and (12) limit the variable domains. We point out that the innovative aspect of this formulation comes from constraints

(7) and (8), which are used to model contiguity. The other constraints are fairly common in the scheduling literature (see, e.g., Koné et al., 2011). Model (1)–(12) can be solved with an MILP solver to produce an optimal solution, but computational experiments have shown that the model could quickly become too large to be solved in a reasonable time (it involves $O(|J| \cdot |M| \cdot |K| \cdot t_{max})$ variables).

4. Advanced algorithms

In this section, we introduce an enhanced MILP model, a CP model, and a combinatorial Benders' decomposition for our RCPMSP.

4.1. Enhanced MILP model

There are often more than one MILP formulation to model a given combinatorial optimization problem. The literature has shown that, for a given instance of a problem and a given MILP solver, some MILP formulations could be faster to solve than others, all else being equal. This difference can be explained by major differences in terms of number of variables, number of constraints, number of non-zero elements, and continuous relaxation value (among others). For example, we refer the reader to Koné et al. (2011) for a comparison of MILP formulations for the resource-constrained project scheduling problem. A question frequently arising in MILP modeling is whether or not one should use a set of α-index variables or two sets of (α-1)-index variables together with a set of linking constraints. While the former is usually shown to have a better continuous relaxation value, the latter tends to involve much fewer variables. As preliminary results showed that the model size was an issue with model (1)–(12), we tried a version of the model using two sets of 3-index variables instead of one set of 4-index variables: we now define x^m_{ijt} that takes the value 1 if job j is processed on machine i on day t ($j \in J, i \in M_j, t \in \mathcal{T}$) and x^w_{jkt} that takes the value 1 if job j is processed by worker k on day t ($j \in J, k \in K_j, t \in \mathcal{T}$). The resulting MILP model, which uses $O(|J| \cdot |M + K| \cdot t_{max})$ variables, is defined as follows:

$$\min \quad \sum_{j \in J} w_j T_j \tag{13}$$

$$\text{s.t.} \quad \sum_{i \in M_j} \sum_{t \in \mathcal{T}} x^m_{ijt} = 1 \qquad j \in J \tag{14}$$

$$\sum_{k \in K_j} \sum_{t \in \mathcal{T}} x^w_{jkt} = 1 \qquad j \in J \tag{15}$$

$$\sum_{j \in J} \sum_{\tau \in S_{jt}} x^m_{ij\tau} \le 1 \qquad i \in M, t \in \mathcal{T} \tag{16}$$

$$\sum_{j \in J} \sum_{\tau \in S_{jt}} q_j x^w_{jk\tau} \le u_{kt} \qquad k \in K, t \in \mathcal{T} \tag{17}$$

$$C_j = \sum_{i \in M_j} \sum_{t \in \mathcal{T}} x^m_{ijt}(t + p_j) \qquad j \in J \tag{18}$$

$$C_l - p_l \ge C_j \qquad (j,l) \in P \cup Q \tag{19}$$

$$\sum_{t \in \mathcal{T}} x^m_{ijt} = \sum_{t \in \mathcal{T}} x^m_{ilt} \qquad i \in M, (j,l) \in Q \tag{20}$$

$$\sum_{j' \in J \setminus \{j,l\}} x^m_{ij't} \le 1 - \sum_{\tau=r_j}^{t} x^m_{ij\tau} + \sum_{\tau=r_l}^{t} x^m_{il\tau} \qquad i \in M, t \in \mathcal{T}, (j,l) \in Q \tag{21}$$

$$T_j \ge C_j - d_j \qquad j \in J \tag{22}$$

$$\sum_{t \in \mathcal{T}} x^m_{ijt} \le \sum_{t \in \mathcal{T}} \sum_{k \in K_{ij}} x^w_{jkt} \qquad j \in J, i \in M_j, \tag{23}$$

$$\sum_{i \in M_j} x^m_{ijt} = \sum_{k \in K_j} x^w_{jkt} \qquad j \in J, t \in \mathcal{T} \tag{24}$$

$$x^m_{ijt} = 0 \qquad j \in J, i \in M_j, t \in \mathcal{T} \setminus \{r_j, \dots, t_{max} - p_j\} \tag{25}$$

$$x^w_{jkt} = 0 \qquad j \in J, k \in K_j, t \in \mathcal{T} \setminus \{r_j, \dots, t_{max} - p_j\} \tag{26}$$

$$x^m_{ijt} \in \{0,1\} \qquad j \in J, i \in M_j, t \in \mathcal{T} \tag{27}$$

$$x^w_{jkt} \in \{0,1\} \qquad j \in J, k \in K_j, t \in \mathcal{T} \tag{28}$$

$$C_j, T_j \ge 0 \qquad j \in J \tag{29}$$

Model (13)–(29) is an adaptation of model (1)–(12) where the additional constraints (23) allow a job j to be assigned to a compatible machine i only if j is assigned to a worker k that is compatible with both j and i, and the additional constraints (24) link the two sets of 3-index variables.

4.2. Constraint programming model

In the last decade, the literature has shown that CP is particularly effective to solve highly constrained combinatorial optimization problems (see, e.g., Jain and Grossmann, 2001 and Gedik et al., 2016). While CP is sometimes used as a stand-alone approach, it can also be integrated into more sophisticated algorithms, either to quickly find a feasible solution to serve as a warm start for an MILP model (see, e.g., Delorme and Santini, 2022) or to solve the subproblem of a decomposition approach (see, e.g., Delorme et al., 2017). We refer the reader to Bockmayr and Hooker (2005) for an outline of CP basic concepts and its integration with MILP.

Several works using CP models to solve scheduling problems have been proposed in the PMSP literature (see Hooker, 2006; Gedik et al., 2016; Fleszar and Hindi, 2018; Gökgür et al., 2018 and Lovato et al., 2023). In the following, we describe our CP formulation together with a detailed explanation of every necessary function. To take the worker availability into account, we consider that every worker is initially available at full capacity every day and we use a set of worker-dependent dummy jobs with pre-defined resource consumption and starting/ending dates to represent the possible reduction of the full capacity. To model the eligibility constraints, we use the multi-mode resource-constrained project scheduling problem transformation mentioned in Section 2, which is to consider $|M|+|K|$ resources and create, for each job j, one mode for every single compatible machine/worker pair (i,k) ($i \in M_j, k \in K_{ij}$) that occupies machine i and consumes resource k for p_j days. To represent the contiguity relations $(j,l) \in Q$, we use a dummy job with one mode per compatible machine i ($i \in M_j$) that occupies machine i and no worker resource for an unrestricted duration. This dummy job must start exactly when job j finishes and must stop exactly when job l starts.

We make use of the following sets of interval variables to model the problem:

- ivl_j, interval variable that represents the execution of job j;
- ivl_m_{ijk}, interval variable that represents the execution of job j using the mode that occupies machine i and worker k;
- $ivld_{jl}$, interval variable that represents the execution of the dummy job associated with the contiguity relation between jobs j and l;
- $ivld\_m_{ijl}$, interval variable that represents the execution of the dummy job associated with the contiguity between jobs j and l using the mode that consumes machine i.

Every interval variable must be processed in exactly one of its modes, and if a constraint applies to a specific interval variable, then this constraint is propagated to each of the variable modes. For example, if a constraint states that ivl_j cannot start before day r_j for a given job j, then none of the ivl_m_{ijk} involving job j can start before day r_j.

In our CP formulation, we minimize the total weighted tardiness:

$$\min \sum_{j \in J} w_j * \texttt{max}(0, \texttt{endof}(ivl_j) - d_j) \tag{30}$$

where $\texttt{endof}(ivl_j)$ indicates the completion time of job j. Constraints (31)–(32) limit the domain of the variables so that the jobs are processed between their release date and t_{max}:

$$ivl_j.\texttt{setStartMin}(r_j) \qquad j \in J \tag{31}$$

$$ivl_j.\texttt{setStartMax}(t_{max} - p_j) \quad j \in J \tag{32}$$

The following constraints fix the duration of each job j to be equal to p_j:

$$ivl_j.\texttt{setSizeMin}(p_j) \quad j \in J \tag{33}$$

$$ivl_j.\texttt{setSizeMax}(p_j) \quad j \in J \tag{34}$$

We use the constraints `alternative` to ensure that exactly one mode ivl_m_{ijk} ($i \in M_j, k \in K_{ij}$) per job j ($j \in J$) is present in the solution:

$$\texttt{alternative}(ivl_j, ivl\_m_{ijk} : \forall i \in M_j, k \in K_{ij}) \quad j \in J \tag{35}$$

