Models and algorithms for configuring and testing prototype cars

François Clautiaux, Siham Essodaigui, Alain Nguyen, Ruslan Sadykov, Nawel Younes

Models and algorithms for configuring and testing prototype cars

François Clautiaux[a,*], Siham Essodaigui[b], Alain Nguyen[b], Ruslan Sadykov[a], Nawel Younes[b]

[a] *Université de Bordeaux, CNRS, INRIA, Bordeaux INP, IMB, UMR 5251, F-33400 Talence, France*
[b] *Renault S.A., F-92100 Boulogne-Billancourt, France.*

Abstract

In this paper, we consider a new industrial problem that occurs in the automobile industry's context during the testing phase of a new vehicle model. It involves determining all the variants of the new vehicle, called *prototypes*, to be manufactured to carry out a set of prescribed tests and schedule these tests over time. We formally cast this problem into a scheduling problem where jobs are tests, machines are prototypes, and two objectives are lexicographically optimized: the number of late jobs and the number of machines used. This problem involves a notion of configuration that is not addressed in the literature. First, we prove that even finding a feasible solution for the problem is NP-hard and characterize the cases where compatibility constraints amount to ensuring that only pairwise compatible jobs are assigned to each machine. We then propose a mathematical model for this problem and an alterative formulation into a path-flow formulation. To deal with the new notion of configuration, we propose a refined labeling algorithm embedded in a state-of-the-art column-and-row generation algorithm to generate primal and dual bounds from this formulation. We compare our results to those obtained by solving a constraint programming model provided by the company on industrial data from Renault and synthetic data produced to study the behavior of the different methods. Our approach finds better solutions than those obtained by the company, proves the optimality for all instances of our benchmark for the first objective function, and obtains small optimality gaps for the second objective function.

Keywords: Scheduling, Mixed integer linear programming, Column and row generation, Diving heuristic, Automobile industry

1. Introduction

In this paper, we study an optimization problem that arises in the context of the automobile industry. When a new vehicle model has just been conceived, it must undergo a series of tests before release. This phase is costly, since specific vehicles called *prototypes* are built exclusively for test purposes. The validation tests carried out in the various facilities are numerous. They are repeated on the different mechanical parts of the prototypes (thermal, acoustic, driving, sound insulation, safety, or endurance tests, for example). With increasing international competition, car manufacturers need to reduce both their costs and the time of the test phase. The company's goal is to minimize the number of tests realized after their due dates and the number of prototypes to manufacture.

*Corresponding author
Email address: francois.clautiaux@math.u-bordeaux.fr (François Clautiaux)

A difficulty the company meets is related to the notion of *configuration*. The same vehicle model is sold in different configurations, corresponding with an association of different options (with or without a sunroof, various colors, engine, sound system, ...). Some tests can only be realized if the prototype has a specific configuration. For example, some waterproofing tests need a prototype with a sunroof. Configuring the prototypes is crucial: finding suitable option associations allows using the same vehicle for more tests. It thus reduces the cost of the test phase.

From an operations research point of view, the problem is a two-step optimization problem. In the first step, one decides how many prototypes are built and which configuration to assign to each prototype. In the second step, a sequence of tests is assigned to each prototype. A test can only be assigned to a prototype if the configuration of the prototype is compatible with the test. Since prototypes are produced and tested in different facilities, the test assignment problem can be seen as a vehicle routing problem or a scheduling problem with time lags. As time lags are short compared to the test times in our instances, the scheduling vocabulary is adopted in the paper: we use the term *machine* for the prototypes and *jobs* for the tests.

In this document, we first overview the literature on scheduling problems with machine configuration and explain how our problem differs from the studied variants. We model and characterize our new industrial problem and prove that finding a feasible solution for this problem is already NP-complete. Then, we discuss some useful properties that allow us to reduce the size of the search space in optimization algorithms, and characterize the cases where the configuration constraints become equivalent to job incompatibility constraints, as defined in [1]. We then propose two MILP models, a compact model and a model with an exponential number of variables, which is solved by column generation. We also propose a matheuristic based on the latter model.

An originality of our work is to address the problem using a path-flow formulation. Network-flow formulations (see, e.g., [2]) have already shown their effectiveness in solving hard parallel machine scheduling problems. In our path-flow model, the sequence of jobs assigned to a machine corresponds with a path in a network, where nodes are related to states representing a partial solution for a machine, and arcs correspond to decisions of assigning a job to a machine in this state. Our methodological contribution is to propose an efficient algorithm to produce columns in the branch-and-cut-and-price algorithm. In our approach, configurations are only considered implicitly, *i.e.* they are a byproduct of the set of jobs assigned to each machine. To satisfy the compatibility constraints, we add new information into the states to avoid assigning a set of jobs to a machine, for which no possible configuration is possible. These features are embedded into a labeling algorithm, which produces feasible job assignments for a machine. In practice, we designed new specific resource consumption and dominance checks to account for this specificity in a state-of-the-art solver.

To obtain primal solutions even for the largest instances of the benchmark, we propose a matheuristic that considers only solutions where jobs are sequenced in increasing order of the due date on each machine. Because of the sequence-dependent lags, this restriction may exclude all optimal solutions, but it aligns to minimize the number of late jobs. This restriction prevents us from imposing elementarity constraints

in the labeling algorithm. It drastically reduces the number of non-dominated labels.

We conducted computational experiments on real-life instances used by the industrial partner. We tested our compact MILP model and our diving heuristic, and computed dual bounds using a column-and-cut generation algorithm to evaluate the quality of the heuristic. We also compared our results to those obtained by a constraint programming (CP) model used by the industrial partner. The path-flow reformulation always finds solutions that are at least as good as the CP solver (except for the smallest instance), and improves the results for the large test cases.

The paper is organized as follows. We first describe formally the problem we consider (section 2). In section 3, we propose a literature review on problem configuration and parallel machine scheduling. Then we state in section 4 some useful properties and propose a first compact MILP model. In section 5, we propose a path-flow formulation for the problem, and algorithms to solve it efficiently. Computational experiments are reported in section 6.

2. Problem definition

This section introduces our industrial problem in its practical context before formally defining it as a parallel scheduling problem.

2.1. Industrial problem

Our problem occurs when the company needs to carry out a set of tests to validate the properties of a new vehicle model. Each test is an activity realized on a given specific vehicle, called *prototype*, which is built specifically for test purposes and destroyed after the test phase. Different variants exist for each vehicle (different engines, different roofs, different colors,...), and a given prototype is not suitable for all possible tests. A given prototype is characterized by its *configuration*. A configuration is a list of values for some *parameters*. For example, parameter `roof` has three possible values: {`standard`, `sunroof`, `electrical-sunroof`}. The value for each possible parameter is decided when the prototype is manufactured and cannot be changed afterward.

Each test requires the prototype to be configured in a specific way (for example, a leak test may require the presence of a sunroof) but several configurations can be suitable for the same test (e.g., a manual or electrical sunroof may be equivalent for a leak test, but not for an electrical test). Each test is performed in a given predefined facility and has a constant processing time, which does not depend on the configuration of the prototype. For example, using a manual or electrical sunroof does not change the time needed for the leak test.

The problem we consider can thus be split into two main sets of decisions. First, we decide how many prototypes are built and how they are configured. Second, a sequence of tests is assigned to each prototype so that each test is compatible with its prototype. At each instant, a prototype cannot undergo more than one test, but several tests can be run simultaneously on different prototypes in the same location.

All vehicles are built and configured in the same factory, which can only produce a limited number of vehicles at any given time. The vehicles are sent sequentially to different facilities where they undergo

their tests. If several tests from different facilities are assigned to the same vehicle, it is moved from one facility to another. The travel times from the factory to the test facilities and between any pair of test facilities is a problem input.

In addition, some tests damage the prototypes. Once a damaging test has been performed, only damaging tests can be conducted on the prototype. Most damaging tests are called *destructive* tests and are considered to damage the whole prototype. No test can be run after them on the same prototype.

There are two objective functions to optimize. The first one is to minimize the number of late tests. This objective aims to satisfy as many internal clients as possible, i.e., those who need the results of the tests to carry on other studies. The second objective is to minimize the number of prototypes built, which is the largest share of the testing phase's cost.

The following section formally defines the problem using a scheduling vocabulary. From now on, prototypes will be called *machines*, and tests will be called *jobs*. The vector of parameter values chosen for each prototype will be called *configuration*. The time needed to travel from one facility to another will be modeled as lags between pairs of jobs.

2.2. Formal problem definition

We now present a formal definition of the problem using the scheduling terminology. Let $\mathcal{G} = \{1, \ldots, G\}$ be a set of *parameters*. Each parameter $g \in \mathcal{G}$ has a finite set $\mathcal{V}(g)$ of possible values. Let $\mathcal{M} = \{1, \ldots, M\}$ be a set of machines. Each machine m has an availability date $r_m \in \mathbb{N}$. Let $\mathcal{J} = \{1, \ldots, N\}$ be a set of *jobs*. Each job $i \in \mathcal{J}$ has a processing time $p_i \in \mathbb{Z}_+$, a due date $d_i \in \mathbb{Z}_+$. We also consider a common deadline $T \in \mathbb{Z}_+$ after which all jobs must have finished their execution. We denote by $A_{ij} \in \mathbb{Z}_+$ the time needed to switch from job i to job j, and A_{0i} a setup time needed if i is the first job processed by a machine. In the industrial problem, it corresponds with the time needed to convey the prototype from the factory to the facility assigned to i. The set of jobs is partitioned into three subsets: $\mathcal{J}^1$ (regular jobs), $\mathcal{J}^2$ (damaging jobs), and $\mathcal{J}^3$ (destructive jobs). For each parameter $g \in \mathcal{G}$, each job i has a set $\mathcal{V}_i(g) \subseteq \mathcal{V}(g)$ of compatible parameter values.

The output of the algorithm will be *configurations* for the machines, and an assignment of jobs to machines over time. We call *configuration* $\boldsymbol{c}(m) = (c_1(m), \ldots, c_G(m))$ of machine m a vector of values, one for each parameter, such that $c_g(m) \in \mathcal{V}(g)$, $\forall g \in \mathcal{G}$. We say that a job i is *compatible* with configuration $\boldsymbol{c}(m)$ if $c_g(m) \in \mathcal{V}_i(g)$, $\forall g \in \mathcal{G}$, i.e. each parameter value of $\boldsymbol{c}(m)$ is compatible with job i.

Problem 1 (Feasibility problem)**.** *Given $\mathcal{G}$, $\mathcal{J}$, $\mathcal{M}$, A, T, a feasible solution of the* ***feasibility problem*** *is an assignment of a configuration $\boldsymbol{c}(m)$ to each machine $m \in \mathcal{M}$, and an assignment of each job $i \in \mathcal{J}$ to a time interval $[s_i, s_i + p_i[$ and a machine m in such a way that*

- *the same machine processes no two distinct jobs at the same time,*
- *no machine m processes a job i before its availability date r_m plus the setup time A_{0i},*
- *the idle time between two jobs i and j on the same machine is greater than or equal to A_{ij},*
- *configuration $\boldsymbol{c}(m)$ is compatible with each job i assigned to m,*

- *no job $i \in \mathcal{J}^1$ is scheduled after a job of $\mathcal{J}^2 \cup \mathcal{J}^3$ on the same machine*
- *no job $i \in \mathcal{J}^1 \cup \mathcal{J}^2$ is scheduled after a job of $\mathcal{J}^3$ on the same machine*
- *no job $i \in \mathcal{J}$ ends after time T*

We consider two objectives in a lexicographic way. The first objective is to minimize the number of late jobs (i.e. the number of jobs i such that $s_i + p_i > d_i$). The second objective is to minimize the number of machines used (i.e., those to which at least one job is assigned).

Problem 2 (Minimum number of late jobs problem)**.** *The minimum number of late jobs problem consists of finding among the solutions of the feasibility problem the solution in which the number of late jobs is minimized.*

Problem 3 (Minimum number of machines problem)**.** *The minimum number of machine problem consists of finding among the optimal solutions of the minimum number of late jobs problem the solution in which the number of machines processing at least one job is minimized.*

This formal definition allows us to model all the features of the industrial problem. The scheduling constraints ensure that no two tests are realized simultaneously on the same vehicle. The tests cannot be performed before the vehicle has been conveyed to the test facility. The configuration constraints are exactly those of the industrial problem. Forbidden sequences of jobs allow us to model that damaging jobs are scheduled last. In what follows, for each job i, we note $\mathcal{J}^+(i)$ the set of jobs $j \in \mathcal{J}$ that cannot be scheduled after i, *i.e.* at least one of the three following conditions is true: 1) there is a parameter g such that $\mathcal{V}_i(g) \cap \mathcal{V}_j(g) = \emptyset$; 2) $i \in \mathcal{J}^3$; 3) $i \in \mathcal{J}^2$ and $j \in \mathcal{J}^1$.

A small instance of the problem with seven jobs and four machines is depicted in Figure 1, and a solution for this instance, using three machines, in Figure 2. Each rectangle represents a job, and each line is related to a machine. The release dates of the different machines are reported on the horizontal axis. The first job i on each machine m is scheduled at time $r_m + A_{0i}$. The configuration of each machine is reported on the vertical axis (e.g., the configuration of machine 1 is (b, e, g)). The idle time between jobs 4 and 3 is due to the lag $A_{43} = 1$. Note that destructive job 6 is the last job scheduled on its machine, as imposed by the problem definition. We do not picture a machine 4, which is not used. In this solution, jobs 1 and 6 are late, while all other jobs are on-time. A better (and optimal) solution of value 1 can be obtained by swapping the contents of machines 1 and 3 and scheduling job 2 before job 1.

