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 scheduling problem, which occurs in the context of the automobile industry. Given a set of machines, and a set of jobs, we seek a fixed configuration for each machine (i.e. a set of values for different parameters), and an assignment of jobs to machines along the time horizon that respects compatibility constraints between jobs and machine configurations. Two objectives are optimized lexicographically: the number of late jobs, and the number of machines used. 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. Then we propose a mathematical model for this problem, and a reformulation into a path-flow formulation. We use a refined labelling algorithm embedded in a column-and-row generation algorithm to produce primal and dual bounds from this formulation. We conducted computational experiments with industrial data from Renault, and compared our results with those obtained by solving a constraint programming model provided by the company. Our approach finds better solutions than those obtained by the company, and proves the optimality for all instances of our benchmark for the first objective function. We also obtain 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 model of vehicle has just been conceived, it needs to undergo a series of tests before being released on the market. The validation tests carried out in the various facilities are numerous, and are repeated on the different mechanical parts of the vehicles (thermal, acoustic, driving, sound insulation, safety or endurance tests for example). This phase is costly, since specific vehicles called *prototypes* are built exclusively for test purposes. Each test requires a certain *configuration* (i.e. specific values for different parameters) and thus several prototypes of the new model, with different configurations, are needed. The goal of the company is to minimize the number of tests realized after their due dates, and then the number of prototypes to manufacture, in a lexicographic way.

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

From an operations research point of view, the problem can be seen as a two-step optimization problem. In the first step, a configuration is chosen for each prototype vehicle, i.e. a value is chosen for each possible parameter. In the second step, a sequence of jobs is assigned to each vehicle. A job can only be assigned to a prototype if the configuration of the latter is compatible. Since vehicles are produced and tested in different facilities, the job assignment problem can be either seen as a vehicle routing problem, or a scheduling problem with time lags. As time lags are short compared to the test durations in our instances, the scheduling vocabulary is adopted in the paper: we use the term machine for the prototypes, and jobs for the tests.

Although the notion of machine and job compatibility exists in the scheduling literature, we did not find a work where machine configurations have to be decided, cannot be modified along the time horizon, and where some configurations are forbidden. In *multi-purpose* scheduling [9], parallel machines are considered, and each machine can only be assigned to a subset of jobs (see the surveys of [13, 14]). A similar concept is studied in [7], under the name *machine eligibility restrictions*. In [7, 9], the compatibility relations between jobs and/or machines are given as an instance parameter and their definition is not part of the decision variables. Similar concepts have also been proposed under various names: *auxiliary equipment constraints* [8], *machine flexibility constraints* [12], *eligibility restrictions* [21], or *multipurpose machines* [6]. Our problem also relates to scheduling with *machine qualification* [11, 15, 16]. In these problems, each machine has to be *qualified* to process certain jobs, which incurs an additional cost, but a machine can be qualified for any set of jobs. Our problem also generalizes scheduling problem with incompatibilities [4], where some pairs of jobs cannot be scheduled on the same machine (this claim is discussed in more detail in the following section). Vehicle configuration also occurs in vehicle routing problems [26]. In this latter case, the configuration has an impact on the capacity and the cost of the vehicles, but does not forbid directly the assignment of some customers.

In this document, we first introduce and characterize our new industrial problem: we propose a formal definition and the proof that finding a feasible solution for the problem is already NP-complete. Then we discuss some useful properties, which allow to reduce the size of the search space in optimization algorithms, and characterize the cases where the configuration constraints become equivalent to job incompatibilities, as defined in [4]. We then propose two MILP models and a matheuristic to solve the problem. The solutions obtained by these methods are compared with those obtained by a constraint programming model previously used by the industrial partner.

An originality of our work is to address the problem by the means of a path-flow formulation. Network-flow formulations (see e.g. [10]) have already shown their effectiveness to solve hard parallel machine scheduling problems. In our path-flow model, the sequence of jobs assigned to a vehicle corresponds with a path in a network, where nodes are related to states representing a partial solution for a vehicle, and arcs correspond to decisions of assigning a job to a vehicle in this state. Configurations are thus only considered implicitly, *i.e.* they are a byproduct of the set of tasks assigned to each vehicle. To satisfy the compatibility constraints, we have to embed additional information into the states to avoid assigning a set of tasks to a vehicle, for which no possible configuration is possible. These constraints are

embedded into a labelling algorithm, which is used to produce feasible assignments of jobs to a machine. From an algorithmic point of view, this calls for a new specific labelling process, where specific resource consumption and dominance checks have to be designed to account for this specificity.

To obtain primal solutions even for the largest instances of the benchmark, we propose a matheuristic, which is devised by restricting the set of solutions to those where jobs are sequenced in increasing order of due date on each machine. Because of the sequence-dependent lags, this restriction may exclude all optimal solutions, but it is in line with the objective of minimizing the number of late jobs. This restriction spares us from imposing elementarity constraints in the labelling algorithm, thus reducing drastically the number of non-dominated labels.