To guarantee that each machine processes at most one job per day (whether the job is a real one requiring a worker or a dummy one representing an idle time on the machine), we impose

$$\sum_{j \in J} \sum_{k \in K_{ij}} \texttt{pulse}(ivl\_m_{ijk}, 1) + \sum_{(j,l) \in Q : i \in M_j} \texttt{pulse}(ivld\_m_{ijl}, 1) \leq 1 \quad i \in M \tag{36}$$

where $\texttt{pulse}(ivl\_m_{ijk}, 1)$ counts one unit of occupation for machine i from the starting time of ivl_m_{ijk} to its completion time, provided that mode ivl_m_{ijk} is used. Similarly, to guarantee that every worker is only active for the number of hours they are supposed to work, we add

$$\sum_{j \in J} \sum_{i \in M_j : k \in K_{ij}} \texttt{pulse}(ivl\_m_{ijk}, q_j) + \sum_{h \in H_k} \texttt{pulse}(ivl_h, q_h) \leq \max_{t \in \mathcal{T}} \{u_{kt}\} \quad k \in K \tag{37}$$

where $\max_{t \in \mathcal{T}} \{u_{kt}\}$ is the number of hours per day worker k is available when they work at full capacity, and where H_k contains the dummy jobs with fixed starting/ending dates and the resource consumption associated with worker k that is used to model a decrease in the worker availability (e.g., for holidays). Precedence relations (including those coming from contiguity) are modeled as

$$\texttt{endBeforeStart}(ivl_j, ivl_l) \quad (j,l) \in P \cup Q \tag{38}$$

where $\texttt{endBeforeStart}(ivl_j, ivl_l)$ indicates that ivl_j must end before ivl_l can start. The supplementary constraints related to contiguity relations are enforced by

$$\sum_{k \in K_{ij}} \texttt{presenceOf}(ivl\_m_{ijk}) = \sum_{k \in K_{il}} \texttt{presenceOf}(ivl\_m_{ilk}) \quad (j,l) \in Q_j, i \in M_j \tag{39}$$

$$\sum_{k \in K_{ij}} \texttt{presenceOf}(ivl\_m_{ijk}) = \texttt{presenceOf}(ivld\_m_{ijl}) \qquad (j,l) \in Q_j, i \in M_j \tag{40}$$

$$\texttt{StartAtEnd}(ivld_{jl}, ivl_j) \qquad (j,l) \in Q \tag{41}$$

$$\texttt{EndAtStart}(ivld_{jl}, ivl_l) \qquad (j,l) \in Q \tag{42}$$

$$\texttt{alternative}(ivld_{jl}, ivld\_m_{ijl} : \forall i \in M_j) \qquad (j,l) \in Q \tag{43}$$

Constraints (39) and (40) use job modes to make sure that two contiguous jobs j and l and their associated dummy job $ivld_{jl}$ are processed on the same machine. To do so, constraint $\texttt{presenceOf}(ivl\_m_{ijk})$ returns true if ivl_m_{ijk} is present in the solution. Constraints (41) and (42) force the dummy job $ivld_{jl}$ to start at the end of job j and finish at the start of job l. To do so, constraints $\texttt{StartAtEnd}(ivld_{jl}, ivl_j)$ and $\texttt{EndAtStart}(ivld_{jl}, ivl_l)$ impose an exact starting and ending time for $ivld_{jl}$. Finally, constraints (43) ensure that exactly one mode among those associated with contiguity relation (j,l) is selected.

4.3. Combinatorial Benders' decomposition

Benders' decomposition, which was introduced more than sixty years ago by Benders (1962), splits a large MILP problem (called *original problem* afterwards) into two problems, called *master problem* (MP) and *subproblem* (SP). The main idea behind the decomposition is to solve an MP of reasonable size (typically, a relaxation of the original problem where a set of variables/constraints is aggregated or removed) to obtain a solution ζ and check in the SP if ζ is also feasible for the original problem. If that is the case, then ζ is also optimal for the original problem, otherwise a cut is added to the MP, so that ζ cannot be generated once more, and the MP is solved again.

Benders' decomposition has significantly evolved during the last decades. In the seminal work of Benders (1962), the MP was an MILP and the SP was an LP. A few years later, Geoffrion (1972) generalized the decomposition to the case where the SP too was an MILP. Later on, Hooker and Ottosson (2003) proposed the logic-based Benders' decomposition, where the SP was modeled with CP. A major advance in the area can be attributed to Codato and Fischetti (2006), who introduced the concept of combinatorial Benders' cuts by searching for the smallest subset of variables in the MP solution that causes infeasibility. Even though such cuts can be costly to find, they were shown to be more effective in practice because they strongly reduce the number of MP/SP iterations since they cut larger portions of the solution space. Another advance was proposed by Côté et al. (2014) who used a lifting procedure to increase even further the solution space removed by the cuts. Recent successful applications can be found in Dell'Amico et al. (2019), Fang et al. (2021), Karlsson and Rönnberg (2022), and Seo et al. (2022), among others.

In our decomposition, the MP determines the starting day of every job while taking into account the precedence constraints and an aggregated form of resource constraints (derived from the machines and the workers). The SP tries to assign every job to a suitable machine and a suitable worker while also ensuring the contiguity relations among jobs. In such a decomposition framework, only feasibility cuts are added to the MP. Indeed, optimality cuts are never needed because the objective value of a given MP solution is always equal to the objective value of the corresponding solution for the original problem (if such a corresponding solution was shown to exist by the SP). Preliminary experiments showed that, most of the time, only a few MP/SP iterations were needed to reach an optimal solution by using this strategy. An alternative idea would be to take the decisions in reverse order and assign workers and machines to jobs in the MP and then determine the best schedule in the SP. Such a decomposition framework would require optimality cuts as the objective value of a given MP solution for the original problem would only be known after solving the SP. Considering that many instances have a non-zero optimal value and that it is difficult to get any non-trivial bound on the total weighted tardiness in the MP without knowing the starting dates of the jobs, it is expected that using such a strategy would result in significantly more MP/SP iterations. We thus opted not to investigate this alternative idea. In the following, we describe each component of the decomposition method in more detail.

Master problem. In the MP, we only consider two resources: one is an aggregation of the $|M|$ machines and the other one is an aggregation of the $|K|$ workers. Therefore, the MP does not take into account the eligibility constraints and only considers a relaxed version of the contiguity ones. The MP uses a new set of binary variables y defined as

$$y_{jt} = \sum_{i \in M_j} \sum_{k \in K_{ij}} x_{ijkt}$$

where y_{jt} takes the value 1 if job j starts at day t, and 0 otherwise ($j \in J, t \in \mathcal{T}$). We re-use the two sets of continuous variables C_j and T_j to represent the completion time and tardiness of job j, respectively.

The MP is defined as follows:

$$\min \quad \sum_{j\in J} w_j T_j \tag{44}$$

$$\text{s.t.} \quad \sum_{t\in\mathcal{T}} y_{jt} = 1 \qquad j\in J \tag{45}$$

$$\sum_{j'\in J}\sum_{\tau\in S_{j't}} y_{j'\tau} \le |M| - \sum_{(j,l)\in Q}\sum_{\tau=r_j}^{t-p_j} y_{j\tau} + \sum_{(j,l)\in Q}\sum_{\tau=r_l}^{t} y_{l\tau} \qquad t\in\mathcal{T} \tag{46}$$

$$\sum_{j\in J}\sum_{\tau\in S_{jt}} q_j y_{j\tau} \le \sum_{k\in K} u_{kt} \qquad t\in\mathcal{T} \tag{47}$$

$$C_j = \sum_{t\in\mathcal{T}} y_{jt}(t+p_j) \qquad j\in J \tag{48}$$

$$C_l - p_l \ge C_j \qquad (j,l)\in P\cup Q \tag{49}$$

$$T_j \ge C_j - d_j \qquad j\in J \tag{50}$$

$$\sum_{(j,t)\in\zeta} y_{jt} \le |\zeta| - 1 \qquad \zeta\in Z \tag{51}$$

$$y_{jt} = 0 \qquad j\in J, t\in\mathcal{T}\setminus\{r_j,\dots,t_{max}-p_j\} \tag{52}$$

$$y_{jt}\in\{0,1\} \qquad j\in J, t\in\mathcal{T} \tag{53}$$

$$C_j, T_j \ge 0 \qquad j\in J \tag{54}$$

which is an adaptation of model (1)–(12) using the two-index variables y_{jt}. The only differences are for constraints (51), which forbid the solutions that were previously shown to be infeasible by the SP, and for constraints (46), which merge constraints (3) and (8). More specifically, constraints (46) limit the number of jobs running in parallel to the total number $|M|$ of machines minus the number of machines that cannot be used because of the contiguity relations. For every day t, the latter quantity can be computed as the number of contiguity pairs for which the first member has been completed by day t (i.e., $\sum_{(j,l)\in Q}\sum_{\tau=r_j}^{t-p_j} y_{j\tau}$) minus the number of contiguity pairs for which the second member has not started by day t (i.e., $\sum_{(j,l)\in Q}\sum_{\tau=r_l}^{t} y_{l\tau}$).