3. Literature review

We now review the main results related to our problem in the scheduling literature, focusing on problems on parallel machines where some assignments of jobs to machines are forbidden, problems where configuration operations are performed on machines, or additional resources/equipment are used. We also review recent literature dedicated to branch-and-price methods for parallel scheduling problems.

Parameter g	1	2	3
Possible values $\mathcal{V}(g)$	a,b,c	d,e	f,g

(a) Parameters

Machine m	1	2	3	4
Release date r_m	2	4	6	8

(b) Machine description

i	p_i	d_i	$\mathcal{V}_i(1)$	$\mathcal{V}_i(2)$	$\mathcal{V}_i(3)$	subset
1	8	20	$\{a,b\}$	$\{d\}$	$\{f\}$	$\mathcal{J}^1$
2	4	8	$\{a,b\}$	$\{d\}$	$\{f,g\}$	$\mathcal{J}^1$
3	5	14	$\{c\}$	$\{e\}$	$\{f,g\}$	$\mathcal{J}^1$
4	3	8	$\{b,c\}$	$\{d,e\}$	$\{f,g\}$	$\mathcal{J}^1$
5	3	10	$\{b,c\}$	$\{d,e\}$	$\{f,g\}$	$\mathcal{J}^2$
6	3	10	$\{a,c\}$	$\{d,e\}$	$\{f,g\}$	$\mathcal{J}^3$
7	3	10	$\{b,c\}$	$\{d,e\}$	$\{f,g\}$	$\mathcal{J}^2$

(c) Job description

	0	1	2	3	4	5	6	7
0	.	2	2	2	1	1	2	1
1	.	0	0	1	1	1	0	1
2	.	0	0	1	1	1	0	1
3	.	0	0	1	1	1	0	1
4	.	1	1	1	0	0	1	0
5	.	1	1	1	0	0	1	0
6	.	0	0	1	1	1	0	1
7	.	1	1	1	0	0	1	0

(d) Distance matrix A

Figure 1: An instance of problem , with seven jobs, four normal, two damaging, and one destructive.

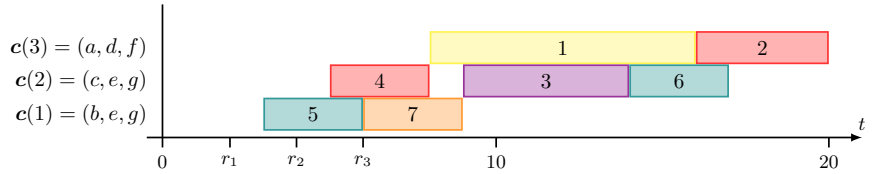

Figure 2: An example of solution for the instance of figure 1.

3.1. Problem positioning

In *multi-purpose* scheduling [3, 4, 5, 6, 7] problems, parallel machines are considered, and each job is related to a different subset of compatible machines. These problems have also been coined as *scheduling typed task systems*, *scheduling with eligibility constraints* [8, 9], *scheduling with processing set restrictions* [10, 11], *machine flexibility constraints* [12]. In all these works, the compatibility relations between jobs and machines are given as instance parameters and their definition is not part of the decisions.

Similar constraints are considered in [13], under the name of *machine qualification*. In this work, tasks belong to disjoint families, and only one machine qualified for the family can process a member of it. An original feature of the problem is that machines lose their qualification for a family after a certain amount of time has elapsed since they processed the first task in the family. The authors propose a MIP model, heuristics, and a CP model to address this version of the machine qualification problem. Machines losing their qualification is also considered in [14].

In [15], the authors use the term *machine qualification* to define a different problem, where it is possible to qualify each machine to allow it to process certain jobs. Qualification is not constrained (all machines could be qualified for all jobs) but incurs a large cost, which is considered in the objective function. The authors propose a compact MIP model for this problem.

When *auxiliary equipment constraints* are considered [16, 17, 12], only a subset of machines can process

a given job, and to process specific jobs, auxiliary equipment has to be attached to the machine. When the equipment attached to a machine is modified, it incurs a setup cost. In these problems, one can move the equipment from one machine to another, and addressing resource availability constraints is crucial.

The notion of configuration is also present in so-called *Seru* production systems (see, e.g., [18, 19, 20]). This increasingly popular organization relies on multiple "cells" of small size, which replace large assembly lines. Setting up such a system consists of deciding the number of serus (machines) and their configuration, then deciding which jobs will be assigned to which seru, and in which order. This type of production system differs from our problem since it is meant to deal with large quantities of jobs, and a key element in the scheduling problem is the number of times the same job is repeated in a given machine/seru. Another significant difference with our problem is the capability to reconfigure the serus when the demand changes.

Our problem is also related to scheduling problems with *incompatibilities* [1], where some pairs of jobs cannot be processed on the same machine. In these problems, incompatibilities are generally modeled as an undirected graph, where vertices are the jobs, and an edge $[i, j]$ belongs to the graph if job i and j are incompatible. The compatibility constraints can be expressed as a coloring constraint (for each machine m, the jobs assigned to m have the same color in a solution). The problem is thus the generalization of both parallel scheduling problems and graph-coloring problems, both known to be NP-hard in the strong sense. In practical problems, job incompatibilities are often structured. Several papers have studied special structures, focusing on a theoretical characterization. When the graph is a set of disjoint cliques, several papers use the term *bags* to designate subsets of jobs such that no two jobs of the same bag can be scheduled on the same machine [6, 21]. See also [22]. The case where the graph is the complement of disjoint cliques (called a complete multipartite graph) has also been studied [23]. From a computational point of view, an efficient method has been proposed in [24] for the general cases. In the latter work, a branch-and-price algorithm is proposed to minimize the makespan when scheduling jobs on identical parallel machines with incompatibilities between jobs.

To conclude, although the notion of machine and job compatibility exists in the scheduling literature, we are not aware of any work where machine configurations have to be decided, cannot be modified along the time horizon and where some configurations are forbidden.

3.2. Extended formulations for parallel scheduling problems

To solve parallel machine scheduling problems, using *network-flow formulations* exploits the fact that they admit a natural decomposition, where one handles the technical constraints related to each machine independently. In these formulations, the problem is cast as a minimum cost flow problem in a network, where nodes represent the possible states of a machine (including time and resource consumption), and arcs represent decisions to schedule a job from a state.

The most common approach to tackle these models is to use a *path-flow formulation*, in which the variables of the model are paths in the network. This leads to a formulation with an exponential number of variables, which is solved using branch-and-cut-and-price (BCP) techniques. From the early works of [25], many papers have been devoted to BCP for parallel scheduling. Most contributions in this field focus on the proposition of efficient algorithms for the pricing subproblem: such as handling job

disjunctions [24], for example. Kowalczyk et al. [26] proposed a Branch-and-Price algorithm for the problem $P||\sum w_jC_j$ in which the pricing problem was solved using zero-suppressed binary decision diagrams (ZDDs). BCP algorithm by Oliveira and Pessoa [27] obtained the current state-of-the-art results for the problem $P||\sum w_jT_j$ by using different families of specialized cutting planes. Bulhoes et al. [28] established new benchmark results for several scheduling problems with unrelated (i.e., non-identical machines) with sequence-dependent setup times, using many state-of-the-art techniques borrowed from BCP algorithms for vehicle routing problems. Kramer et al. [29] proposed MIP formulations and a BCP algorithm for scheduling jobs on identical parallel machines with family setup times and total weighted completion time minimization. The latter algorithm showed the best results for several classes of test instances. Song and Leus [30] came up with an efficient BCP algorithm for minimizing the makespan on parallel machines with uncertain processing times. These techniques have also been recently leveraged to solve scheduling problems with unrelated machines and batches [31].

Network-flow formulations can also be tackled by *arc-flow models* (see [32] for a survey on this topic). In these models, variables are directly flow variables on the network (see, e.g., [33] for an application of this technique to a parallel scheduling problem with release dates where the objective is to minimize the total weighted completion time). These models can lead to competitive results, mainly when all machines are identical. When the column-generation subproblem of the path-flow formulation is a resource-constrained shortest path problem, then the arc-flow formulation has to include the resource consumption into the definition of the nodes of the network, leading to a formulation of exponential size, which is generally intractable, except for some specific cases such as the temporal bin packing problem [34].

4. Problem property and an integer linear programming formulation

4.1. Problem properties

The *Minimum number of machines problem* is NP-hard since it generalizes the bin packing problem. The minimum number of late jobs problem is a generalization of $P||\sum_i U_i$, which admits a pseudo-polynomial algorithm. In our version of the problem, late jobs also have to be scheduled (they are generally discarded in classical scheduling problems where the number of late jobs is minimized). This is an issue in some cases since the choice of the configuration and the limited number of machines may make the problem infeasible. We now prove that even the feasibility problem is NP-hard by showing that the classical set-covering problem, which is known to be NP-hard, is a particular case of Problem 1.

Problem 4 (Set-covering problem (decision))**.** *Given a set $\mathcal{E}$ of n elements, a collection $\mathcal{C}$ of q sets such that $\cup_{1,\dots,q}\mathcal{C}_i = \mathcal{E}$, and an integer value M, is there a sub-collection of $\mathcal{C}$ of size M whose union is $\mathcal{E}$?*

Proposition 1. *Problem 1 is NP-complete.*

Proof. Problem 1 belongs to class NP since the size of an assignment of jobs to the machines is polynomial in the size of the input data, and each constraint can be directly checked in polynomial time.

Consider an instance $(\mathcal{E}, \mathcal{C}, M)$ of set-covering. From this instance, one can build an instance to Problem 1 as follows. The set of parameters is a singleton $\{1\}$, and its set of possible values $\mathcal{V}(1) =$

$\{1, \ldots, q\}$ (one for each possible member of family $\mathcal{C}$). We denote by v_g the value corresponding with member g of the family. For each element e of $\mathcal{E}$, we create a job e, with $p_e = 0$, $d_e = 1$, and $\mathcal{V}_e(1) = \{v_g \in \mathcal{V}(1) : e \in \mathcal{C}_g\}$. We set distance A_{ij} to 0 for any pair $(i, j) \in \mathcal{E}^2$, and $\mathcal{J}^2 = \mathcal{J}^3 = \emptyset$. The number of machines is equal to M, and for each machine $m = 1, \ldots, M$, $r_m = 0$.

If there is a solution to the obtained instance of Problem 1 using M machines, then one can obtain a solution for the corresponding set-covering instance by collecting the parameter value v_g assigned to each machine and selecting the corresponding set $\mathcal{C}_g$.

Suppose there is a solution to the set covering problem. In that case, a solution for Problem 1 can be obtained by collecting the M sets used in the solution. For each set $\mathcal{C}_g$ in the solution, assign value v_g to parameter 1 for a machine m, and assign to m the jobs e such that $e \in \mathcal{C}_m$ (and remove duplicated jobs). □

The first objective is to minimize the number of late jobs. Since there are time lags between jobs (related to transportation time between facilities) and precedence constraints, classical dominance rules used when one minimizes the number of late jobs on parallel machines ($P||\sum_i U_i$ according to the notations of [35]) do not apply anymore. Below, we prove that some weaker dominance rules can still be applied if suitable conditions are satisfied. Several results rely on the fact that there is always an optimal solution where no job can be scheduled earlier without modifying other parts of the solution (we say that such a solution is *left-shifted*).

Observation 1. *For both objective functions, there is an optimal solution that is left-shifted for each machine.*

Definition 1. *A machine m is said* horizon-unconstrained *if replacing deadline T by $+\infty$ does not modify the set of possible sequences of left-shifted jobs on machine m.*

When a machine is horizon-unconstrained, the start times of late jobs can be set arbitrarily, as long as precedence constraints are satisfied (*i.e.* damaging jobs are scheduled last).

Proposition 2. *If machine m is horizon-unconstrained, and if two distinct jobs $i \in \mathcal{J}$, $j \in \mathcal{J} \setminus \mathcal{J}^+(i)$ are such that $A_{ik} = A_{jk}$ and $A_{ki} = A_{kj}$ for all $k \neq i, j$, $\mathcal{J}^+(i) = \mathcal{J}^+(j)$, and $d_i > d_j$ then there is an optimal solution for Problem 2 (min nb late jobs) where either i does not directly precede j on the same machine m, or i is on-time and j is late. If, in addition, $p_j < p_i$, then there is an optimal solution where i does not directly precede j on machine m.*

The same dominance rule holds for Problem 3 (min nb machines).

Proof. Let S be a feasible solution for Problem 2 in which job i directly precedes j on the same machine, and i is late or j is on time. Consider the solution S' obtained by exchanging the position of i and j. We first show that S' is feasible. Then, we show that the cost of S' is not higher than that of S.

First, note that the exchange does not impact the machine's configuration, which does not depend on the sequence of jobs. Second, the exchange is locally feasible since $j \in \mathcal{J} \setminus \mathcal{J}^+(i)$ does not impact any job scheduled before i and j. Since $A_{ki} = A_{kj}$ for all $k \neq i, j$, job j begins at the same time in solution S' as i in solution S. The completion time of job i in S' is equal to the completion time of j in S'. Since $\mathcal{J}^+(i) = \mathcal{J}^+(j)$, any job scheduled after j in S can be scheduled after i in S', at the same time since

$A_{ik} = A_{jk}$ for all $k \neq i, j$. Therefore, solution S' is feasible, and only the completion times of i and j have changed compared to S.