Subproblem. Given an MP solution $\{\widetilde{y_{jt}}, \widetilde{C_j}\}$ indicating the starting time of every job, the SP determines whether or not there is a feasible job/machine/worker allocation respecting both the eligibility and contiguity constraints. The SP uses a set of binary decision variables ξ_{ijk} that take the value 1 if job j is processed on machine i by worker k, and 0 otherwise ($j\in J, i\in M_j, k\in K_{ij}$). Let $J^Q_{jl} = \{j'\in J\setminus\{j,l\} : \widetilde{C_j} - p_j < \widetilde{C_{j'}},\ \widetilde{C_{j'}} - p_{j'} < \widetilde{C_l}\}$. The SP becomes

$$\min \quad 0 \tag{55}$$

$$\text{s.t.} \quad \sum_{i\in M_j}\sum_{k\in K_{ij}} \xi_{ijk} = 1 \qquad j\in J \tag{56}$$

$$\sum_{\substack{j\in J\\ \widetilde{C_j}-p_j\in S_{jt}}}\sum_{k\in K_{ij}} \xi_{ijk} \le 1 \qquad i\in M, t\in\mathcal{T} \tag{57}$$

$$\sum_{\substack{j\in J\\ \widetilde{C_j}-p_j\in S_{jt}}}\sum_{i\in M_j: k\in K_{ij}} q_j\xi_{ijk} \le u_{kt} \qquad k\in K, t\in\mathcal{T} \tag{58}$$

$$\sum_{k\in K_{ij}} \xi_{ijk} = \sum_{k\in K_{il}} \xi_{ilk} \qquad (j,l)\in Q, i\in M_j \tag{59}$$

$$\sum_{j'\in J^Q_{jl}}\sum_{k\in K_{ij'}} \xi_{ij'k} + \sum_{k\in K_{ij}} \xi_{ijk} \le 1 \qquad (j,l)\in Q, i\in M_j \tag{60}$$

$$\xi_{ijk}\in\{0,1\} \qquad j\in J, i\in M_j, k\in K_{ij} \tag{61}$$

Model (55)–(61) is an adaptation of model (1)–(12) where the starting date of each job is fixed. We point out the necessary adaptations in resource constraints (57) and (58), where, for each day t, the ξ_{ijk} sum is taken only over the jobs j that are active at day t (i.e., those whose starting time $\widetilde{C_j} - p_j$ is in set S_{jt}). As far as constraints (60) are concerned, they simply indicate that a job j' cannot be processed on the same machine as a pair of contiguous jobs $(j,l)\in Q$ with which it is in conflict. A job j' is in conflict with a pair of contiguous jobs (j,l) if the completion time of job j' is greater than the starting time of job j (i.e., $\widetilde{C_{j'}} > \widetilde{C_j} - p_j$) and if the starting time of job j' is smaller than the completion time of job l (i.e., $\widetilde{C_{j'}} - p_{j'} < \widetilde{C_l}$). We also tried a version of the SP that uses two sets of two-index binary variables instead of one set of three-index variables, but we did not observe significant improvements in terms of average computation time.

Cut generation and overall framework. Given an MP solution $\{\widetilde{y_{jt}}, \widetilde{C_j}\}$ that was proven to be infeasible by the SP, a valid cut to prevent $\{\widetilde{y_{jt}}, \widetilde{C_j}\}$ to be generated again is

$$\sum_{(j,t)\in\zeta} y_{jt} \le |J| - 1 \tag{62}$$

where ζ contains all the variables y_{jt} ($j\in J, t\in\mathcal{T}$) for which $\widetilde{y_{jt}}$ is equal to 1 in the solution. Such a constraint is known as a "no-good cut" in the literature (Hooker, 2011) and does not forbid any integer solution besides $\{\widetilde{y_{jt}}, \widetilde{C_j}\}$. In other words, it only forbids the integer solution in which every job j starts exactly at day $\widetilde{C_j} - p_j$.

Stronger cuts can be obtained by using the combinatorial Benders' cuts of Codato and Fischetti (2006). To do so, one needs to find a reduced subset of jobs J' for which the SP remains infeasible. In other words, one now aims at forbidding all the integer solutions in which every job j in J' starts exactly at day $\widetilde{C_j} - p_j$, regardless of the starting day of the jobs in $J\setminus J'$. This enhanced cut therefore forbids $O\left(\prod_{j\in J\setminus J'}(t_{max} - r_j)\right)$ solutions. As more MP solutions are removed when $|J'|$ decreases, we are interested in finding the smallest subset of jobs J' causing infeasibility for the SP, also known as the Minimum Infeasible Subset (MIS) in the literature. Since finding a MIS is usually $\mathcal{NP}$-hard, a common strategy is to determine it heuristically by removing the set of variables that is the least likely to cause infeasibility, solve the SP again, and iterate until the SP becomes feasible. In our case, this difficulty is confirmed by the $\mathcal{NP}$-completeness of the SP. We thus adopt the same strategy.

In detail, the variables that are less likely to cause infeasibility in model (55)–(61) are the ξ_{ijk}'s associated with the jobs that (i) are short, (ii) are involved in few precedence/contiguity constraints, (iii) are compatible with most machines and workers, and (iv) have their completion interval $[\widetilde{C_j} - p_j; \widetilde{C_j}]$ intersecting with few other jobs. Preliminary experiments showed that criterion (i) was the most relevant in our instances. Therefore, we determine the minimum subset of jobs J' that cause infeasibility in the SP by initializing J' to J and removing the job with the smallest duration from J' until the SP becomes feasible. The last subset J' for which the SP is infeasible is then used to generate the combinatorial Benders' cut:

$$\sum_{(j,t)\in\zeta'} y_{jt} \le |\zeta'| - 1. \tag{63}$$

This is the modified version of (51) where ζ' contains all the variables y_{jt} ($j\in J', t\in\mathcal{T}$) for which $\widetilde{y_{jt}}$ is equal to 1 in the MP solution. A flowchart summarizing the main idea behind the combinatorial Benders' decomposition is shown in Fig. 2.

In our implementation, we embedded the SP solution in a callback function provided by the MILP solver we adopted. In this way, we only need to solve the MP once. We also tried the lifting procedure proposed by Côté et al. (2014). The idea behind such a cut lifting is to forbid integer solutions in which every job $j\in J$ starts within a job-specific range. The resulting cut is a generalization of the combinatorial Benders' cut that forbids integer solutions in which every job $j\in J'$ starts in the (job-specific) range $[\widetilde{C_j} - p_j, \widetilde{C_j} - p_j]$ and every job $j\in J\setminus J'$ starts in the (job-specific) range $[r_j, t_{max} - p_j]$. We adapted the lifting procedure described in Côté et al. (2014) to our problem but did not observe significant improvements in terms of average computation time.

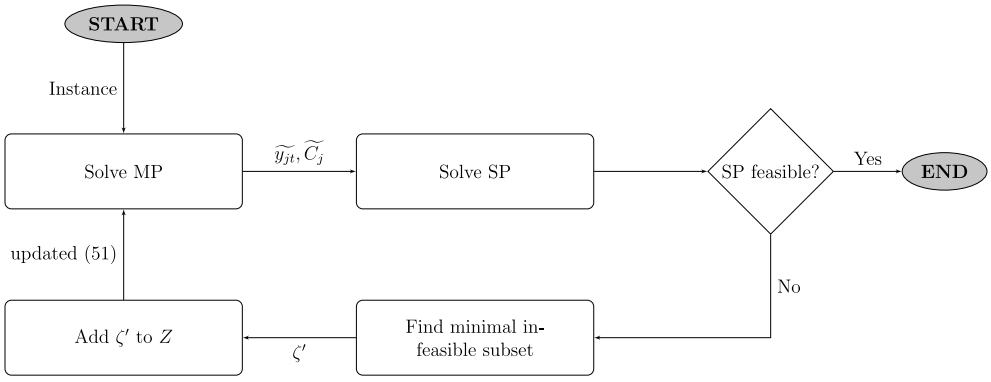


Fig. 2. Flowchart of the combinatorial Benders' decomposition approach.

5. Computational experiments

In this section, we study the empirical performance of each of the proposed algorithms on three sets of instances. All algorithms were coded in C++. The MILP models were solved with Gurobi 9.5.1 while the CP model was solved with IBM ILOG CPLEX CP Optimizer 22.1.0 (as Gurobi does not offer a CP solver). The tests were executed on a single thread of a virtual machine Intel(R) Xeon(R) Gold with 2.30 GHz and 20 GB of RAM memory, running under Windows 10 Pro N. We fed each approach with a warm start obtained by running the CP model until a feasible solution was found.

We first investigate the impact of key instance features such as the presence of precedence/contiguity constraints, the worker/machine/job ratios, and the probability for worker/ machine/job to be compatible. We then describe a case study originating from Dana Inc. and outline the potential gain that could be reached using optimization models instead of handmade solutions. We conclude by studying the scaling potential of our algorithms using large size instances designed to mimic the case study. All instances can be downloaded from the online repository https://github.com/regor-unimore/Parallel-Machine-Scheduling-with-Contiguity.

5.1. Experiment on random instances

To test the impact of various instance features, we generated a set of random instances where some parameters are randomly created while others are taken from the case study. The features that were investigated are:

1. number of jobs, machines, and workers: we tested 9 combinations $(|J|, |M|, |K|) \in \{(50,2,2), (50,5,3), (50,5,5), (100,4,4), (100,10,5), (100,10,10), (200,8,8), (200,20,10), (200,20,20)\}$;
2. number of precedence and contiguity relations: we tested two configurations, one without any precedence/contiguity constraints at all, and another one with a number of precedence/contiguity constraints randomly distributed in the range $[0, 0.05|J|+1]$ for the precedence constraints and $[0, 0.35|J|+1]$ for the contiguity constraints. Once the numbers of constraints were determined, we randomly selected pairs of successive jobs $(j, j+1)$ and added them to either P or Q. The resulting chain structure for the precedence/contiguity relations imitates the project structure of the case study. Instances without precedence and contiguity relations were tested in order to serve as a "control group" and determine whether an approach is better than another because it is more effective at handling precedence and contiguity constraints or because it is more effective overall;
3. job/machine, job/worker, and machine/worker eligibility ratios: we tested five ratios $r \in \{1, 0.9, 0.8, 0.7, 0.6\}$ and randomly selected $r \times |M|$ compatible machines and $r \times |K|$ compatible workers for every job. We also randomly selected $r \times |K|$ compatible workers per machine. A post-processing step changed the eligibilities of the jobs involved in the same chain of contiguities so that every job in the chain has the same job/machine and job/worker eligibilities as the first job of the chain.

One instance was generated for each of the $9 \times 2 \times 5 = 90$ combinations. The other features are:

1. the time horizon was fixed at $t_{max} = 400$;
2. every worker is available 8 h per day during the week and is not available during the weekend;
3. the release date r_j of a job j was randomly selected in the range $[0, 230]$. A post-processing step changed the release dates of the jobs involved in the same chain of precedence/contiguity relations so that every job in the chain has the same release date, which is set to the earliest release date r_{min} among all the jobs in the chain;
4. the daily resource consumption q_j of a job j is either equal to 1, with probability 0.2, or 8 with probability 0.8. In other words, a job needs a worker either punctually (we call such a job "passive") or for its whole duration (we call such a job "active");
5. the processing time p_j of every job j is a value randomly selected in the range $[1, 10]$ for active jobs and $[10, 50]$ for passive jobs;
6. the due date of every job d_j was set to $r_j + \xi p_j$, where ξ is an integer value randomly selected in the range $[1, 3]$. A post-processing step changed the due dates of the jobs involved in the same chain of precedence/contiguity relations so that every job in the chain has the same due date. This was set to $r_{min} + \xi \sum p_j$, where ξ is an integer value randomly selected in the range $[1, 2]$. Note that due dates above the time horizon were set to t_{max};
7. the weight of every job w_j is equal to 4 with probability 0.4, to 3 with probability 0.3, to 2 with probability 0.2, and to 1 with probability 0.1.

We report in Tables 3–6 a summary of the results obtained by our algorithms on the random instances. In each table, the first set of columns indicates the instance features, while the following columns report, for each algorithm, the number of optimal solutions found within the time limit ("# opt") , the average CPU time in seconds ("T(s)") including the instances that were not solved to optimality and the instances that were stopped because of the memory limit (for such instances, 3600 s of running time were considered), and the average CPU time in seconds computed only on the instances solved to optimality ("T_{opt}(s)"). In the tables and thereafter, "MILP" refers to the MILP model (1)–(12), "MILP+" refers to the enhanced MILP model (13)–(29), "CP" refers to model (30)–(43), and "Decomp" refers to the combinatorial Benders' decomposition algorithm in Section 4.3. We also report in columns "OBA" ("Only Best Algorithm") the results

Table 3
Summary of results: impact of $(|J|, |M|, |K|)$ combinations.

Parameters				Results													
$\|J\|$	$\|M\|$	$\|K\|$	#	MILP			MILP+			CP			Decomp			OBA		
				# opt	T(s)	T_{opt}(s)	# opt	$T(s)$	T_{opt}(s)	# opt	T(s)	T_{opt}(s)	# opt	T(s)	T_{opt}(s)	# opt	T(s)	T_{opt}(s)
50	2	2	10	8	966	307	9	762	446	5	2233	867	6	1456	27	9	546	207
50	5	3	10	10	129	129	10	124	124	10	2	2	7	1089	12	10	2	2
50	5	5	10	10	102	102	10	57	57	9	360	0	10	3	3	10	1	1
100	4	4	10	4	2372	528	5(1)	2342	1083	1	3259	188	4(3)	2314	385	8	1173	566
100	10	5	10	10	774	774	9	570	234	10	2	2	9	385	28	10	2	2
100	10	10	10	8	889	211	10	145	145	10	1	1	10	4	4	10	1	1
200	8	8	10	2	3289	2044	2	3159	1395	1	3249	94	5(2)	2078	557	5	2078	557
200	20	10	10	3	2718	659	7	1239	227	10	8	8	10	25	25	10	7	7
200	20	20	10	2	2936	279	9	824	516	10	8	8	10	9	9	10	6	6
Tot			90	57	1575	402	71(1)	1025	336	66	1014	73	71(5)	818	74	82	424	114

Table 4
Summary of results: comparison between MILP and MILP+.

Parameters				Results													
$\|J\|$	$\|M\|$	$\|K\|$	#	MILP							MILP+						
				# opt	T(s)	T_{opt}(s)	LP	# var.	# cons.	# nzs.	# opt	T(s)	T_{opt}(s)	LP	# var.	# cons.	# nzs.
50	2	2	10	8	966	307	692.20	39,106	5444	2,157,317	9	762	446	692.20	44,528	25,594	1,380,991
50	5	3	10	10	129	129	0.00	115,790	14,985	7,844,411	10	124	1234	0.00	89,742	35,285	4,019,729
50	5	5	10	10	102	102	0.00	192,341	13,780	11,707,342	10	57	57	0.00	112,252	34,080	3,960,135
100	4	4	10	4	2372	528	78.90	265,873	16,984	15,787,486	5	2342	1083	77.50	178,565	57,484	5,862,264
100	10	5	10	10	774	774	0.00	780,006	60,450	64,969,996	9	570	234	0.00	342,722	101,550	20,064,260
100	10	10	10	8	889	211	0.00	1,346,082	22,840	43,757,243	10	145	145	0.00	423,762	63,940	10,382,416
200	8	8	10	2	3289	2044	6.98	1,750,779	9675	37,842,773	2	3159	1395	6.98	673,733	91,475	11,047,170
200	20	10	10	3	2718	659	0.00	4,648,955	12,600	97,902,775	7	1239	227	0.00	1,193,206	96,800	21,548,621
200	20	20	10	2	2936	278	0.00	6,265,306	16,600	125,042,161	9	825	516	0.00	1,455,916	100,800	26,859,121
Tot			90	57	1575	402	86.45	1,711,582	19,262	45,223,501	71	1025	336	86.31	501,603	67,445	11,680,523

obtained by the best approach among MILP, MILP+, CP, and Decomp for every instance. In other words, OBA simulates the performance of a hyper-algorithm able to predict with 100% accuracy the algorithm that is the fastest to solve a given instance; it is thus of interest as a comparison term with respect to the performance of the previous algorithms. To identify the methods that contribute the most to the performance of OBA, we also include in the "# opt" columns (within round brackets) the number of instances that are solved to optimality by a given approach while being unsolved by the other three algorithms. If no number is included within brackets, then each of the instances in the group was solved to proven optimality by two or more algorithms.

We report in Table 3 the summary of results on the nine $(|J|, |M|, |K|)$ combinations. Overall, we observe that algorithms MILP+ and Decomp are both able to find an optimal solution for 71 out of 90 instances, compared with 66 for CP and 57 for MILP. The results obtained by the hypothetic OBA hyper-algorithm show that 82 instances could be solved to optimality (if we knew a priori which is the best algorithm for each instance). This indicates that the best algorithm is not always the same for all the instances: for example, five instances are solved by Decomp but not solved by MILP, MILP+, and CP; similarly, one instance is solved by MILP+, but not solved by MILP, CP, and Decomp. As far as instance combinations are concerned, MILP+ solves the most instances for combinations (50,2,2) and (100,4,4), Decomp solves the most instances for combination (200,8,8), and CP solves the most instances, together with MILP, for combination (100,10,5). We also point out that CP and Decomp results are very heterogeneous in the sense that the algorithms either solve an instance to optimality very fast or not at all, as witnessed by the low T_{opt}(s) values for the two algorithms compared with the values of MILP and MILP+. The most difficult instances seem to be the ones with 200 jobs, 8 machines, and 8 workers.

Our next set of experiments, which is reported in Table 4, aims at comparing the performance of MILP and MILP+. In the table, column "LP" indicates the average continuous relaxation value of the model and columns "# var.", "# cons.", and "# nzs." indicate the average number of variables, constraints, and non-zero elements in the coefficient matrix of the model, respectively. To provide meaningful comparisons, the averages were computed by only using the instances for which the LP relaxation value could be obtained by both models (i.e., the instances for which neither of the two algorithms ran out of memory). We observe that, overall, MILP+ is faster on average (336 s to solve an instance on average compared with 402 s for MILP) and solves more instances (71 instances solved in total compared with 57 for MILP). These results can mostly be explained by the reduced size of the model (501,603 variables on average for MILP+ compared with 1,711,582 for MILP) at the expense of a negligible decrease in the quality of the continuous linear relaxation.

We report in Table 5 the impact of the job/machine/worker eligibility ratios ("r") and the impact of precedence/contiguity relations ("P/Q") on the performance of our methods. Following the aforementioned description, an instance in which the eligibility ratio "r" is 0.9 has (i) each of its jobs compatible with 90% of the machines, (ii) each of its jobs compatible with 90% of the workers, and (iii) each of its machines compatible with 90% of the workers. Overall, it appears that instances with lower eligibility ratios are harder to solve, and so are instances with precedence/contiguity constraints. A detailed analysis shows that Decomp obtains the best results for instances with eligibility ratio above 0.9, CP shines on instances with precedence/contiguity constraints and eligibility ratio below 0.7, and MILP+ is working well in the remaining cases.

We report in Table 6 the impact of other instance features (in column "Par") on the performance of our methods such as the number of jobs ($|J|$) and the ratios job/machine, job/worker, and machine/worker

Table 5
Summary of results: impact of eligibility percentage and precedence/contiguity relations.