If i is late, swapping i and j cannot increase the number of late jobs since i remains late and j cannot be on-time in S and late in S' if its completion time decreases. If j is on time, then it cannot become late in S', and since $d_i > d_j$ if i was on time in S, it cannot become late in S' either. Therefore, the objective function cannot increase.

The same proof holds for Problem 3, since the swap does not change the feasibility of the solution, and does not modify the number of machines used. □

4.2. Simple parameters

We now study the structure of the configuration constraints, and characterize cases where they can be expressed as pairwise conflicts between jobs.

Observation 2. *For two jobs $i, j \in \mathcal{J}$, if there exists a parameter g such that $\mathcal{V}_i(g) \cap \mathcal{V}_j(g) = \emptyset$, then there cannot be a solution where i and j are assigned to the same machine.*

If for two jobs i and j, $\mathcal{V}_i(g) \cap \mathcal{V}_j(g) \neq \emptyset$, we say that i and j are (pairwise) *compatible* with respect to g. Similarly, two jobs are said *incompatible* when $\mathcal{V}_i(g) \cap \mathcal{V}_j(g) = \emptyset$. For some parameters, pairwise compatibility is sufficient to satisfy the configuration constraint. We call these parameters *simple*.

Definition 2. *A parameter g is said* simple *if ensuring that no two jobs $i, j \in \mathcal{J}$ assigned to a machine m are incompatible with respect to g is sufficient to guarantee that there is a value for parameter g that is compatible with all jobs assigned to m.*

In the example of figure 1, parameter 1 is not simple, since jobs 1, 4 and 6 are pairwise-compatible with respect to 1, but there is no feasible value for parameter 1 if jobs $\{1, 4, 6\}$ are assigned to the same machine.

It is helpful to detect simple parameters because it provides an alternative way of ensuring compatibility by prohibiting incompatible jobs from being assigned to the same machine (and thus, there is no need to decide the value for this parameter during the optimization process). The fact that a parameter is simple does not depend on the other parameters. Therefore, we consider only one parameter in the remainder of this section. To avoid unnecessarily heavy notations, we will use "compatible jobs" instead of "compatible jobs with respect to parameter g". The main result of this section is the following theorem, whose proof will stem from Lemmas 1, 2 and 3 below.

Theorem 1. *Parameter g is simple if and only if for all triplets of pairwise compatibles jobs (i, j, k) of $\mathcal{J}$, $\mathcal{V}_i(g) \cap \mathcal{V}_j(g) \cap \mathcal{V}_k(g) \neq \emptyset$.*

To prove this theorem, we introduce parameter/job compatibility matrices. For a parameter $g \in \mathcal{G}$, and $S \subseteq \mathcal{J}$ a subset of jobs, let $Q^g(S)$ be the $|\mathcal{V}(g)| * |S|$ binary matrix defined as follows: $Q^g(S)_{ki} = 1$ if job i is compatible with value k of parameter g, 0 otherwise. For example, matrix (1) below represents $Q^1(\{1, 4, 6\})$ of figure 1, where columns 1, 2 and 3 represent values a, b, and c, and lines 1, 2 and 3 represent jobs 1, 4 and 6. We use the shortcut Q^g for $Q^g(\mathcal{J})$.

Lemma 1. *Parameter g is simple if and only if for all subsets of pairwise compatible jobs $S \subseteq \mathcal{J}$, $Q^g(S)$ has a line containing only* 1*s.*

Proof. Assume that for a given parameter g, $Q_g(S)$ contains a line of ones for any subset S of pairwise compatibles jobs. Consider any subset $\hat{S}$ of pairwise compatible jobs assigned to a given machine m. By assumption, $Q^g(\hat{S})$ contains a line v of ones, and thus it is possible to assign the value v to parameter g for machine m. Therefore g is simple.

Now assume that there exists a set of pairwise compatible jobs $\hat{S}$ for which $Q^g(\hat{S})$ does not have a line of ones. Consider a solution where all jobs of $\hat{S}$ are assigned to the same machine. For any possible value for g, at least one job of $\hat{S}$ is incompatible with this value, which means that parameter g is not simple. □

We now use the classical notion of *consecutive ones property.*

Definition 3 (see e.g. [36])**.** *A (0, 1)-matrix satisfies the consecutive ones property if there exists a column permutation such that the ones in each row of the resulting matrix are consecutive.*

Lemma 2. *If Q^g has the consecutive ones property, then for any subset S of pairwise compatible jobs, $Q^g(S)$ has a line containing only* 1*s.*

Proof. Assume that for a given parameter g, Q^g has the consecutive ones property. Without loss of generality, we consider that jobs are sorted in the order of the consecutive one matrix Q^g. Consider any set S of pairwise compatible jobs, and let i and j be the smallest and largest indices of jobs in S. By assumption, i and j are compatible, therefore there exists a value v such that $Q^g_{vi} = Q^g_{vj} = 1$. Since Q^g has the consecutive one property, all jobs k between i and j are such that $Q^g_{vk} = 1$. Therefore, there is a value v that is compatible with all elements of S, and thus all elements of the line v of $Q^g(S)$ are equal to one. □

Lemma 3. *For a parameter g, $Q^g(S)$ has a line of ones for all subsets of pairwise compatible jobs $S \subseteq \mathcal{J}$ if and only if for all triplets of pairwise compatible jobs (i, j, k) of $\mathcal{J}$, $\mathcal{V}_i(g) \cap \mathcal{V}_j(g) \cap \mathcal{V}_k(g) \neq \emptyset$.*

Proof. We first show that the condition is necessary. If there is a triplet of pairwise compatible jobs (i, j, k) of $\mathcal{J}$, such that $\mathcal{V}_i(g) \cap \mathcal{V}_j(g) \cap \mathcal{V}_k(g) = \emptyset$, then matrix $Q^g(\{i, j, k\})$ is the following.

$$\begin{pmatrix} 1 & 1 & 0 \\ 0 & 1 & 1 \\ 1 & 0 & 1 \end{pmatrix} \tag{1}$$

This matrix does not have a line of ones, so the condition is necessary.

Now, we show that the condition is sufficient. Assume that for all triplets of pairwise compatible jobs (i, j, k) of $\mathcal{J}$, $\mathcal{V}_i(g) \cap \mathcal{V}_j(g) \cap \mathcal{V}_k(g) \neq \emptyset$. By Lemma 2, we know that if Q^g has the consecutive one property, then for all pairwise compatible sets S, $Q^g(S)$ has a line of ones. In [37], the author shows that a matrix has the consecutive ones property if and only if it does not contain the submatrices depicted in Figure 3.

Matrices M_{IV} and M_V cannot be obtained by selecting pairwise compatible jobs (columns 1 and 3 for M_{IV}, columns 2 and 5 for M_V have no common element). In matrix $M_{III}(k)$, for any $k \geq 3$, the first

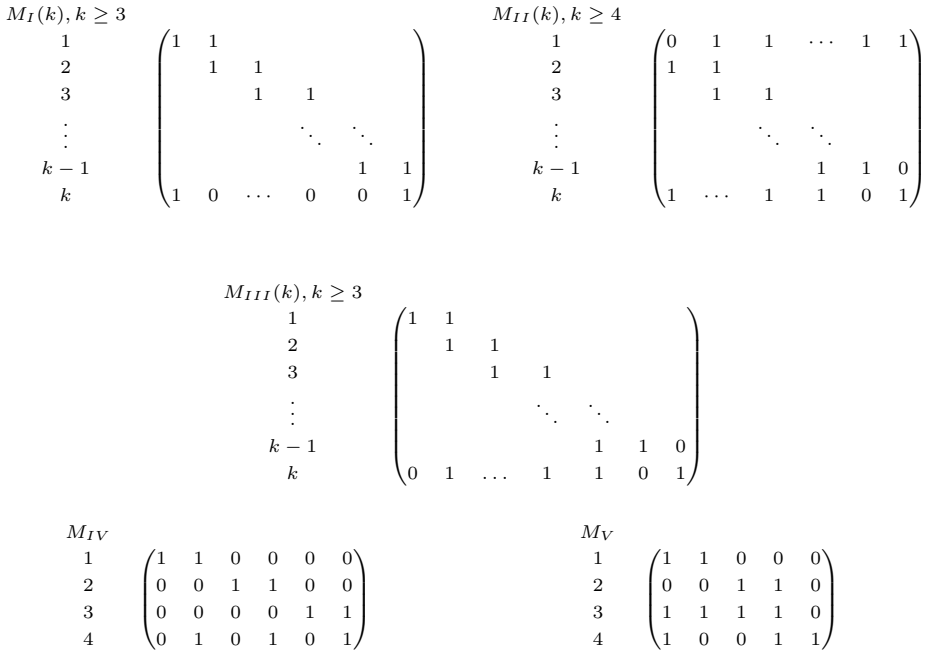

Figure 3: Five forbidden structures for the consecutive ones property [37].

and the last columns do not have any common value, so they cannot be obtained by selecting pairwise compatible jobs. The same can be said about $M_{II}(k)$ for $k \geq 4$ (first and penultimate columns). For $M_I(k)$, $k \geq 4$, columns 2 and 4 do not have any common value; therefore, this configuration cannot occur either. Only one configuration remains: $M_I(3)$ (matrix (1) above).

Therefore, we know that a value/job matrix obtained by selecting a set of pairwise compatible jobs has the consecutive one property if and only if it does not contain $M_I(3)$. This matrix is related to a set of three values i, j, and k that are pairwise compatible, and such that $\mathcal{V}_i(g) \cap \mathcal{V}_j(g) \cap \mathcal{V}_k(g) \neq \emptyset$, which contradicts the assumption. This proves that the condition is sufficient. □

The proof of Theorem 1 directly derives from Lemmas 1 and 3.

4.3. An Integer Linear Programming Formulation

We first define an integer linear programming formulation for the first version of the problem (minimization of the number of late jobs). We define the following variables. For $i \in \mathcal{J}$, $m \in \mathcal{M}$, $x_{im} \in \{0,1\}$ is equal to 1 if i is the first job assigned to machine m, 0 otherwise. For $i \in \mathcal{J}$, variable $u_i \in \{0,1\}$ is equal to 1 if job i is late, 0 otherwise. For $i \in \mathcal{J}$, $j \in \mathcal{J} \setminus \{i\}$, $m \in \mathcal{M}$, binary variable y_{ijm} indicates that j directly follows i on machine m (value 1), or not (value 0). For $m \in \mathcal{M}$, $g \in \mathcal{G}$ and $v \in \mathcal{V}(g)$, binary variable α_{mgv} indicates that machine m has value v for parameter g in its configuration (value 1) or not (value 0). For $i \in \mathcal{J}$, variable $t_i \in \mathbb{R}_+$ is equal to the completion time of job i. In what follows, Q is a large real constant, which can be set to $\sum_{i \in \mathcal{J}}(p_i + \max_{j \neq i} A_{ji})$ for example. We also recall that $\mathcal{J}^+(i)$ is the set of jobs that cannot be scheduled after i.

$$\min \sum_{i \in \mathcal{J}} u_i \tag{2a}$$

$$\sum_{m \in \mathcal{M}} x_{im} + \sum_{m \in \mathcal{M}} \sum_{j \in \mathcal{J} \setminus \{i\}} y_{jim} = 1 \qquad \forall i \in \mathcal{J} \tag{2b}$$

$$\sum_{i \in \mathcal{J}} x_{im} \leq 1 \qquad \forall m \in \mathcal{M} \tag{2c}$$

$$\sum_{j \in \mathcal{J} \setminus \{i\}} y_{ijm} \leq x_{im} + \sum_{j \in \mathcal{J} \setminus \{i\}} y_{jim} \qquad \forall i \in \mathcal{J}, \forall m \in \mathcal{M} \tag{2d}$$

$$t_i \geq p_i + \sum_{m \in \mathcal{M}} (A_{0i} + r_m) x_{im} \qquad \forall i \in \mathcal{J} \tag{2e}$$

$$t_i \geq t_j - Q + (A_{ji} + p_i + Q) \sum_{m \in \mathcal{M}} y_{jim} \qquad \forall i \in \mathcal{J}, \forall j \in \mathcal{J} \setminus \{i\} \tag{2f}$$

$$t_i - Q u_i \leq d_i \qquad \forall i \in \mathcal{J} \tag{2g}$$

$$\sum_{v \in \mathcal{V}(g)} \alpha_{mgv} = 1 \qquad \forall m \in \mathcal{M}, \forall g \in \mathcal{G} \tag{2h}$$

$$x_{im} + \sum_{j \in \mathcal{J} \setminus \{i\}} y_{jim} \leq \sum_{v \in \mathcal{V}_i(g)} \alpha_{mgv} \qquad \forall i \in \mathcal{J}, \forall m \in \mathcal{M}, \forall g \in \mathcal{G} \tag{2i}$$

$$\sum_{j \in \mathcal{J}^+(i)} y_{ijm} = 0 \qquad \forall i \in \mathcal{J}, \forall m \in \mathcal{M} \tag{2j}$$

$$\alpha_{mgv} \in \{0, 1\} \qquad \forall m \in \mathcal{M}, \forall g \in \mathcal{G}, \forall v \in \mathcal{V}(g) \tag{2k}$$

$$x_{im} \in \{0, 1\} \qquad \forall i \in \mathcal{J}, \forall m \in \mathcal{M} \tag{2l}$$

$$y_{ijm} \in \{0, 1\} \qquad \forall i \in \mathcal{J}, \forall j \in \mathcal{J} \setminus \{i\}, \forall m \in \mathcal{M} \tag{2m}$$

$$t_i \in \mathbb{R}_+ \qquad \forall i \in \mathcal{J} \tag{2n}$$

$$u_i \in \{0, 1\} \qquad \forall i \in \mathcal{J} \tag{2o}$$

Constraints (2b) ensure that each job is assigned to exactly one machine (either it is the first job on a machine, or it follows another job), while constraints (2c) ensure that there cannot be two first jobs on any machine. Constraints (2d) ensure that if a job is assigned to machine m, it has a predecessor or is the first job on m. Constraints (2e) and (2f) are used to compute the completion time of job i if it is assigned to machine m. Constraints (2g) indicate that u_i has to be 1 if i is late. Constraints (2h) indicate that for each parameter and each machine, exactly one value has to be selected. Constraints (2i) forbid assigning job i to machine m if the configuration of m is incompatible with i. Constraints (2j) ensure that a job cannot be followed by a forbidden successor.