Parameters			Results														
r	P/Q	#	MILP			MILP+			CP			Decomp			OBA		
			# opt	T(s)	T_{opt}(s)	# opt	T(s)	T_{opt}(s)	# opt	T(s)	T_{opt}(s)	# opt	T(s)	T_{opt}(s)	# opt	T(s)	T_{opt}(s)
1	No	9	6	1339	208	9	150	150	7	803	4	9	5	5	9	5	5
	Yes	9	3	2431	93	6	1408	312	6	1205	8	8(1)	577	200	8	575	197
0.9	No	9	6	1264	96	7	815	20	7	891	117	9(2)	9	9	9	8	8
	Yes	9	4	2326	733	7	1196	509	6	1204	6	9(2)	306	306	9	292	292
0.8	No	9	6	1414	321	9(1)	480	480	8	689	325	8	407	7	9	219	219
	Yes	9	6	1534	501	7	1084	365	6	1224	35	6	1210	15	8	429	32
0.7	No	9	9	492	492	8	560	180	7	864	82	7	824	31	9	89	89
	Yes	9	5	2038	789	6	1828	942	6	1203	5	5	1608	14	7	991	245
0.6	No	9	8	563	183	8	567	187	7	851	66	6	1218	27	8	430	33
	Yes	9	4	2347	781	4	2159	358	6	1203	4	4	2019	43	6	1203	4
Tot		90	57	1575	402	71(1)	1025	336	66	1014	73	71(5)	818	74	82	424	114

Table 6
Summary of results: impact of size and worker/machine/job ratios.

Parameters			Results														
Par.	Value	#	MILP			MILP+			CP			Decomp			OBA		
			# opt	T(s)	T_{opt}(s)	# opt	T(s)	T_{opt}(s)	# opt	T(s)	T_{opt}(s)	# opt	T(s)	T_{opt}(s)	# opt	T(s)	T_{opt}(s)
$\|J\|$	50	30	28	399	170	29	314	201	24	865	182	23	849	12	29	183	65
	100	30	22	1345	525	24(1)	1019	374	21	1087	10	23(3)	901	80	28	392	163
	200	30	7	2981	946	18	1741	501	21	1088	12	25(2)	704	125	25	697	116
$\frac{\|J\|}{\|M\|}$	25	30	14	2209	618	16(1)	2087	764	7	2914	659	15(5)	1950	299	22	1266	417
	10	60	43	1258	332	55	493	211	59	63	4	56	253	13	60	3	3
$\frac{\|J\|}{\|K\|}$	25	30	14	2209	618	16(1)	2087	764	7	2914	659	15(5)	1950	299	22	1265	417
	20	30	23	1207	479	26	644	190	30	4	4	26	450	23	30	4	4
	10	30	20	1309	163	29	342	230	29	123	3	30	5	5	30	3	3
$\frac{\|M\|}{\|K\|}$	2	30	23	1207	479	26	644	190	30	4	4	26	500	23	30	4	4
	1	60	34	1759	351	45(1)	1215	420	36	1518	131	45(5)	978	103	52	634	178

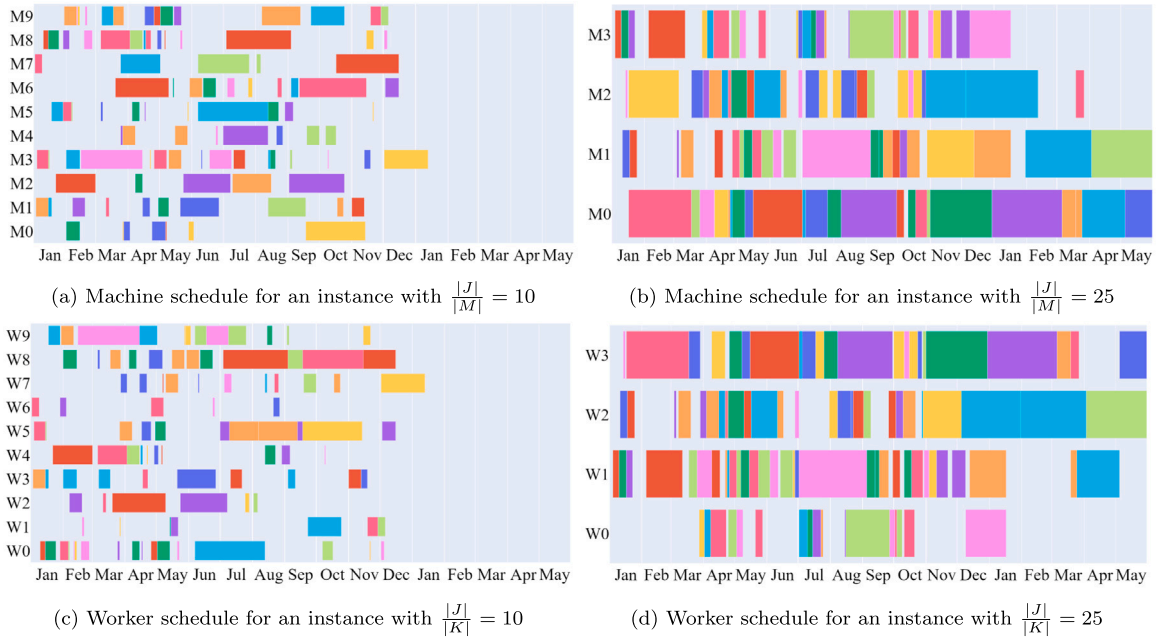
($\frac{|J|}{|M|}$, $\frac{|J|}{|K|}$, and $\frac{|M|}{|K|}$, respectively). As expected, instances with 200 jobs tend to be harder to solve than instances with 50 jobs, especially for the two MILP models.

We also notice that instances are easier to solve when the ratios $\frac{|J|}{|M|}$ and $\frac{|J|}{|K|}$ decrease. This can be explained by the fact that the optimal schedule for instances with high $\frac{|J|}{|M|}$ and $\frac{|J|}{|K|}$ ratios tend to be very busy compared with the optimal schedule of instances with lower ratios. This can be observed in Fig. 3, which reports the best machine schedule (Figs. 3(a)–3(b)) and the best worker schedule (Figs. 3(c)–3(d)) for two instances, one with $\frac{|J|}{|M|} = \frac{|J|}{|K|} = 10$ (on the left) solved in 0.23 s by CP (214.45 s for MILP+), and the other with $\frac{|J|}{|M|} = \frac{|J|}{|K|} = 25$ unsolved after 3600 s by CP and MILP+. In the two top figures, a rectangle represents the machine occupation for a job. In the two bottom figures, a rectangle represents the worker occupation for at least a job (since a worker may perform either one active job or up to eight passive jobs per day). As far as the ratio $\frac{|M|}{|K|}$ is concerned, we observe that instances with twice as many machines as workers are easier to solve than instances with the same number of machines and workers. This can be explained by our instance generation procedure, since each of the 30 instances in which the ratio $\frac{|M|}{|K|}$ is equal to 2 has a counterpart (i.e., an instance with the same numbers of jobs and machines) in which the ratio is equal to 1 (i.e., with twice as many workers). In other words, supplementary workers increase the instance difficulty (all other parameters being equal). If we focus on the individual performance of each algorithm, we notice that (i) MILP+ is the most efficient approach in practice for instances with 50 and 100 jobs while instances with 200 jobs are solved best with Decomp; (ii) CP is particularly effective on instances where the ratio $\frac{|J|}{|M|}$ is equal to 10, while MILP+ obtains the best performance on instances where that ratio is equal to 25; (iii) Decomp obtains the best results on instances where the ratio $\frac{|J|}{|K|}$ is equal to 10, CP is the best when that ratio is equal to 20, and MILP+ is the best when that ratio is equal to 25; and (iv) CP performs best on instances where the ratio $\frac{|M|}{|K|}$ is equal to 2, while Decomp obtains the best results for instances where this ratio is equal to 1. Once more, an analysis of T_{opt}(s) shows that, when CP and Decomp solve an instance to optimality, they tend to do so relatively fast, whereas MILP and MILP+ require a few hundred seconds on average. Overall, we observe that each algorithm is effective on a subset of instances.

5.2. Case study

We now test our approaches on the case study instance provided by Dana Inc. A *project* is defined as a set of one or more *tests* (i.e., jobs) to be executed on the same product. Each project has a release date, a desired due date, and a *priority* (i.e., a weight). Therefore, all jobs belonging to a given project have the same release date, the same due date, and the same weight, and are linked by either a precedence or a contiguity relation.

To transform the original data into a valid instance, we took into account all the jobs taking place in calendar year 2019. We selected the jobs whose release date was between day 01 (January 01) and day 240 (early December – note that Saturdays and Sundays are not counted) and whose due date was before day 400 (mid-July 2020), obtaining an instance with 49 projects, 86 jobs (with an average duration of 10 days), 6 precedence relations, 27 contiguity relations, and a time



(a) Machine schedule for an instance with $\frac{|J|}{|M|} = 10$

(b) Machine schedule for an instance with $\frac{|J|}{|M|} = 25$

(c) Worker schedule for an instance with $\frac{|J|}{|K|} = 10$

(d) Worker schedule for an instance with $\frac{|J|}{|K|} = 25$

Fig. 3. Graphical representation of the optimal schedule for 2 instances with different $\frac{|J|}{|M|}$ and $\frac{|J|}{|K|}$ ratios.

horizon of 400 days. In the real-world application, there were also some tasks that were released in year 2018 and scheduled for early 2019. Based on the discussions with the company and the limited data we could access, we estimated that an additional 20% of jobs were in that situation. As we had no accurate information regarding the release date or due date of such tasks in the available dataset, we decided to take them into account by forbidding the jobs to be scheduled during the first 2 months and a half of the year.