When the minimum number of late jobs, denoted $\bar{u}$ below, is computed, the secondary objective is to minimize the number of machines used. The model is modified as follows.

$$\min \sum_{i \in \mathcal{J}} \sum_{m \in \mathcal{M}} x_{im} \tag{3a}$$

$$\sum_{i \in \mathcal{J}} u_i = \bar{u} \tag{3b}$$

$$(2a) - (2o)$$

Observations made in the previous section can be used to propose valid inequalities for the models above.

Proposition 3. *If machine m is horizon-unconstrained, and if two jobs $i \in \mathcal{J}$, $j \in \mathcal{J} \setminus \mathcal{J}^+(i)$ are such that $A_{ik} = A_{jk}$ and $A_{ki} = A_{kj}$ for all $k \neq i, j$, $\mathcal{J}^+(i) = \mathcal{J}^+(j)$, and $d_i > d_j$, the following constraints are valid for* (2a)-(2o).

$$y_{ijm} \leq u_i \tag{4}$$

$$y_{ijm} \leq 1 - u_j \tag{5}$$

Proof. The proof directly stems from Proposition 2. We know that there is an optimal solution where either i does not precede j on the same machine ($y_{ijm} = 0$), or i is on time ($u_i = 1$) and j is late ($1 - u_j = 1$). Therefore, there is an optimal solution for (2a)-(2o) that satisfies inequalities (4) and (5). □

The following observation states that if no job is assigned to a machine m, machine m does not need to be configured. Without loss of generality, we allow a machine m to have all binary variables α_{mgv} equal to 0 if m is not used.

Observation 3. *It is possible to replace constraint* (2h) *by the following set of equalities.*

$$\sum_{v \in \mathcal{V}(g)} \alpha_{mgv} = \sum_{i \in \mathcal{J}: g \in \mathcal{G}(i)} x_{im} \quad \forall m \in \mathcal{M}, \forall g \in \mathcal{G} \tag{6}$$

The following proposition allows to force variable u_i (indicating that job i is late) to 1 without relying on completion time variables t_i. This occurs when i is scheduled on a machine whose availability date is too late, or if i cannot be on time if it is scheduled after a job j on machine m (considering the availability date of m, the computing time of j, and the lag between j and i).

Proposition 4. *The following constraint is valid for* (2a)-(2o).

$$u_i \geq \sum_{m \in \mathcal{M}: r_m + A_{0i} + p_i > d_i} x_{im} + \sum_{m \in \mathcal{M}} \sum_{j \in \mathcal{J} \setminus \{i\}: r_m + p_j + A_{ji} + p_i > d_i} y_{jim} \quad \forall i \in \mathcal{J} \tag{7}$$

Proof. First, note that in an integer solution, the right-hand side of the inequality can only be equal to zero or one by constraint (2b). If the right-hand side is equal to zero, the inequality is always true since u_i is non-negative. If the right-hand side equals one, there are two possibilities. First possibility: there is a value m such that $x_{im} = 1$ and $r_m + A_{0i} + p_i > d_i$. In this case, i ends at time $r_m + A_{0i} + p_i$, and thus it is late, and u_i equals 1. Second possibility: there is are value m and a value j such that $y_{jm} = 1$, and $r_m + p_j + A_{ji} + p_i > d_i$. In this case, job i cannot end before $r_m + p_j + A_{ji} + p_i$, and thus it is late, and u_i must be equal to 1. □

5. A path formulation

In this section, we propose a reformulation of the problem, where paths in a graph model the sequences of jobs. An originality of our reformulation is that configurations are not considered explicitly. They are deduced *a posteriori* by the sequence of jobs chosen on each machine. At the beginning of the time horizon, all values are possible for each parameter. Each time a job is selected, it forbids a set of values for each parameter. If the set of possible values for a parameter becomes empty, the solution/path is not feasible.

Network flow formulations can generally be solved either by using an arc-flow formulation or a path-flow formulation. In our case, an arc-flow formulation is not an option, since one has to recall along a path the parameter values that previous jobs have forbidden, which leads to a graph of exponential size. Therefore, we use a path-flow formulation and solve it using a column-and-cut generation algorithm. Below, we show how to extend a state-of-the-art labeling algorithm to account for the new configuration constraints in the pricing procedure.

We first introduce the graph that represents the assignment of jobs to machines. Then, we describe our master problem, the subproblem, and how the latter is solved through a labeling algorithm.

5.1. A graph representing job schedules and machine assignment

We first define the graph that represents possible job schedules on machines. Our graph $H = (\mathcal{N}, \mathcal{A})$ has a set of nodes $\mathcal{N} = \{b\} \cup \mathcal{N}^m \cup \mathcal{N}^o \cup \mathcal{N}^\ell \cup \{e\}$. Nodes b and e are artificial vertices related to the unique source and termination nodes. Nodes in $\mathcal{N}^m$ represent machines, i.e., $\mathcal{N}^m = \mathcal{M}$. For each job $j \in \mathcal{J}$, two nodes are created: one on-time node in $\mathcal{N}^o$ and one late node in $\mathcal{N}^\ell$. We denote by $j(n)$ the job represented by any node $n \in \mathcal{N}^o \cup \mathcal{N}^\ell$. To simplify the notation, we consider that $j(n) = 0$ for nodes $n \in \{b, e\} \cup \mathcal{N}^m$, which do not represent jobs. The arc set is also partitioned into several subsets: $\mathcal{A} = \mathcal{A}^m \cup \mathcal{A}^n \cup \mathcal{A}^e$. Set $\mathcal{A}^m$ contains an arc (b, m) for every machine $m \in \mathcal{M}$. Set $\mathcal{A}^n$ contains one arc (n_i, n_j) for every pair of nodes $\{n_i, n_j\} \in \left(\mathcal{N}^m \cup \mathcal{N}^o \cup \mathcal{N}^\ell\right) \times \left(\mathcal{N}^o \cup \mathcal{N}^\ell\right)$ such that either $n_i \in \mathcal{N}^m$ or $j(n_j) \notin \mathcal{J}^+(j(n_i))$. Set $\mathcal{A}^e$ contains one arc (n, e) for every node $n \in \mathcal{N}^o \cup \mathcal{N}^\ell$. Figure 4 gives an example of such a network for 3 machines and 4 jobs. The source b and the sink e are colored in white, machine nodes, standard job nodes, and damaging job nodes are respectively colored in yellow, blue, and orange. Nodes $1^o, \ldots, 4^o$ correspond with selecting these jobs on time, while nodes $1^\ell, \ldots, 4^\ell$ correspond with selecting these jobs late. Arcs of $\mathcal{A}^m$, $\mathcal{A}^n$, and $\mathcal{A}^e$ are respectively colored in red, black, and green. Note that for any job j, it is impossible to go directly from j^ℓ to j^o and vice-versa. Job 4 is the only damaging job, so the only out-going arcs from 4^o and 4^ℓ are headed to e.

Any feasible path from b to e in graph H passes by exactly one node in $\mathcal{N}^m$ (selection of the machine) and at least one node in $\mathcal{N}^o \cup \mathcal{N}^\ell$ (at least one job is assigned). It follows exactly one arc in $\mathcal{A}^m$, at least one arc in $\mathcal{A}^n$, and exactly one arc in $\mathcal{A}^e$. Figure 5 shows a possible path in the example of figure 4, which corresponds to assigning a sequence of jobs to a machine. This path corresponds with job sequence $(2, 1, 4)$ on machine 1, where 2 and 4 are on-time and 1 is late. The path is clearly elementary. It is feasible if the sequence $(2, 1, 4)$ allows scheduling job 2 and jobs 4 on time (accounting for the release

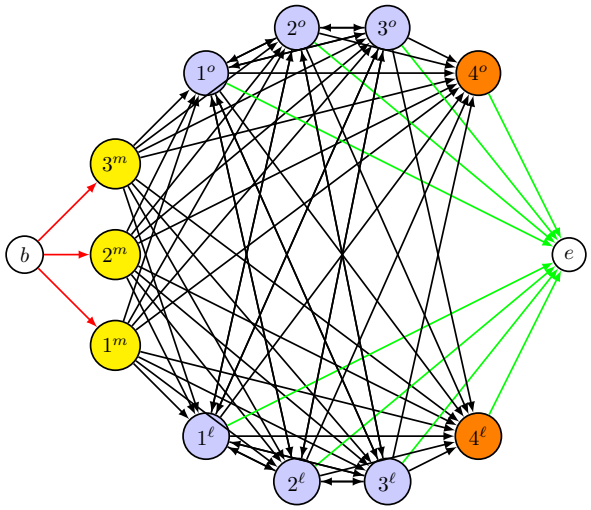

Figure 4: An example of network with three machines and four jobs

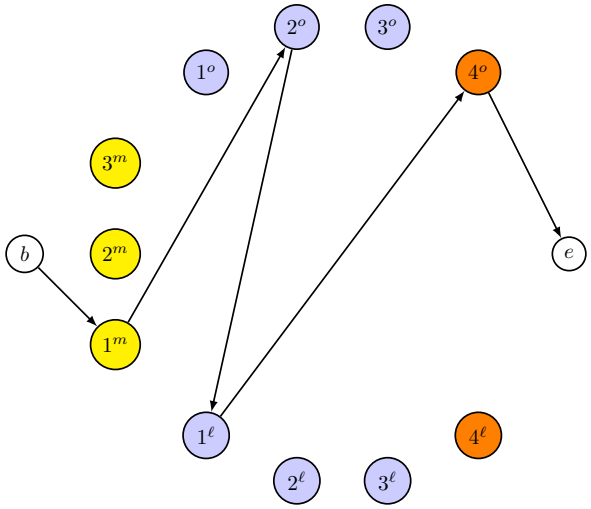

Figure 5: A path in the network of Figure 4

date of machine 1 and the different lags), and if, in addition, there exists a possible configuration for the machine that is compatible with jobs 1, 2 and 4. The machine used, and the jobs assigned can be retrieved by inspecting the nodes used in the path.

We call *feasible path* a path from b to e in which 1) each job is not selected twice, 2) on-time jobs are completed before their due dates, 3) late jobs are completed after their due dates and before T, and 4) a configuration for the machine exists that is compatible with all selected jobs. We use *resources* to verify the feasibility of paths.

We use a resource to compute the accumulated time along the path. The time resource consumption q_a of each arc $a \in \mathcal{A}$ is equal to

$$q_a = \begin{cases} 0, & a \in \mathcal{A}^m \cup \mathcal{A}^e, \\ A_{0,j(n_k)} + p_{j(n_k)}, & a = (m, n_k) \in \mathcal{A}^n \text{ and } m \in \mathcal{N}^m, \\ A_{j(n_i),j(n_k)} + p_{j(n_k)}, & a = (n_i, n_k) \in \mathcal{A}^n \text{ and } n_i \notin \mathcal{N}^m. \end{cases}$$

The accumulated time resource consumption bounds $[t_n^-, t_n^+]$ for a node $n \in \mathcal{N}$ are equal to

$$[t_n^-, t_n^+] = \begin{cases} [0, T], & n \in \{b, e\}, \\ [r_n, T], & n \in \mathcal{N}^m, \\ \left[0, d_{j(n)}\right], & n \in \mathcal{N}^o, \\ \left[d_{j(n)} + 1, T\right], & n \in \mathcal{N}^\ell. \end{cases}$$

The accumulated consumption q_k^P of the time resource after visiting the k-th node of path $P = (b = n_0^P, a_1^P, n_1^P, \ldots, a_{k(P)}^P, n_{k(P)}^P = e)$ is equal to

$$q_k^P = \max\left\{t_{n_k^P}^-, q_{k-1}^P + q_{a_k^P}\right\}, \ k \in \{1, \ldots, k(P)\}, \ \text{ and } q_0^P = 0. \tag{8}$$

This resource is called a *disposable resource* [38]. The term *disposable* means that an additional positive value of the time resource may be accumulated to satisfy bounds $[t_n^-, t_n^+]$ without paying an extra cost, i.e., a positive value of the resource may be disposed of. A path P satisfies the time resource if $q_k^P \leq t_{n_k^P}^+$ for all $k \in \{1, \ldots, k(P)\}$.

To verify elementarity constraints, we use one binary resource for each job $j \in J$. The accumulated consumption l_{kj}^P of the j-th elementarity resource after visiting k-th node of path P is equal to

$$l_{kj}^P = l_{k-1,j}^P + \begin{cases} 1, & \text{if } a_k^P = (n', n) : \ j(n) = j, \\ 0, & \text{otherwise,} \end{cases} \quad k \in \{1, \ldots, k(P)\}, \ \text{ and } l_{0j}^P = 0. \tag{9}$$

A path P satisfies the elementarity resources if $l_{kj}^P \leq 1$ for all $k \in \{1, \ldots, k(P)\}$ and for all $j \in \mathcal{J}$.

Finally, we introduce so-called *selection resources* to impose the constraint that there exists a configuration compatible with all jobs assigned to that machine. There is one selection resource for every parameter $g \in \mathcal{G}$. Let C_{kg}^P be set of possible values for parameter $g \in \mathcal{G}$ after visiting the k-th node of path P. Initially $C_{0g}^P = \mathcal{V}(g)$, then values are removed along the path as follows:

$$C_{kg}^P = C_{k-1,g}^P \bigcap \begin{cases} \mathcal{V}_j(g), & j = j(n_k^P) \in \mathcal{J}, \\ \mathcal{V}(g), & \text{otherwise,} \end{cases} \quad k \in \{1, \ldots, k(P)\}. \tag{10}$$

A path P satisfies the selection resources if $C_{k(P)g}^P \neq \emptyset$ for all $g \in \mathcal{G}$.

We now explain how we can exploit the fact that many parameters are simple (see definition 2). If a parameter is simple, we do not create a selection resource and consider directly pairwise disjunctions between jobs. For this purpose, we define an undirected conflict graph $H^d = (\mathcal{J}, \mathcal{E}^d)$, where $\mathcal{E}^d$ is the set of incompatible pairs of jobs. A pair (i, j) of jobs is incompatible if there exists a simple parameter $g \in \mathcal{G}$ such that $\mathcal{V}_i(g) \cap \mathcal{V}_j(g) = \emptyset$. In graph H^d, we identify complete bipartie subgraphs $(\mathcal{X}^d \cup Y^d, \mathcal{E}^d)$ where $\mathcal{E}^d = \mathcal{X}^d \times \mathcal{Y}^d$, i.e., each element of $\mathcal{X}^d$ is in conflict with each element of $\mathcal{Y}^d$. We report an example of such complete bipartite subgraph in Figure 6, where $\mathcal{X}^d = \{X_1, X_2, X_3, X_4\}$ and $\mathcal{Y}^d = \{Y_1, Y_2, Y_3\}$. In this case, one selection resource is sufficient to check that if a job of $\mathcal{X}^d$ is selected, no job from $\mathcal{Y}^d$ can be selected, and vice-versa. For each bipartite subgraph found, the corresponding conflicts are replaced by a single selection resource. This resource has two possible values, say 0 and 1. Every job in $\mathcal{X}^d$ is compatible only with value 0 of this resource, every job in $\mathcal{Y}^d$ is compatible only with value 1 of this

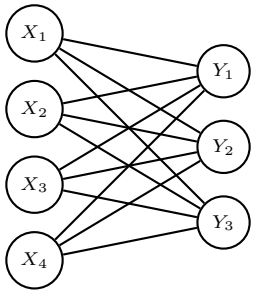

Figure 6: A complete bipartite subgraph of the conflict graph

resource, and other jobs are compatible with both values of this resource. Thus, a potentially smaller number of selection resources are defined for simple parameters. In our experiments we compute a cover of the conflict graph by complete bipartite subgraphs using a greedy heuristic.

In the remainder, we denote by $\mathcal{P}$ the set of all feasible paths in graph H, i.e., paths that satisfy all disposable, elementarity, and selection resources.

5.2. Master problem

For each path $P \in \mathcal{P}$, we define constant $\alpha_a^P \in \mathbb{Z}_+$, which is equal to the number of times arc $a \in \mathcal{A}$ is used in path P. Our formulation is based on a unique set of variables. For each path $P \in \mathcal{P}$, binary variable λ_P equals one if and only if path P is used in the solution. With a slight abuse of notation, we introduce $\mathcal{A}(j) = \{(n, n') \in \mathcal{A} : j(n') = j\}$ the set of arcs that lead to job i (on-time or late). We also use the shortcut $\mathcal{A}^\ell = \{(n, n') \in \mathcal{A}, n' \in \mathcal{N}^\ell\}$ for the set of arcs that lead to a late job. We now present the path formulation for our problem for the first objective function.

$$\min \sum_{P \in \mathcal{P}} \sum_{a \in \mathcal{A}^\ell} \alpha_a^P \lambda_P \tag{11a}$$

$$\sum_{P \in \mathcal{P}} \sum_{a \in \mathcal{A}(j)} \alpha_a^P \lambda_P = 1, \qquad \forall j \in \mathcal{J}, \tag{11b}$$

$$\sum_{P \in \mathcal{P}} \sum_{a=(b,m) \in \mathcal{A}^m} \alpha_a^P \lambda_P \leq 1, \qquad \forall m \in \mathcal{M}, \tag{11c}$$

$$\lambda_P \in \{0, 1\}, \qquad \forall P \in \mathcal{P}. \tag{11d}$$

The objective function minimizes the number of times an arc going to a late job node is selected. Constraints (11b) ensure that every job is processed exactly once. Constraints (11c) guarantee that no more than one schedule is assigned to every machine. Constraints (11d) state the integrality of variables λ.

Let z^* be the value of the optimal solution for the first objective function (number of late jobs). To minimize the number of machines used under the condition that the number of late jobs equals z^*, the objective function is replaced by $\min \sum_{P \in \mathcal{P}} \sum_{a \in \mathcal{A}^m} \alpha_a^P \lambda_P$, and the additional constraint is $\sum_{P \in \mathcal{P}} \sum_{a \in \mathcal{A}^\ell} \alpha_a^P \lambda_P \leq z^*$.

5.3. Column-and-cut generation

The linear relaxation of formulation (11) is solved by the standard column generation procedure, which alternates between solving the linear master problem with a restricted set of variables and solving the pricing problem that dynamically generates variables λ_P with a negative reduced cost. Let $\pi \in \mathbb{R}^{|\mathcal{J}|}$ and $\theta \in \mathbb{R}_{-}^{|\mathcal{M}|}$ be the dual variable vectors respectively associated with constraints (11b) and (11c). The reduced cost of a variable λ_P can be computed as follows.

$$\min_{P \in \mathcal{P}} \sum_{a \in \mathcal{A}^\ell} \alpha_a^P - \sum_{j \in \mathcal{J}} \sum_{a \in \mathcal{A}(j)} \pi_j \alpha_a^P - \sum_{m \in \mathcal{M}} \sum_{a=(b,m) \in \mathcal{A}^m} \theta_m \alpha_a^P$$

Note that the reduced cost only depends on the arc variables. The pricing problem consists of finding in graph H a resource-feasible path of smallest reduced cost. We use a labeling algorithm where every label L represents a partial path $P(L)$ started in the source node. Every label L is characterized by a tuple $(c^L, n^L, q^L, \boldsymbol{l}^L, \boldsymbol{C}^L)$, where c^L is the reduced cost of path $P(L)$, n^L is the last visited node of path $P(L)$, q^L is the accumulated consumption of the time resource of path $P(L)$, $\boldsymbol{l}^L$ is the vector of accumulated consumption of the elementarity resources of path $P(L)$, and $\boldsymbol{C}^L$ is the vector of sets of possible values for machine parameters of path $P(L)$.

Every label L is extended along every arc a whose tail is n^L, and its values q^L, $\boldsymbol{l}^L$, and $\boldsymbol{C}^L$ are updated according to equations (8)–(10). Every label L corresponding to an infeasible partial path is not created. To avoid complete enumeration, dominated labels are also discarded from the search. A label L' is dominated by a label L if for any complete feasible path $P(L') \oplus P'$, path $P(L) \oplus P'$ is also feasible, and the reduced cost of path $P(L') \oplus P'$ is not smaller than the reduced cost of path $P(L) \oplus P'$, where $\oplus$ denotes the concatenation of two partial paths. A sufficient condition for domination of a label L' by a label L is

$$c^L \leq c^{L'},\ n^L = n^{L'},\ q^L \leq q^{L'},\ \boldsymbol{l}^L \leq \boldsymbol{l}^{L'},\ C_g^L \supseteq C_g^{L'}\ \forall g \in \mathcal{G}.$$

Even when the *bucket graph* variant of the labelling algorithm [39] is used to implement our method, the algorithm's running time is large for real-life instances. Therefore, we relax the elementarity constraints in the pricing problem. Note that the path formulation (11) remains valid even if the set $\mathcal{P}$ contains non-elementary paths due to set-partitioning constraints (11b).

To improve the quality of the relaxation, we use state-of-the-art column-and-row techniques from the literature. For the sake of completeness, we recall below the main components used, referencing the scientific document describing the techniques used in depth. In particular, precise algorithmic details are provided in [40, 41].

To improve the lower bound generated by the master problem, we use the dynamic *ng*-path technique and rank-one Chvátal-Gomory cuts based on the set-partitioning constraints. In the *ng*-path relaxation, introduced by [42] for a vehicle routing problem, a memory consisting of neighbor clients is defined for every client, and the latter cannot be revisited as long as the customer does not leave its memory. Thus, elementarity constraints are imposed only partially. In the dynamic *ng*-path technique, proposed by [43], memories are augmented dynamically by analyzing the paths forming the solution of the master problem.

Analogously, we define a memory for each job, which contains only the job itself at the start of the algorithm. Each time a fractional solution $\bar{\boldsymbol{\lambda}}$ is obtained by column generation, we consider every non-elementary path P such that $\bar{\lambda}_P > 0$ and include in the memory of all jobs j appearing more than once in P all jobs scheduled between two appearances of j in P. Afterward, we exclude from the master problem all variables λ_P, such that P is not feasible anymore according to the ng-path relaxation with augmented job memories. Thus, each non-elementary path participating in the current fractional solution cannot appear in the master problem anymore. We apply the column generation algorithm again after increasing job memories and excluding the corresponding variables λ, and we repeat this process at most seven times.

After applying dynamic ng-path relaxation, we try to increase the lower bound further by generating valid inequalities, which are based on constraints (11b) relaxed to set-packing constraints. Given a vector $\boldsymbol{\rho}$ of multipliers, one multiplier ρ_j for every job $j \in \mathcal{J}$, the following inequality obtained by the Chvátal-Gomory rounding procedure is valid for formulation (11):

$$\sum_{P \in \mathcal{P}} \left\lfloor \sum_{j \in \mathcal{J}} \rho_j \cdot \sum_{a \in \mathcal{A}(j)} \alpha_a^P \right\rfloor \lambda_P \leq \left\lfloor \sum_{j \in \mathcal{J}} \rho_j \right\rfloor . \tag{12}$$

We separate the Chvátal-Gomory rank-one cuts (12) based on up to five set packing constraints, i.e., such that the number of non-zero values in vector $\boldsymbol{\rho}$ is at most five. Non-dominated vectors of multipliers for this case were found in [40] [1]. The cut separation problem is solved independently for each such vector of multipliers. Again, at most, seven rounds of cut separation are performed, and the column generation algorithm is rerun between separation rounds. Each cut (12) is related to a dual value, which introduces one additional resource into the pricing problem. Increasing the number of resources weakens the dominance checks since a label dominates another if it is better for each resource (including those related to elementarity cuts). Thus, many active rank-one cuts may slow down the pricing phase considerably. To mitigate the impact of active rank-one cuts on the pricing problem difficulty, we use the limited-memory technique introduced in [41].

To summarize, we initially solve the column generation process until convergence is reached without taking into account the elementarity of the solution. Then, we analyze the columns used in the solution obtained, increase the parameter of the limited memory in the labeling algorithm accordingly, and rerun the column generation. This procedure is repeated seven times. Then, we consider the master solution obtained, add violated Chvátal-Gomory rank-one cuts, and rerun the column generation to complete convergence. This procedure is also repeated seven times.

5.4. The diving heuristic

We now present a heuristic method based on the above column-and-cut generation algorithm. This method computes primal feasible solutions, even when the branch-and-price cannot converge to an optimal solution. Our method relies on two ingredients: a truncated tree search and a heuristic pricing subproblem

[1] These vectors are $(\frac{1}{2}, \frac{1}{2}, \frac{1}{4}, \frac{1}{4}, \frac{1}{4}), (\frac{3}{4}, \frac{1}{4}, \frac{1}{4}, \frac{1}{4}, \frac{1}{4}), (\frac{3}{5}, \frac{2}{5}, \frac{2}{5}, \frac{1}{5}, \frac{1}{5}), (\frac{2}{3}, \frac{2}{3}, \frac{2}{3}, \frac{1}{3}, \frac{1}{3}), (\frac{3}{4}, \frac{3}{4}, \frac{2}{4}, \frac{2}{4}, \frac{1}{4})$ and their permutations.

based on an *aggressive* dominance rule, i.e., a dominance rule that may exclude optimal solutions, but helps reduce the complexity of the subproblem by a wide range.

To produce integer solutions for formulation (11), we use the diving heuristic with Limited Discrepancy Search (LDS) [44]. After obtaining a fractional solution $\bar{\boldsymbol{\lambda}}$ by column-and-cut generation, we fix to one the value of a variable λ_P such that $P = \text{argmax}_{P \in \mathcal{P}}(\bar{\lambda}_P)$. We then resume the iterative column generation algorithm (without cut generation) and repeat this process until the solution produced by column generation is integer. The alternation of the column generation algorithm and the fixing procedure corresponds to the simple diving heuristic or one dive in the branch-and-bound tree. We apply ten dives in the branch-and-bound tree to diversify the search using LDS. The diving heuristic with LDS uses a tabu list of forbidden columns, which cannot be selected in later dives. For details of this heuristic, refer to [44]. We use the same parameter set as in this reference.