In the resulting instance, machines are considered available all the time after 2.5-months. Workers are active at most eight hours per working day and are all available eight hours per day except during the weekends, the bank holidays, and the four annual weeks of holidays in which the company is closed. Tasks can be interrupted during the weekend (i.e., a three-day job may start on a Friday morning and finish on the following Tuesday evening) but not during the annual holidays (i.e., every job that was started needs to be finished before the first day of holidays). As explained in the previous section, a job is either active (i.e., it necessitates the constant supervision of a worker) and requires eight worker hours per day or it is passive (i.e., it only involves a punctual supervision) and requires one worker hour per day. Eligibility relations were given by the company and are based on machine features and worker skills. On average, a job is compatible with 14% of machines and 38% of workers, while a machine is compatible with 29% of workers.

MILP+ was able to find an optimal solution in 572 s. Fig. 4 shows the manual schedule used by the company (on the left) and the optimal schedule found by MILP+ (on the right). We obtained an overall decrease of 73% in the total weighted tardiness, with 27 late jobs against 44 jobs in delay in the manual solution. This outstanding result can be explained by two reasons. First, manual solutions are often sub-optimal, even when they are provided by experienced workers (see, e.g., Seifi et al., 2021; Baykasoğlu and Özbel, 2021). Second, there are some real-world aspects that could not be captured by our model, such as machine breakdown/maintenance, worker-specific unavailability (e.g., due to illness or extra holidays), stochasticity of the job duration due to unexpected delay of product delivery, among others. In summary, it is very unlikely that the best schedule for Dana Inc. can be determined solely with our optimization tools, but the solutions found can serve as a basis to support the company experts in producing the final schedules.

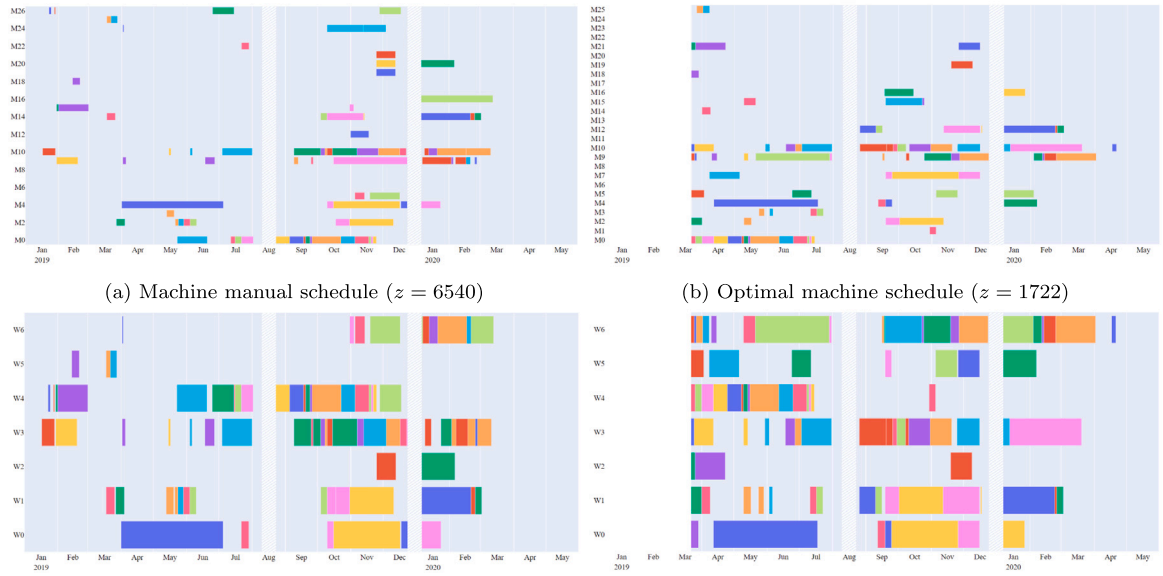
5.3. Experiment on realistic instances

To test the scaling potential of our algorithms, we generated a set of realistic instances that include the real-world features of the case study. In particular,

1. the number of projects is set to 40, 80, 120, 160, or 200. For each project, the number of jobs is equal to 1 with probability 0.7, 2 with probability 0.2 (linked by a precedence constraint with probability 0.4 or a contiguity constraint with probability 0.6), or is a random number between 3 and 15 with probability 0.1 (always linked by contiguity constraints);
2. every job is considered active (requiring eight worker-hours per day) except the second job of the projects that contains two tasks linked by a precedence constraint;
3. the job/machine, job/worker, and machine/worker eligibility ratios are set to the realistic values (14%, 38%, and 29%, respectively). We enforced consistency in the eligibility matrices by imposing that every worker that is compatible with job i must also be compatible with at least one machine that is compatible with i;
4. 20% supplementary late projects are included to model all jobs from the previous year. All tasks in these projects have their release date and due date set to the first time period.

We report in Table 7 a summary of the results obtained by our algorithms on the realistic instances. Table 7 has the same structure as the other tables except for the first column which now indicates the number of projects contained in the instance.

As expected, larger instances (with 120/160/200 projects) are harder to solve, even though we observe that a few instances with 200 projects (4 times as large as the case study) can be solved to optimality. MILP and MILP+ appear to be the weakest approach es on this dataset as they are only able to solve 15 and 21 instances in total, respectively. CP performs slightly better than MILP+ as it is able to solve 22 instances in less than one minute on average (compared with 17 min on average for MILP+, if we exclude unsolved instances). Interestingly, we observe that CP is often able to find good-quality (or even optimal) solutions, even for the largest instances, but it struggles in finding good-quality lower bounds. Decomp is the



(a) Machine manual schedule ($z = 6540$)

(b) Optimal machine schedule ($z = 1722$)

(c) Workers' manual schedule ($z = 6540$)

(d) Optimal workers' schedule ($z = 1722$)

Fig. 4. Graphical representation of solutions for the Dana case study (summer and winter holidays are identified by two white rectangles).

Table 7
Summary of results: impact of size on realistic instances.

Parameters		Results													
Size	#	MILP			MILP+			CP			Decomp			OBA		
		# opt	T(s)	T_{opt}(s)	# opt	T(s)	T_{opt}(s)	# opt	T(s)	T_{opt}(s)	# opt	T(s)	T_{opt}(s)	# opt	T(s)	T_{opt}(s)
40	10	10	526	526	10	332	332	10	61	61	10	16	16	10	10	10
80	10	5	2944	2288	9	1681	1467	3	2521	2	10	108	108	10	42	42
120	10	0	3600	–	2	3309	2143	4	2163	6	10(4)	799	799	10	284	284
160	10	0	3600	–	0	3600	–	4(2)	2172	31	8(6)	1887	1459	10	784	784
200	10	0	3600	–	0	3600	–	1(1)	3248	82	3(3)	3297	2591	4	2946	1964
Tot	50	15	2854	1113	21	2504	991	22(3)	2033	38	41(13)	1222	699	44	813	433

most effective algorithm in terms of the number of instances solved to proven optimality (41 out of 50, among which 13 are unsolved by the other three methods, compared with 22 for CP, among which 3 are unsolved by the other three methods). However, an analysis of T_{opt}(s) shows that Decomp may take more than a thousand seconds to solve an instance whereas CP solves an instance to optimality in less than one hundred seconds. We also emphasize that if one could pre-select the best algorithm for each instance, then the total number of instances solved to optimality would have increased by 3 compared with Decomp, while simultaneously reducing the average computation time of the solved instances by 266 s.

We also tested a version of MILP and MILP+ in which `Cplex 22.1.0.0` was used instead of `Gurobi 9.5.1` as solver. The results, reported in Table 8, show that both models solved more instances to optimality with Gurobi than they did with Cplex. While one should be cautious before claiming an empirical advantage of one solver over another (see, e.g., Mittelmann, 2020), such a comparison indicates that the poor performance of the two MILP models on this dataset is not caused by the choice of the solver.

Our next set of experiments, which is reported in Table 9, aims at comparing the performance of MILP and MILP+ on the realistic instances. As MILP ran out of memory when solving the instances with 120 projects (which means that the LP relaxation value could not be computed or was not returned), we only compare the two models on instances with 40 and 80 projects. Once more, we observe that, overall, MILP+ is faster on average (1467 s to solve instances with 80 projects on average compared with 2288 s for MILP, if we exclude unsolved instances) and solves more instances (19 instances with 80 projects or less solved compared with 15 for MILP). Again, these results can mostly be explained by the reduced size of the model (683,197 variables on average for MILP+ on instances with 80 projects compared with 869,438 for MILP). This time, MILP and MILP+ had the same continuous linear relaxation value for all instances.

Our two final sets of experiments, which are reported in Tables 10 and 11, aim at evaluating whether our algorithms remain effective when solving related RCPMSP with contiguity constraints. We first tested our three best approaches MILP+, CP, and Decomp on the version of our problem where one wants to minimize the total weighted flow time, which is the same as minimizing the total weighted tardiness when the due date of every task is equal to its release date. Note that the total weighted flow time (a generalization of the total flow time, which is a common measure in system performance) is often used in the PMSP literature (see, e.g., Shabtay and Kaspi, 2004 and Becchetti et al., 2006). As shown in Table 10, changing the objective function from weighted tardiness minimization to weighted flow time minimization barely impacts the performance of Decomp and MILP+, whereas it has a negative effect on CP. This could be explained by the fact that the optimal weighted flow time is always strictly positive, meaning that a good quality lower bound (which can be difficult to obtain with CP) is necessary to provide a certificate of optimality.

We then tested the same three approaches on the version of our problem where one wants to minimize the maximum tardiness. As shown in Table 11, changing the objective function from weighted tardiness minimization to maximum tardiness minimization has a slight positive impact on the performance of Decomp and MILP+ and an outstanding positive effect on CP. This could be explained by the

Table 8
Summary of results: solvers comparison.

Parameters		Gurobi						Cplex					
Size	#	MILP			MILP+			MILP			MILP+		
		# opt	T(s)	T_{opt}(s)	# opt	T(s)	T_{opt}(s)	# opt	T(s)	T_{opt}(s)	# opt	T(s)	T_{opt}(s)
40	10	10	526	526	10	332	332	2	2905	123	3	2970	1501
80	10	5	2944	2288	9	1681	1467	0	3600	–	0	3600	–
120	10	0	3600	–	2	3309	2143	0	3600	–	0	3600	–
160	10	0	3600	–	0	3600	–	0	3600	–	0	3600	–
200	10	0	3600	–	0	3600	–	0	3600	–	0	3600	–
Tot	50	15	2854	1113	21	2504	991	2	3461	123	3	3474	1501

Table 9
Summary of results: comparison between MILP and MILP+.

Parameters		Results												
Size	#	MILP							MILP+						
		# opt	T(s)	T_{opt}(s)	LP	# var.	# cons.	# nzs.	# opt	T(s)	T_{opt}(s)	LP	# var.	# cons.	# nzs.
40	10	10	526	526	993.28	230,910	559,444	53,061,913	10	332	332	993.28	250,044	600,028	32,912,832
80	10	5	2944	2288	874.00	869,438	1,336,717	173,201,182	9	1681	1467	874.00	683,197	1,398,401	81,498,170

Table 10
Summary of results: impact of size on realistic instances with weighted flow time minimization.

Parameters		Results											
Size	#	MILP+			CP			Decomp			OBA		
		# opt	T(s)	T_{opt}(s)	# opt	T(s)	T_{opt}(s)	# opt	T(s)	T_{opt}(s)	# opt	T(s)	T_{opt}(s)
40	10	10	200	200	10	327	327	10	28	28	10	9	9
80	10	9	1029	744	2	2880	2	10	85	85	10	39	39
120	10	2	3063	1181	4	2162	6	10(4)	578	578	10	310	310
160	10	0	3600	–	2	2887	37	8(6)	1705	1231	8	1232	640
200	10	0	3600	–	1(1)	3248	81	4(4)	2691	1327	5	2339	1078
Tot	50	21	2283	526	19(1)	2301	182	42(14)	1018	526	43	786	328

Table 11
Summary of results: impact of size on realistic instances with maximum tardiness minimization.

Parameters		Results											
Size	#	MILP+			CP			Decomp			OBA		
		# opt	T(s)	T_{opt}(s)	# opt	T(s)	T_{opt}(s)	# opt	T(s)	T_{opt}(s)	# opt	T(s)	T_{opt}(s)
40	10	10	338	338	10	0	0	10	16	16	10	0	0
80	10	8	1646	1157	10	3	3	10	111	111	10	3	3
120	10	1	3346	1060	10	11	11	10	651	651	10	11	11
160	10	0	3600	–	10(1)	28	28	9	2047	1875	10	28	28
200	10	0	3600	–	10(6)	49	49	4	3283	2808	10	49	49
Tot	50	19	2506	721	50(7)	18	18	43	1222	835	50	18	18

fact that the optimal maximum tardiness is always a small number (sometimes even zero), meaning that being able to find a good quality upper bound (which is where CP shines) is more important to obtain a certificate of optimality.

6. Conclusions

We studied a new parallel machine scheduling problem motivated by the real-world case study of an hydraulic motion components engineering test laboratory. The problem contains release dates, due dates, weights, workforce resource with daily availability, three sets of eligibility constraints, precedence constraints, and contiguity constraints. The introduction of contiguity constraints makes our problem original in the machine scheduling literature. We proposed two original MILP models, a CP formulation, and a combinatorial Benders' decomposition. We also introduced method-specific strategies to model the contiguity constraints for each of the proposed approaches. We tested the effectiveness of the four algorithms with an extensive set of computational experiments, which included random, realistic, and real instances. Our best algorithm, the decomposition approach, was able to solve 71 out of 90 random instances and 41 out of 50 realistic instances to optimality in one hour of computing time. However, we empirically showed that this algorithm does not clearly dominate the others because (i) both the CP formulation and the MILP formulations were able to solve instances that the decomposition approach could not solve and (ii) the CP formulation was able to solve some large realistic instances much faster than the decomposition approach. We also showed that our approaches could easily be extended to take other objective functions into account. We also identified some features that make an instance more difficult to solve (e.g., the presence of precedence/contiguity constraints, and high job/machine and job/worker ratios). We were also able to solve a real instance to optimality with a significant improvement on the total weighted tardiness with respect to the manual solution produced by the company. However, there are a few aspects that we could not take into account in our problem modelization such as machine unavailability caused by a breakdown or by a maintenance intervention, worker unavailability due to illness or supplementary holidays, and the job release date and duration stochasticity caused by unexpected events such as a delay in the product delivery or a machine breakdown happening while the job is being processed.

Future research directions include (i) the development of a hyper-algorithm able to predict and select the most effective method to solve a given instance with supervised learning techniques (i.e., an algorithm able to replicate the results of OBA) and (ii) the study of the dynamic version of the problem to include a rolling horizon aspect where the problem is solved at regular time intervals and where, in each run, a set of job/machine and job/worker assignments is fixed (because it was determined in the previous run), a set of job/machine and job/worker assignments is planned (because it was determined in the previous run but can still be modified if necessary), and a set of job/machine and job/worker assignments is free (because the projects were released after the last run took place or were delayed and need to be rescheduled); the objective function would then be hybridized to also include a minimization component for disruption (number of planned assignments that are modified). While Dana Inc. is interested in the one-year-ahead solutions obtained by our approaches for scheduling and prevision purposes, the company also needs to have the tools to regularly update those schedules and predictions later on when new data are available.

CRediT authorship contribution statement

Giulia Caselli: Conceptualization, Methodology, Software, Validation, Investigation, Data curation, Writing – original draft, Visualization. **Maxence Delorme:** Conceptualization, Methodology, Writing – review & editing, Visualization, Supervision. **Manuel Iori:** Conceptualization, Resources, Writing – review & editing, Supervision. **Carlo Alberto Magni:** Writing – review & editing, Supervision.

Data availability

Data can be downloaded from the online repository https://github.com/regor-unimore/Parallel-Machine-Scheduling-with-Contiguity.

Acknowledgments

We would like to thank the three anonymous reviewers for their valuable comments, which have helped improve the presentation of this paper. We thank Andrea Lucchi from Dana Inc. for his support during this project.