As it is not required anymore to calculate a valid lower bound, we use a heuristic algorithm for the pricing subproblem. We produce this heuristic by reducing the graph H, considering only a subset of arcs, and thus potentially excluding arcs that are needed in an optimal solution. Our graph reduction is based on the observation that in the classic scheduling problem $P||\sum_i U_i$, on-time jobs are scheduled in a non-decreasing order of their due dates on every machine. Since the time lags in our instances are relatively small compared to the processing times of the jobs, imposing the smallest due date rule is a reasonable choice in a heuristic. Thus, we remove in graph H all arcs $a = (n, n') \in \mathcal{A}^n$ such that $d_{j(n)} > d_{j(n')}$. As the order of the jobs is now fixed, elementarity constraints are automatically satisfied in the pricing problem, and the dynamic ng-path relaxation is no longer necessary.

6. Computational experiments

The primary purpose of our computational experiments is to evaluate the quality of the solutions found by our diving heuristic on real cases, compared to reference solutions computed by the company and those obtained using the compact formulation. We also estimate the gap between these results and the optimal values using lower bounds produced by our MIP models.

6.1. Instance description and practical setting

We use industrial instances provided by Renault. Each instance has a unique facility that produces the prototypes (machines) and several facilities in different countries where tests (jobs) are performed. It generally takes two days to move from one facility to another in the same country and three to five days when they are located in different countries. The data set is split into two subsets. Set 1 contains only simple parameters, while in each instance of Set 2, there is at least one non-simple parameter. For each instance $i \in \{3, \ldots, 11\}$, the only differences between i and im are the possible parameter values. We report in Tables 1 and 2 a summary of the instance sizes: number of jobs (N), number of machines (M), number of parameters (G), and total number of parameter values. The highest values are reported in bold.

For the different MIP approaches, we optimize the two objectives in a two-stage process. We first minimize the number of late jobs. Then, we add this number of late jobs as a constraint when the minimum

Instance	1	2	3	4	5	6	7	8	9	10	11
Nb. jobs	69	70	116	135	146	152	164	197	216	230	**365**
Nb. machines	30	34	75	90	82	52	50	84	116	70	**180**
Nb. parameters	58	32	172	215	165	155	178	61	30	**216**	98
Nb. values	124	81	368	462	348	337	385	142	74	**479**	240

Table 1: Instance summary (set 1, simple parameters only)

Instance	3m	4m	5m	6m	7m	8m	9m	10m	11m
Nb. jobs	116	135	146	152	164	197	216	230	**365**
Nb. machines	75	90	82	52	50	84	116	70	**180**
Nb. parameters	169	219	169	157	181	62	32	**219**	100
Nb. values	364	474	358	343	394	145	82	**487**	246

Table 2: Instance summary (set 2, mixed simple and non-simple parameters)

number of machines is sought. In some cases, the solver has not converged for the first objective function after one hour. In these cases, we stop the search and use the best solution value found by the solver to parameterize the second. The maximum computing time for the second objective function is two hours minus the time spent for the first objective function.

As a reference, we use the results obtained by a CP model previously designed in the company (see Appendix A). To solve the model, we used IBM CP optimizer [45], which handles lexicographic optimization. It was run only once with the two ordered objectives and thus provides an upper bound for both objective functions. The time needed by the solver to get stable objective values was two hours, and the solver could not deduce any lower bound.

We use IBM CPLEX version 22.1.1.0 [46] to solve the compact formulation. The solution minimizing the first objective is given as MIP start to solve the second objective. If some machines are not used in the input solution, they are removed from the problem as a preprocessing. We used the default parameters of the MIP solver.

We used several existing tools to develop our algorithms. For the global branch-and-price framework (branching, column generation process including stabilization using the in-out technique [47]), we used BapCod [48], a generic branch-and-price library developed by our team. To solve the pricing subproblem, we modified the labeling algorithm developed in VRPSolver [38]. We needed to implement the selection resource type introduced above, which ensures that a suitable machine configuration can be chosen for the jobs assigned to the same machine. Finally, the restricted master problem is solved by the same version of Cplex solver as above. As the running time of the exact column-and-cut separation is large, we use it only in the testing phase for calculating valid lower bounds, which are then used to estimate the quality of the solutions obtained by heuristics. For this reason, we do not perform branching and terminate the execution after the completion of column-and-cut generation.

We recall the different methods tested in Table 3. For each method, we introduce the short name used in the computational results table and the reference to the method description in the paper.

Abbreviation	Method	Comment
CP	UB-5m: CP approach (5 minutes)	Described in Appendix A
	UB-2h: CP approach (2 hours)	Described in Appendix A
Compact	Compact model (no improvements)	(2b)–(2o)
Impr. Compact	Compact model (improved)	(2b)–(2o), (4),(5),(6),(7)
Path formulation	LB1: Column generation	(11a)–(11d) (no bipartite cover)
	LB2: Column generation	(11a)–(11d) (bipartite cover)
	UB: Diving heuristic	(11a)–(11d) (heuristic dominance, diving)

Table 3: Methods tested in our experiments

	CP (ref)		Compact		Impr. Compact		Path formulation					
	($\sum U_i$, nb m)		($\sum U_i$, nb m)		($\sum U_i$, nb m)		($\sum U_i$, nb m)			time		
Inst.	UB-2h	UB-5m	LB	UB	LB	UB	LB1	LB2	UB	LB1	LB2	UB
1	(**50**,**25**)	(**50**,**25**)	(50,23)	(**50**,**25**)	(50,23)	(**50**,**25**)	(50,25)	(50,25)	(**50**,27)	50	52	13
2	(**21**, 22)	(**21**,23)	(18,19)	(22,20)	(21,20)	(**21**,21)	(21,20)	(21,20)	(**21**,**20**)	10	10	2
3	(**47**, **46**)	(**47**,52)	(16,41)	(**47**,48)	(47,42)	(**47**,48)	(47,46)	(47,46)	(**47**,**46**)	119	99	9
4	(**60**, **43**)	(**60**,**43**)	(23,37)	(**60**,62)	(60,41)	(**60**,**43**)	(60,43)	(60,43)	(**60**,**43**)	1165	1148	12
5	(**88**, 31)	(90,82)	(-,-).	(-,-)	(84,-)	(-,-)	(86,24)	(86,24)	(**86**,26)	1087	687	30
6	(**55**, 31)	(**55**,44)	(52,-)	(-,-)	(54,-)	(-,-)	(55,29)	(55,29)	(**55**,31)	304	303	19
7	(64, 43)	(74,50)	(32,-)	(-,-)	(54,-)	(-,-)	(62,41)	(62,41)	(**62**,42)	311	313	26
8	(**31**, 56)	(32,69)	(-,-).	(-,-)	(28,-)	(-,-)	(31,43)	(31,43)	(**31**,48)	1265	1251	176
9	(97, 94)	(134,116)	(-,-).	(-,-)	(91,-)	(-,-)	(92,84)	(92,84)	(**92**,86)	800	797	116
10	(89, 63)	(122,70)	(28,-)	(-,-)	(82,-)	(-,-)	(88,57)	(88,57)	(**88**,58)	556	546	119
11	(126,122)	(-,-)	(-,-).	(-,-)	(124,-)	(-,-)	(125,91)	(125,91)	(**125**,93)	8784	8690	190
3m	(**47**,**46**)	(**47**,53)	(16,41)	(**47**,48)	(47,42)	(**47**,48)	(47,46)	(47,46)	(**47**,**46**)	173	124	9
4m	(**60**,**43**)	(**60**,**43**)	(2,-).	(-,-)	(60,41)	(**60**,**43**)	(60,43)	(60,43)	(**60**,**43**)	421	422	13
5m	(**86**,32)	(91,81)	(41,-)	(-,-)	(84,-)	(-,-)	(86,30)	(86,30)	(**86**,32)	97	119	132
7m	(**55**, 31)	(**55**,39)	(52,-)	(-,-)	(54,-)	(-,-)	(55,29)	(55,29)	(**55**,31)	256	254	24
6m	(**62**,**43**)	(**62**,**43**)	(35,-)	(-,-)	(54,-)	(-,-)	(62,43)	(62,43)	(**62**,44)	280	263	50
8m	(33,55)	(35,71)	(-,-).	(-,-)	(28,-)	(-,-)	(31,44)	(31,44)	(**31**,49)	614	608	178
9m	(98,89)	(135,116)	(-,-).	(-,-)	(91,-)	(-,-)	(92,83)	(92,83)	(**92**,85)	373	382	53
10m	(90,66)	(145,70)	(28,-)	(-,-).	(82,-)	(-,-)	(88,55)	(88,55)	(**88**,57)	548	546	88
11m	(127,137)	(-,-)	(-,-)	(-,-)	(124,-)	(-,-)	(125,96)	(125,96)	(**125**,97)	6552	6514	150

Table 4: Summary of the results obtained by the different methods described in the document

6.2. Numerical results on industrial instances

Our experiments are summarized in Table 4. All computational experiments were run on an Intel(R) Core(TM) i7-9700K CPU @ 3.60GHz. We report the reference value obtained by the CP optimizer in 7200 seconds (column CP ref / UB-2h), and in 300 seconds (column CP ref / UB-5m), and the results obtained by our methods. We use the notations of Table 3 for the methods. For the compact MIP formulations, we report the best lower and upper bounds found in 7200 seconds. For the path formulation, we report the lower bounds obtained by the two versions of column generation (LB1 and LB2) and the upper bounds obtained by the diving heuristic. For instance 11, we had to run the column generation for slightly more than two hours to obtain a lower bound. The symbol "-" means the model could not compute any value. All times are reported in seconds (rounded up). Bold faces indicate optimal primal solutions.

The improvements on the compact MIP formulation significantly increase the lower bound obtained by the general-purpose MIP solver. However, this only reflects marginally on the quality of the primal

solutions. Overall, applying them improves one solution (from 62 to 43, for instance 4), produces an additional feasible solution (instance 4m), and worsen the result for one instance (from 20 to 21, for instance number 2). Even with these improvements, the compact model is only able to find feasible solutions for six instances (the six smallest of the benchmark), and no lower bounds for the second objective except for the smallest instances.

The path formulations outperform the compact formulation for producing dual and primal bounds. For dual bound, there are some slight improvements for the first objective function (for a shorter computing time), whereas the bounds are significantly improved for the second objective. In most cases, no bounds are obtained by the compact formulation, although the path formulation can find bounds in two hours for all instances except instance 11, which takes 2 hours and 24 minutes. Using the bipartite clique cover does not significantly impact the results. The same bounds are found, and the computing time is only marginally reduced. The most significant improvement occurs for instance number 5, where the total time goes from 1087 seconds to 687 seconds. For primal bounds, the difference is even more significant. The diving heuristic can find a solution for all instances of the benchmark, with a maximum total time of 3 minutes and 10 seconds (instance 11). For the first objective, the heuristic can find solutions better than those obtained by the reference CP solver after two hours of computing time. For the second objective, the diving heuristic improves the results obtained by CP, with some notable improvements for large instances (i.e., instance 11, from 122 machines to 93). When a similar time is given to the CP solver (five minutes), our diving heuristic outperforms this method, which cannot always find a feasible solution before this time limit. The diving heuristic fails to reach the results of the CP solver for only one instance (the smallest one). It appears that the linear relaxation obtained when the heuristic pricing is used is already larger than the optimal value. This is not too surprising since we know that the *aggressive* dominance rule can sometimes exclude optimal solutions.

Finally, we draw some conclusions on the problem itself. First, the second objective function (min machines) leads to a more challenging problem than minimizing the number of late jobs. This is true for all methods. The gap is larger for the column generation method, and the time needed to compute a new column quickly becomes large. Solving the problem exactly is beyond the reach of our branch-and-price method for the largest instances. Second, the existence of non-simple parameters does not make the problem harder for the column-generation method (although it seems to hurt the results of the compact model). This can be explained by the fact that selection resources do not weaken the dominance relations used in the label-setting algorithm, which tends to perform similarly on both types of instances.

From an industrial point of view, the lower bounds computed show that in most real instances, the number of late jobs is large in any solution. This comes from different deciders setting due dates independently, with no global view of the process. For several instances, the machines with the largest release dates are never used since any job assigned to one of them would be late, even if it is the first job. In our additional computational experiments below, we study how modifying the due dates impacts the difficulty of the problems and the number of machines needed.

	CP (ref)		Impr. Compact		Path formulation			
	($\sum U_i$, nb m)		($\sum U_i$, nb m)		($\sum U_i$, nb m)		time	
Inst.	UB-2h	UB-5m	LB	UB	LB	UB	LB	UB
1	(50,25)	(50,25)	(50,23)	(50,25)	(50,25)	(52,-)	64	3
2	(21,22)	(21,22)	(21,20)	(21,20)	(21,20)	(21,20)	11	2
3	(47,49)	(47,57)	(47,42)	(47,48)	(47,46)	(47,46)	131	10
4	(60,45)	(60,45)	(60,41)	(60,43)	(60,43)	(60,43)	769	13
5	(89,33)	(89,61)	(84,-)	(-,-)	(86,23)	(86,26)	805	21
6	(55,32)	(55,34)	(55,-)	(-,-)	(55,30)	(55,31)	643	28
7	(-,-)	(-,-)	(54,-)	(-,-)	(62,41)	(62,42)	293	30
8	(33,57)	(35,68)	(28,-)	(-,-)	(31,43)	(31,48)	1706	213
9	(98,95)	(132,98)	(90,-)	(-,-)	(92,84)	(92,86)	844	58
10	(91,59)	(132,59)	(82,-)	(-,-)	(88,57)	(89,58)	605	304
11	(127,138)	(-,-)	(124,-)	(-,-)	(-,-)	(125,93)	7200	219

Table 5: Summary of the computational results for instances with fewer machines

6.3. Additional computational experiments

We now report additional numerical experiments on synthetic data obtained by modifying key parameters in our real-life instances: the due dates, the number of damaging jobs, and the number of machines. To simplify the exposition, in the remainder of this section, we only report the results obtained by the CP solver, the best version of compact MIP (with all cuts), and the best version of column generation (with bipartite cover). We first focus our comments on the behavior of the different algorithms, before giving some insights about the difficulty of the various versions of the data.

We first analyze in Table 5 the results obtained on instances obtained from the industrial test bed by removing 15% of the vehicles (those with the largest release dates). Reducing the number of machines helps the MIP solver find better lower bounds with the compact formulation. This is explained by the fact that reducing the number of possible machines reduces the total number of variables, and induces a better linear relaxation. However, it does not help the compact formulation to produce practical primal solutions. The constraint programming model does not solve instance 7 anymore. For this instance, using only the minimum number of machines to find a feasible solution is mandatory. For the other instances, the primal solutions found by the solver in two hours vary from up to two units. This indicates that the heuristic used in the solver is sensitive to modifications in the data. The path formulation finds the same values as in the original instances, except for two instances: instance 1, for which the method failed to find a feasible solution for the second objective. The optimal solution of the linear relaxation is different, which triggers a different column choice in the diving algorithm. Again, this is the smallest instance, and it is not always possible to compensate a bad assignment choice for one job. For instance number 11, the solver did not converge in one hour for the first objective and could not compute a dual bound better than 0. The method begins with artificial columns and a phase 1 simplex to obtain a feasible basis. A more constrained problem means more time in this first phase, in which the dual bound is not improved.

	CP (ref)		Impr. Compact		Path formulation			
	($\sum U_i$, nb m)		($\sum U_i$, nb m)		($\sum U_i$, nb m)		time	
Inst.	UB-2h	UB-5m	LB	UB	LB	UB	LB	UB
1	(50,10)	(50, 11)	(50, 4)	(50, 12)	(50,8)	(50,12)	763	59
2	(21,18)	(21, 19)	(21, 6)	(21, 17)	(21,14)	(21,19)	25	8
3	(47,19)	(47, 28)	(47, 6)	(47, 37)	(47,16)	(47,18)	3852	47
4	(60,31)	(60, 31)	(60,14)	(60, 28)	(60,0)	(60,28)	4024	53
5	(86,29)	(88, 82)	(84, -)	(86, 58)	(86,14)	(86,21)	3692	57
6	(55,16)	(55, 29)	(54, -)	(-, -)	(55,13)	(55,16)	3972	321
7	(63,28)	(69, 50)	(52, -)	(-, -)	(62,17)	(62,24)	1182	482
8	(34,58)	(34, 57)	(28, -)	(-, -)	(30,35)	(30,41)	1960	330
9	(93,50)	(122, 116)	(90, -)	(-, -)	(92,34)	(92,41)	4347	290
10	(89,66)	(-,-)	(82, -)	(-, -)	(88,27)	(88,35)	3460	640
11	(126,126)	(238,180)	(124, -)	(-, -)	(-,-)	(125,68)	7200	1722

Table 6: Summary of the computational results for instances with non-damaging jobs only

In Table 6, we report the results obtained by relaxing the constraints related to damaging jobs. In these instances, any succession of jobs is allowed on the machines. This modification has a limited impact on the results obtained by solving the compact model. The solver only finds solutions for the four smallest instances. The gap between the lower and upper bounds is larger for the second objective. This indicates that the disjunctions between damaging jobs help tighten the linear relaxation. For the first objective, the results of the CP solver are similar to those obtained with damaging jobs. Some solution values changed for the first objective (solution for instances 5, 7, and 9 are slightly improved, while the solution for instance 7 is slightly worse). For the second objective, the CP solver struggles to find competitive solutions for the largest instances: it uses 126 vehicles for instances 11, while the best solution uses 68. The path formulation remains the best method for producing primal solutions. It still finds an optimal solution for the first objective for all instances, but the primal solution found for the second objective is worse than those obtained by the CP solver for the two smallest instances. Note that there is a significant impact on the lower bound computation. Removing the precedence constraints induced by the damaging jobs increases the number of possible paths by a wide range, and the column generation procedure is not able to converge in one hour for the second objective, for six instances, and cannot even converge for the first objective for instance number 11.

In Table 7 and Table 8, we report the results obtained for our instances when all due dates are multiplied by two and three. The MIP solver can find one additional solution (instance 4) when the values of the due dates are doubled and three additional solutions when they are tripled. However, the quality of the solutions is inferior to those obtained using other methods. The CP solver finds solutions for all instances, although their values may be far from the best primal bounds. The path formulation finds the best primal solutions for all instances. It is able to compute a dual bound for all instances, except instance 11. When it obtains a dual bound, it is always equal to the primal bound for the first

	CP (ref)		Impr. Compact		Path formulation			
	($\sum U_i$, nb m)		($\sum U_i$, nb m)		($\sum U_i$, nb m)		time	
Inst.	UB-2h	UB-5m	LB	UB	LB	UB	LB	UB
1	(17,27)	(18,27)	(9,22)	(17,28)	(17,27)	(17,27)	19	2
2	(2,23)	(2,24)	(0,20)	(2,25)	(2,22)	(2,22)	24	4
3	(19,47)	(19, 50)	(17, 42)	(17,50)	(17,47)	(17,47)	601	9
4	(6,51)	(26,90)	(0,0)	(6,60)	(3,45)	(3,46)	6741	23
5	(8,38)	(19,56)	(0,19)	(2,48)	(2,35)	(2,36)	231	94
6	(5,35)	(10,51)	(1,0)	(-,-)	(5,32)	(5,34)	1754	36
7	(0,47)	(0,50)	(0,0)	(-,-)	(0,45)	(0,45)	346	85
8	(0,54)	(4,63)	(0,0)	(-,-)	(0,40)	(0,41)	921	31
9	(45,86)	(78, 116)	(22,0)	(-,-)	(29,79)	(29,80)	1292	65
10	(5,65)	(52,70)	(0,0)	(-,-)	(2,58)	(2,59)	3070	939
11	(24,161)	(365,180)	(3,0)	(-,-)	(-,-)	(3,102)	7200	154

Table 7: Summary of the computational results for instances with large due dates (twice the initial ones)

objective, and the gap is never greater than two for the second objective.

We now conclude this section with some comments about the difficulty of the different variants of data sets. We first remark that reducing the number of machines does not decrease the number of late jobs, which confirms that machines with large release dates are not useful with tight due dates. Our experiments show that damaging jobs does not significantly modify the number of late jobs but significantly impact the number of machines needed. This is in line with our previous observations. Our modifications of the due dates led to the most important modifications in the structure of the problem. The number of late jobs decreased significantly for each instance, which was expected. In Table 8, for nine instances out of 11, the number of late jobs is zero in an optimal solution. An interesting observation is that the number of machines increases significantly when due dates increase. Larger due dates allow solutions where jobs are assigned to machines with large release dates, which were never used with our original industrial instances.

	CP (ref)		Impr. Compact		Path formulation			
	($\sum U_i$, nb m)		($\sum U_i$, nb m)		($\sum U_i$, nb m)		time	
Inst.	UB-2h	UB-5m	LB	UB	LB	UB	LB	UB
1	(0,25)	(1,27)	(0,22)	(0,25)	(0,25)	(0,25)	13	4
2	(2,23)	(0,24)	(0,20)	(0,23)	(0,21)	(0,21)	14	2
3	(7,48)	(9,48)	(5,43)	(5,49)	(5,46)	(5,48)	3666	11
4	(0,48)	(13,89)	(0,42)	(0,44)	(0,43)	(0,43)	603	9
5	(0,29)	(0,37)	(0,0)	(53,51)	(0,27)	(0,29)	68	13
6	(0,31)	(0,35)	(0,0)	(8,46)	(0,29)	(0,31)	436	25
7	(0,43)	(0,45)	(0,0)	(-,-)	(0,41)	(0,41)	625	32
8	(0,45)	(0,74)	(0,0)	(24,69)	(0,37)	(0,37)	864	24
9	(21,95)	(48,116)	(11,0)	(-,-)	(17,71)	(17,72)	3739	50
10	(0,62)	(29,70)	(0,0)	(-,-)	(0,58)	(0,58)	1188	61
11	(22,179)	(22,180)	(0,0)	(-,-)	(-,-)	(0,94)	7200	148

Table 8: Summary of the computational results for instances with large due dates (three times the initial ones)

7. Conclusion

In this paper, we have introduced a new scheduling problem that arises in the car industry. An original feature of the problem is that machines must be configured to accept the different jobs, and a machine has a unique configuration that cannot be changed over time. To the best of our knowledge, this type of configuration has not been considered in the literature. We have shown that even the feasibility version of the problem is NP-complete and proposed a precise characterization of the cases where the job/configuration compatibility constraints are equivalent to disjunctions between pairs of jobs. Then, we proposed several models to solve the problem in practice. A compact MIP model uses a natural formulation where configurations of the machines are considered explicitly. In our path formulation, configurations are a by-product of the job assignment. This required the adaptation of a state-of-the-art labeling algorithm to exclude solutions containing subsets of jobs for which there is no feasible machine configuration. Computational experiments show that the path-flow formulation can close the gap for all industrial instances when the number of late jobs has to be minimized. When the number of machines is optimized, our method can find feasible solutions whose values are better than those obtained by a CP model used by the company. This justifies that advanced methods are needed to solve the problem effectively.

The exact resolution of the problem through branch-and-cut-and-price for the industrial instances is out of reach for our current methodologies, but our path-flow model can also produce dual bounds for all test cases. Although the gap is generally small, the time needed for each pricing phase requires a different approach to solve optimally the largest instances.

References

[1] H. L. Bodlaender, K. Jansen, G. J. Woeginger, Scheduling with incompatible jobs, Discrete Applied Mathematics 55 (3) (1994) 219–232. doi:https://doi.org/10.1016/0166-218X(94)90009-4.
URL https://www.sciencedirect.com/science/article/pii/0166218X94900094

[2] V. L. de Lima, C. Alves, F. Clautiaux, M. Iori, J. M. V. de Carvalho, Arc flow formulations based on dynamic programming: Theoretical foundations and applications, European Journal of Operational Research 296 (1) (2022) 3–21. doi:10.1016/j.ejor.2021.04.024.

[3] A. Aubry, A. Rossi, M. L. Espinouse, M. Jacomino, Minimizing setup costs for parallel multi-purpose machines under load-balancing constraint, European Journal of Operational Research 187 (3) (2008) 1115–1125. doi:10.1016/j.ejor.2006.05.050.

[4] P. Brucker, Scheduling Algorithms, Springer Berlin, Heidelberg, 2007. doi:https://doi.org/10.1007/978-3-540-69516-5.

[5] R. de Freitas Rodrigues, M. C. Dourado, J. L. Szwarcfiter, Scheduling problem with multi-purpose parallel machines, Discrete Applied Mathematics 164 (2014) 313–319, combinatorial Optimization. doi:https://doi.org/10.1016/j.dam.2011.11.033.
URL https://www.sciencedirect.com/science/article/pii/S0166218X11004872

[6] K. Grage, K. Jansen, K.-M. Klein, An EPTAS for Machine Scheduling with Bag-Constraints, in: The 31st ACM Symposium on Parallelism in Algorithms and Architectures, SPAA '19, Association for Computing Machinery, New York, NY, USA, 2019, pp. 135–144. doi:10.1145/3323165.3323192.

[7] A. Rossi, A robustness measure of the configuration of multi-purpose machines, International Journal of Production Research 48 (4) (2010) 1013–1033. doi:10.1080/00207540802473997.

[8] G. Centeno, R. L. Armacost, Minimizing makespan on parallel machines with release time and machine eligibility restrictions, International Journal of Production Research 42 (6) (2004) 1243–1256. doi:10.1080/00207540310001631584.
URL https://doi.org/10.1080/00207540310001631584

[9] M. L. Pinedo, Scheduling, Springer New York, NY, 2012. doi:https://doi.org/10.1007/978-1-4614-2361-4.

[10] J. Y.-T. Leung, C.-L. Li, Scheduling with processing set restrictions: A survey, International Journal of Production Economics 116 (2) (2008) 251–262. doi:https://doi.org/10.1016/j.ijpe.2008.09.003.
URL https://www.sciencedirect.com/science/article/pii/S0925527308003071

[11] J. Y.-T. Leung, C.-L. Li, Scheduling with processing set restrictions: A literature update, International Journal of Production Economics 175 (2016) 1–11. doi:https://doi.org/10.1016/j.ijpe.2014.09.038.
URL https://www.sciencedirect.com/science/article/pii/S0925527316000190

[12] J. Lamothe, F. Marmier, M. Dupuy, P. Gaborit, L. Dupont, Scheduling rules to minimize total tardiness in a parallel machine problem with setup and calendar constraints, Computers & Operations Research 39 (6) (2012) 1236–1244, special Issue on Scheduling in Manufacturing Systems. doi:https://doi.org/10.1016/j.cor.2010.07.007.
URL https://www.sciencedirect.com/science/article/pii/S0305054810001383

[13] M. Nattaf, S. Dauzère-Pérès, C. Yugma, C.-H. Wu, Parallel machine scheduling with time constraints on machine qualifications, Computers & Operations Research 107 (2019) 61–76. doi:https://doi.org/10.1016/j.cor.2019.03.004.
URL https://www.sciencedirect.com/science/article/pii/S0305054819300589

[14] L. Mönch, C. Yugma, Scheduling jobs on parallel machines with qualification constraints, in: 2015 IEEE International Conference on Automation Science and Engineering (CASE), 2015, pp. 657–658. doi:10.1109/CoASE.2015.7294154.