References

Afzalirad, M., Rezaeian, J., 2016. Resource-constrained unrelated parallel machine scheduling problem with sequence dependent setup times, precedence constraints and machine eligibility restrictions. Comput. Ind. Eng. 98, 40–52.
Baker, K.R., Trietsch, D., 2019. Principles of Sequencing and Scheduling. John Wiley & Sons, New York.
Baykasoğlu, A., Özbel, B.K., 2021. Modeling and solving a real-world cutting stock problem in the marble industry via mathematical programming and stochastic diffusion search approaches. Comput. Oper. Res. 128, 105173.
Becchetti, L., Leonardi, S., Marchetti-Spaccamela, A., Pruhs, K., 2006. Online weighted flow time and deadline scheduling. J. Discrete Algorithms 4 (3), 339–352.
Becker, C., Scholl, A., 2006. A survey on problems and methods in generalized assembly line balancing. European J. Oper. Res. 168 (3), 694–715.
Benders, J.F., 1962. Partitioning procedures for solving mixed-variables programming problems. Numer. Math. 4 (1), 238–252.
Blazewicz, J., Lenstra, J.K., Rinnooy Kan, A.H.G., 1983. Scheduling subject to resource constraints: classification and complexity. Discrete Appl. Math. 5 (1), 11–24.
Bockmayr, A., Hooker, J.N., 2005. Constraint programming. Handbooks Oper. Res. Management Sci. 12, 559–600.
Bruno, J., Downey, P., 1978. Complexity of task sequencing with deadlines, set-up times and changeover costs. SIAM J. Comput. 7 (4), 393–404.
Caselli, G., Delorme, M., Iori, M., Magni, C.A., 2022. Mixed integer linear programming for a real-world parallel machine scheduling problem with workforce and precedence constraints. In: Amorosi, L., Dell'Olmo, P., Lari, I. (Eds.), Optimization in Artificial Intelligence and Data Sciences. Springer International Publishing, Cham, pp. 61–71.
Chen, Z.-L., 2004. Simultaneous job scheduling and resource allocation on parallel machines. Ann. Oper. Res. 129 (1), 135–153.
Cheng, T.C.E., Sin, C.C.S., 1990. A state-of-the-art review of parallel-machine scheduling research. European J. Oper. Res. 47 (3), 271–292.
Chudak, F.A., Shmoys, D.B., 1999. Approximation algorithms for precedence-constrained scheduling problems on parallel machines that run at different speeds. J. Algorithms 30 (2), 323–343.
Codato, G., Fischetti, M., 2006. Combinatorial Benders' cuts for mixed-integer linear programming. Oper. Res. 54 (4), 756–766.
Côté, J.-F., Dell'Amico, M., Iori, M., 2014. Combinatorial Benders' cuts for the strip packing problem. Oper. Res. 62 (3), 643–661.
Daniels, R.L., Hoopes, B.J., Mazzola, J.B., 1997. An analysis of heuristics for the parallel-machine flexible-resource scheduling problem. Ann. Oper. Res. 70, 439–472.
Dell'Amico, M., Delorme, M., Iori, M., Martello, S., 2019. Mathematical models and decomposition methods for the multiple knapsack problem. European J. Oper. Res. 274 (3), 886–899.
Delorme, M., Iori, M., Martello, S., 2017. Logic based Benders' decomposition for orthogonal stock cutting problems. Comput. Oper. Res. 78, 290–298.
Delorme, M., Santini, A., 2022. Energy-efficient automated vertical farms. Omega 109, 102611.
Edis, E.B., Oguz, C., Ozkarahan, I., 2013. Parallel machine scheduling with additional resources: Notation, classification, models and solution methods. European J. Oper. Res. 230 (3), 449–463.
Edis, E.B., Ozkarahan, I., 2011. A combined integer/constraint programming approach to a resource-constrained parallel machine scheduling problem with machine eligibility restrictions. Eng. Optim. 43 (2), 135–157.
Edis, E.B., Ozkarahan, I., 2012. Solution approaches for a real-life resource-constrained parallel machine scheduling problem. Int. J. Adv. Manuf. Technol. 58 (9), 1141–1153.
Fang, K., Wang, S., Pinedo, M.L., Chen, L., Chu, F., 2021. A combinatorial Benders decomposition algorithm for parallel machine scheduling with working-time restrictions. European J. Oper. Res. 291 (1), 128–146.
Fleszar, K., Hindi, K.S., 2018. Algorithms for the unrelated parallel machine scheduling problem with a resource constraint. European J. Oper. Res. 271 (3), 839–848.
Fuchigami, H.Y., Rangel, S., 2018. A survey of case studies in production scheduling: Analysis and perspectives. J. Comput. Sci. 25, 425–436.
Gedik, R., Rainwater, C., Nachtmann, H., Pohl, E.A., 2016. Analysis of a parallel machine scheduling problem with sequence dependent setup times and job availability intervals. European J. Oper. Res. 251 (2), 640–650.
Geoffrion, A.M., 1972. Generalized Benders decomposition. J. Optim. Theory Appl. 10 (4), 237–260.
Gökgür, B., Hnich, B., Özpeynirci, S., 2018. Parallel machine scheduling with tool loading: a constraint programming approach. Int. J. Prod. Res. 56 (16), 5541–5557.
Graham, R.L., Lawler, E.L., Lenstra, J.K., Rinnooy Kan, A.H.G., 1979. Optimization and approximation in deterministic sequencing and scheduling: a survey. In: Annals of Discrete Mathematics, Vol. 5. Elsevier, pp. 287–326.
Hall, N.G., Potts, C.N., Sriskandarajah, C., 2000. Parallel machine scheduling with a common server. Discrete Appl. Math. 102 (3), 223–243.
Hartmann, S., Briskorn, D., 2022. An updated survey of variants and extensions of the resource-constrained project scheduling problem. European J. Oper. Res. 297 (1), 1–14.
Hooker, J.N., 2006. An integrated method for planning and scheduling to minimize tardiness. Constraints 11 (2), 139–157.
Hooker, J.N., 2011. Logic-Based Methods for Optimization: Combining Optimization and Constraint Satisfaction. John Wiley & Sons, New York.
Hooker, J.N., Ottosson, G., 2003. Logic-based Benders decomposition. Math. Program. 96 (1), 33–60.
Hu, X., Bao, J.-S., Jin, Y., 2010. Minimising makespan on parallel machines with precedence constraints and machine eligibility restrictions. Int. J. Prod. Res. 48 (6), 1639–1651.
Jain, V., Grossmann, I.E., 2001. Algorithms for hybrid MILP/CP models for a class of optimization problems. INFORMS J. Comput. 13 (4), 258–276.
Janiak, A., Janiak, W.A., Krysiak, T., Kwiatkowski, T., 2015. A survey on scheduling problems with due windows. European J. Oper. Res. 242 (2), 347–357.
Karlsson, E., Rönnberg, E., 2022. Logic-based Benders decomposition with a partial assignment acceleration technique for avionics scheduling. Comput. Oper. Res. 146, 105916.
Khatami, M., Salehipour, A., 2021a. A binary search algorithm for the general coupled task scheduling problem. 4OR 19 (4), 593–611.
Khatami, M., Salehipour, A., 2021b. Coupled task scheduling with time-dependent processing times. J. Sched. 24 (2), 223–236.
Khatami, M., Salehipour, A., Cheng, T.C.E., 2020. Coupled task scheduling with exact delays: Literature review and models. European J. Oper. Res. 282 (1), 19–39.
Koné, O., Artigues, C., Lopez, P., Mongeau, M., 2011. Event-based MILP models for resource-constrained project scheduling problems. Comput. Oper. Res. 38 (1), 3–13.
Koulamas, C., 1994. The total tardiness problem: review and extensions. Oper. Res. 42 (6), 1025–1041.
Kramer, R., Dell'Amico, M., Iori, M., 2017. A batching-move iterated local search algorithm for the bin packing problem with generalized precedence constraints. Int. J. Prod. Res. 55 (21), 6288–6304.

Kramer, A., Dell'Amico, M., Iori, M., 2019. Enhanced arc-flow formulations to minimize weighted completion time on identical parallel machines. European J. Oper. Res. 275 (1), 67–79.

Lawler, E.L., Lenstra, J.K., Rinnooy Kan, A.H.G., Shmoys, D.B., 1993. Sequencing and scheduling: Algorithms and complexity. In: Graves, S., Rinnooy Kan, A., Zipkin, P. (Eds.), Logistics of Production and Inventory. Elsevier, Amsterdam, pp. 445–522.

Lee, C.Y., 2004. Machine scheduling with availability constraints. In: Leung, J.Y.T. (Ed.), Handbook of Scheduling: Algorithms, Models, and Performance Analysis. CRC Press, Boca Raton, pp. 22–1–22–14.

Lenstra, J.K., Rinnooy Kan, A.H.G., 1978. Complexity of scheduling under precedence constraints. Oper. Res. 26 (1), 22–35.

Leung, J.Y.-T., Li, C.-L., 2016. Scheduling with processing set restrictions: A literature update. Int. J. Prod. Econ. 175, 1–11.

Li, Y., Côté, J.-F., Coelho, L.C., Wu, P., 2022. Novel efficient formulation and matheuristic for large-sized unrelated parallel machine scheduling with release dates. Int. J. Prod. Res. 60 (20), 6104–6123.

Lovato, D., Guillaume, R., Thierry, C., Battaia, O., 2023. Managing disruptions in aircraft assembly lines with staircase criteria. Int. J. Prod. Res. 61 (2), 632–648.

Mischek, F., Musliu, N., 2021. A local search framework for industrial test laboratory scheduling. Ann. Oper. Res. 302 (2), 533–562.

Mittelmann, H.D., 2020. Benchmarking optimization software-a (hi) story. In: SN Operations Research Forum, Vol. 1. Springer, p. 2.

Mokotoff, E., 2001. Parallel machine scheduling problems: A survey. Asia-Pac. J. Oper. Res. 18 (2), 193–242.

Pinedo, M.L., 2016. Scheduling - Theory, Algorithms, and Systems. Springer International Publishing.

Potts, C.N., Kovalyov, M.Y., 2000. Scheduling with batching: A review. European J. Oper. Res. 120 (2), 228–249.

Pritsker, A.A.B., Waiters, L.J., Wolfe, P.M., 1969. Multiproject scheduling with limited resources: A zero-one programming approach. Manag. Sci. 16 (1), 93–108.

Reddy, N.S., Ramamurthy, D.V., Padma Lalitha, M., Prahlada Rao, K., 2022. Integrated simultaneous scheduling of machines, automated guided vehicles and tools in multi machine flexible manufacturing system using symbiotic organisms search algorithm. J. Ind. Prod. Eng. 39 (4), 317–339.

Seifi, C., Schulze, M., Zimmermann, J., 2021. A new mathematical formulation for a potash-mine shift scheduling problem with a simultaneous assignment of machines and workers. European J. Oper. Res. 292 (1), 27–42.

Seo, K., Joung, S., Lee, C., Park, S., 2022. A closest Benders cut selection scheme for accelerating the Benders decomposition algorithm. INFORMS J. Comput. 34 (5), 2804–2827.

Shabtay, D., Kaspi, M., 2004. Minimizing the total weighted flow time in a single machine with controllable processing times. Comput. Oper. Res. 31 (13), 2279–2289.

Su, H., Pinedo, M.L., Wan, G., 2017. Parallel machine scheduling with eligibility constraints: A composite dispatching rule to minimize total weighted tardiness. Nav. Res. Logist. 64 (3), 249–267.

Ulusoy, G., Sivrikaya-Şerifoğlu, F., Bilge, Ü., 1997. A genetic algorithm approach to the simultaneous scheduling of machines and automated guided vehicles. Comput. Oper. Res. 24 (4), 335–351.

Ventura, J.A., Kim, D., 2000. Parallel machine scheduling about an unrestricted due date and additional resource constraints. IIE Trans. 32 (2), 147–153.

Węglarz, J., Józefowska, J., Mika, M., Waligóra, G., 2011. Project scheduling with finite or infinite number of activity processing modes–a survey. Eur. J. Oper. Res. 208 (3), 177–205.