[15] M. Fu, M. Haghnevis, R. Askin, J. Fowler, M. Zhang, Machine qualification management for a semiconductor back-end facility, in: Proceedings of the 2010 Winter Simulation Conference, 2010, pp. 2486–2492. doi:10.1109/WSC.2010.5678944.

[16] A. Bitar, S. Dauzère-Pérès, C. Yugma, R. Roussel, A memetic algorithm to solve an unrelated parallel machine scheduling problem with auxiliary resources in semiconductor manufacturing, Journal of Scheduling 19 (4) (2016) 367–376. doi:10.1007/s10951-014-0397-6.

[17] J.-F. Chen, T.-H. Wu, Total tardiness minimization on unrelated parallel machine scheduling with auxiliary equipment constraints, Omega 34 (1) (2006) 81–89. doi:https://doi.org/10.1016/j.omega.2004.07.023.
URL https://www.sciencedirect.com/science/article/pii/S0305048304001197

[18] Z. Zhang, X. Gong, X. Song, Y. Yin, B. Lev, J. Chen, A column generation-based exact solution method for seru scheduling problems, Omega 108 (2022) 102581. doi:10.1016/j.omega.2021.102581.

[19] Z. Zhang, X. Song, X. Gong, Y. Yin, B. Lev, X. Zhou, An exact quadratic programming approach based on convex reformulation for seru scheduling problems, Naval Research Logistics (NRL) 69 (8) (2022) 1096–1107. doi:10.1002/nav.22078.

[20] Z. Zhang, X. Song, H. Huang, X. Zhou, Y. Yin, Logic-based Benders decomposition method for the seru scheduling problem with sequence-dependent setup time and DeJong's learning effect, European Journal of Operational Research 297 (3) (2022) 866–877. doi:10.1016/j.ejor.2021.06.017.

[21] D. R. Page, R. Solis-Oba, Makespan minimization on unrelated parallel machines with a few bags, Theoretical Computer Science 821 (2020) 34–44. doi:10.1016/j.tcs.2020.03.013.

[22] K. Jansen, A. Lassota, M. Maack, T. Pikies, Total Completion Time Minimization for Scheduling with Incompatibility Cliques, Proceedings of the International Conference on Automated Planning and Scheduling 31 (2021) 192–200. doi:10.1609/icaps.v31i1.15962.

[23] T. Pikies, K. Turowski, M. Kubale, Scheduling with complete multipartite incompatibility graph on parallel machines: Complexity and algorithms, Artificial Intelligence 309 (2022) 103711. doi:10.1016/j.artint.2022.103711.

[24] N. Bianchessi, E. Tresoldi, A stand-alone branch-and-price algorithm for identical parallel machine scheduling with conflicts, Computers & Operations Research 136 (2021) 105464. doi:10.1016/j.cor.2021.105464.

[25] J. Van den Akker, C. Hurkens, M. Savelsbergh, Time-Indexed Formulations for Machine Scheduling Problems: Column Generation, INFORMS Journal on Computing 12 (2) (2000) 111–124. doi:10.1287/ijoc.12.2.111.11896.

[26] D. Kowalczyk, R. Leus, A branch-and-price algorithm for parallel machine scheduling using ZDDs and generic branching, INFORMS Journal on Computing 30 (4) (2018) 768–782. doi:10.1287/ijoc.2018.0809.

[27] D. Oliveira, A. Pessoa, An Improved Branch-Cut-and-Price Algorithm for Parallel Machine Scheduling Problems, INFORMS Journal on Computing 32 (1) (2020) 90–100. doi:10.1287/IJOC.2018.0854.

[28] T. Bulhões, R. Sadykov, A. Subramanian, E. Uchoa, On the exact solution of a large class of parallel machine scheduling problems, Journal of Scheduling 23 (4) (2020) 411–429. doi:10.1007/s10951-020-00640-z.

[29] A. Kramer, M. Iori, P. Lacomme, Mathematical formulations for scheduling jobs on identical parallel machines with family setup times and total weighted completion time minimization, European Journal of Operational Research 289 (3) (2021) 825–840.

[30] G. Song, R. Leus, Parallel machine scheduling under uncertainty: Models and exact algorithms, INFORMS Journal on Computing 34 (6) (2022) 3059–3079.

[31] O. Shahvari, R. Logendran, M. Tavana, An efficient model-based branch-and-price algorithm for unrelated-parallel machine batching and scheduling problems, Journal of Scheduling 25 (5) (2022) 589–621. doi:10.1007/s10951-022-00729-7.

[32] V. de Lima, C. Alves, F. Clautiaux, M. Iori, J. Valério de Carvalho, Arc flow formulations based on dynamic programming: Theoretical foundations and applications, European Journal of Operational Research 296 (1) (2022) 3–21. doi:10.1016/j.ejor.2021.04.024.

[33] A. Kramer, M. Dell'Amico, D. Feillet, M. Iori, Scheduling jobs with release dates on identical parallel machines by minimizing the total weighted completion time, Computers and Operations Research 123 (2020). doi:10.1016/j.cor.2020.105018.

[34] J. Martinovic, N. Strasdat, J. V. de Carvalho, F. Furini, A combinatorial flow-based formulation for temporal bin packing problems, European Journal of Operational Research 307 (2) (2023) 554–574.

[35] R. Graham, E. Lawler, J. Lenstra, A. Kan, Optimization and approximation in deterministic sequencing and scheduling: A survey, in: P. Hammer, E. Johnson, B. Korte (Eds.), Discrete Optimization II, Vol. 5 of Annals of Discrete Mathematics, Elsevier, 1979, pp. 287–326. doi:10.1016/S0167-5060(08)70356-X.

[36] S. Booth, S. Lueker, Testing for the consecutive ones property, interval graphs, graph planarity using pq-trees algorithms, Journal of Computer and System Sciences 13 (1976) 335–379.

[37] A. Tucker., A structure theorem for the consecutive 1's property., Journal of Combinatorial Theory, Series B 12 (1972) 153–162.

[38] A. Pessoa, R. Sadykov, E. Uchoa, F. Vanderbeck, A generic exact solver for vehicle routing and related problems, Mathematical Programming 183 (2020) 483–523.

[39] R. Sadykov, E. Uchoa, A. Pessoa, A bucket graph–based labeling algorithm with application to vehicle routing, Transportation Science 55 (1) (2021) 4–28.

[40] D. Pecin, A. Pessoa, M. Poggi, E. Uchoa, H. Santos, Limited memory rank-1 cuts for vehicle routing problems, Operations Research Letters 45 (3) (2017) 206 – 209.

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

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

[43] R. Roberti, A. Mingozzi, Dynamic ng-path relaxation for the delivery man problem, Transportation Science 48 (3) (2014) 413–424.

[44] R. Sadykov, F. Vanderbeck, A. Pessoa, I. Tahiri, E. Uchoa, Primal heuristics for branch-and-price: the assets of diving methods, INFORMS Journal on Computing 31 (2) (2019) 251–267.

[45] IBM ILOG CPLEX Optimization Studio CP Optimizer User's Manual, https://www.ibm.com/docs/en/SSSA5P_12.8.0/ilog.odms.studio.help/pdf/usrcpoptimizer.pdf.

[46] Cplex 20.1 manual, https://www.ibm.com/docs/en/icos/20.1.0.

[47] A. Pessoa, R. Sadykov, E. Uchoa, F. Vanderbeck, In-out separation and column generation stabilization by dual price smoothing, in: V. Bonifaci, C. Demetrescu, A. Marchetti-Spaccamela (Eds.), Experimental Algorithms, Springer Berlin Heidelberg, Berlin, Heidelberg, 2013, pp. 354–365.

[48] R. Sadykov, F. Vanderbeck, BaPCod - a generic branch-and-price code, Technical report, Inria Bordeaux Sud-Ouest (Nov. 2021).
URL https://hal.inria.fr/hal-03340548

Appendix A. A constraint programming model

We now present a constraint programming model, which was used to produce heuristic solutions for the problem. This model is a simplification of the model used by the company in practice. In the real-life model, additional secondary objectives and constraints are considered, such as time windows on some jobs. Our simplifications do not change the main structure of the problem but rather simplify the exposition. The solver used was CP Optimizer (CPO), and OPL (Optimization Programming Language) was employed as a modeling language. We report this model using the vocabulary of CP. We use *optional job variables* to model that jobs are assigned to one machine only. Interval variable job_i corresponds with the interval related to the start and completion time of job i (the interval size has to be equal to p_i). If i is assigned to machine m, the optional interval assignedJob_{im} is equal to job_i. Otherwise, it is deactivated. Binary variable α_{mgv} has the same meaning as the one used in our linear integer program. We use two sets of expressions deduced from the variable values: u_i indicates that job i is late, and z_m indicates that machine m is used.

$$
(P)\begin{cases}
\min \textbf{staticLex}(\sum_{i\in\mathcal{J}} u_i \ , \ \sum_{m\in\mathcal{M}} z_m) \\
\textbf{subject to :} \\
\textbf{alternative}(\text{job}_i, [\text{assignedJob}_{im}]_{m\in\mathcal{M}}) \ \forall i \in \mathcal{J} \ \text{(C1)} \\
\textbf{noOverlap}(\text{sequence}_m, \ \text{A}, \ \textbf{true}) \ \forall m \in \mathcal{M} \ \text{(C2)} \\
\sum_{v\in\mathcal{V}(g)} \alpha_{mgv} = 1 \ \forall m \in \mathcal{M}, \forall g \in \mathcal{G} \ \text{(C3)} \\
\textbf{presenceOf}(\text{assignedJob}_{im}) - \sum_{v\in\mathcal{V}_i(g)} \alpha_{mgv} \leq 0 \ \forall i \in \mathcal{J}, \forall m \in \mathcal{M}, \forall g \in \mathcal{G} \ \text{(C4)} \\
\textbf{presenceOf}(\text{assignedJob}_{im}) + \textbf{presenceOf}(\text{assignedJob}_{jm}) \leq 1 \ \forall i \in \mathcal{J}, \forall j \in \mathcal{J}^o(i), \forall m \in \mathcal{M} \ \text{(C5)} \\
\textbf{endOf}(\text{assignedJob}_{im}, T) \geq \textbf{endOf}(\text{assignedJob}_{jm}, 0) \ \forall i \in \mathcal{J}, \forall j \in \mathcal{J}^+(i), \forall m \in \mathcal{M} \ \text{(C6)} \\
u_i = (\textbf{endOf}(\text{job}_i) > d_i) \ \forall i \in \mathcal{J} \ \ \text{(C7)} \\
z_m = \max_{i\in\mathcal{J}} \ \textbf{presenceOf}(\text{assignedJob}_{im}) \ \forall m \in \mathcal{M} \ \ \text{(C8)} \\
\\
\textbf{Decision variables :} \\
\textbf{interval} \ \text{job}_i \ \textbf{size} \ p_i \ \forall i \in \mathcal{J} \ \text{(C9)} \\
\textbf{interval} \ \text{assignedJob}_{im} \ \textbf{optional} \ \subseteq [r_m, T] \quad \forall i \in \mathcal{J} \ , \forall m \in \mathcal{M} \ \text{(C10)} \\
\textbf{sequence} \ \text{sequence}_m = [\text{assignedJob}_{im}]_{i\in\mathcal{J}} \ \ \text{(C11)} \\
\textbf{boolean} \ \alpha_{mgv} \in \{0,1\} \quad \forall m \in \mathcal{M}, \forall g \in \mathcal{G}, \forall v \in \mathcal{V}(g) \ \ \text{(C12)}
\end{cases}
$$

The objective function is to minimize lexicographically the number of late jobs and the number of machines using the staticLex directive. Constraints **(C1)** ensure that each job is assigned to one single machine (the global constraint **alternative** ensures that the optional interval assignedJob_{im} is present on machine m and has the same size as the interval job_i which is the duration of job i if i is assigned to m, the optional interval is absent otherwise) Constraints **(C2)** ensure that on each machine, the jobs do not overlap and time-lags between jobs are respected (**A** is the distance matrix between tests, and **true** indicates time-lags are only enforced for direct successors). Constraints **(C3)** say that for each parameter

and each machine, exactly one value has to be selected. Constraints **(C4)** forbid assigning a job i to machine m if at least one value on a parameter of the latter is not compatible with i (**presenceOf** is an operator that indicates if an optional interval is present or not). Constraints **(C5)** ensure that no two conflicting jobs can be assigned to the same machine. Constraints **(C6)** indicate that a job i cannot be followed by a job in $\mathcal{J}^+(i)$. Constraints **(C7)** define the binary variable u_i on each job i: it has to be 1 if i is late, 0 otherwise (d_i is the due date of job i). Constraints **(C8)** define the binary variable z_m on each machine m: it has to be 1 if m is used, 0 otherwise. Constraint **(C9)** is the declaration of each job i as an **interval** variable in CPO with size p_i, which is the duration of the job. Constraint **(C10)** is the declaration of an **optional interval** variable on each job i and each machine m in CPO. The interval is present if i is assigned to m. It is absent otherwise (has zero as a size). The optional interval of each job and each machine is only contained in $[r_m, T]$. Constraint **(C11)** is the declaration of a **sequence** variable on each machine m: an ordered list of the intervals assignedJob_{im}. Constraint **(C12)** is the declaration of the allocation variables of each parameter's value to each machine.