We conducted computational experiments on real-life instances used by the industrial partner. For each test case, we tested our compact model, and our diving heuristic. We also compute dual bounds using a column-and-cut generation algorithm, to evaluate the quality of the heuristic. We compared our results to those obtained by a constraint programming (CP) model used by the industrial partner. The network reformulation always finds solutions that are at least as good as the CP solver (except for the smallest instance), improve the results for the large instances, and is able to prove the optimality of several solutions.

The paper is organized as follows. In section 2, we present formally the problem, state some useful properties, and propose a first compact MILP model. In section 3 we propose a path-flow formulation for the problem, and algorithms to solve it efficiently. Computational experiments are reported in section 4.

2. Problem definition and properties

2.1. Industrial problem

The problem we are considering is to optimise the configuration and use of some prototype vehicles. Each prototype car is manufactured in a production facility, and then sent sequentially in different facilities where it undergoes some tests. The location at which each test has to be performed is part of the data. At each instant, a vehicle cannot undergo more than one test, but several tests can be run simultaneously on different vehicles in the same location.

The purpose of a test is to validate some properties of the new vehicle. The same car model can exist in different configurations, and 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. manual or electrical sunroof may be equivalent for one test, but not for another one). The value for each possible parameter is decided when the vehicle is manufactured, and cannot be changed afterwards. There are no forbidden pairs of parameter values, as long as the vehicle has exactly one value for each parameter. In addition, some tests damage the vehicles. Once a damaging test has been performed, only damaging tests can be performed on the vehicle. A majority of damaging tests are called *destructive* tests and are considered to damage the whole vehicle, no test can be run after them on the same vehicle (even other damaging ones).

All prototype cars are built and configured in the same facility, before being sent to other facilities for testing. The factory that builds the prototypes has a fixed production capacity and only produces a limited number of vehicles at each time period. If a vehicle has to be moved from one facility to another (either from the production plant to a test facility, or from a test facility to another), it will take one to several periods of time, depending on the distance between the two facilities.

The objective is to find a configuration for each vehicle, and an assignment of all prescribed tests to these vehicles in such a way that all tests are realized, and a test is only performed if the configuration of the vehicle allows it.

In the following section, we give a formal definition of the problem using a scheduling vocabulary. From now on, vehicles will be called *machines*, tests will be called *jobs*. The vector of parameter values chosen for each prototype vehicle will be called *configuration*. The time needed to travel from one facility to another will be modelled 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} the time needed to convey the vehicle from the factory to the facility assigned to i. The set of jobs is partitioned into three subsets $\mathcal{J}^1$ (normal 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 task $i \in \mathcal{J}$ to a time interval $[s_i, s_i + p_i[$ and a machine m in such a way that*

- *no two distinct jobs are processed by the same machine at the same time,*
- *no job is processed by a machine before the availability date of the latter,*
- *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*

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

(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.

- *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 on which at least one job is assigned).

Problem 2 (Minimum number of late jobs problem)**.** *The minimum number of late jobs problem consists in 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 vehicles problem consists in 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. 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.

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 $j \in \mathcal{J}^1$.

2.3. Problem properties

The *Minimum number of machines problem* is clearly 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

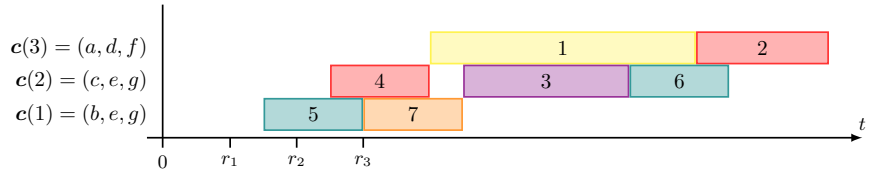

Figure 2: An example of solution for the instance of figure 1. Each rectangle represents a job, and each line is related to a machine. The release dates of the different machines are reported on the horizontal axis. The first job i on each machine m is scheduled at time $r_m + A_{0i}$. The configuration 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 scheduled last on its machine, as imposed by the problem definition. We do not picture machine 4, which is not used.

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-partitioning problem, which is known to be NP-hard, is a special 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. The problem clearly 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 a instance to Problem 1 as follows. The set of parameters is a singleton $\{1\}$, and its set of possible values $\mathcal{V}(1) = \{1, \dots, q\}$ (one for each possible member of family $\mathcal{C}$). We denote by $v(g)$ the parameter corresponding with member g of the family. For each element e of $\mathcal{E}$, we create a job j, with $p_j = 0$, $d_j = +\infty$, and $\mathcal{V}_j(1) = \{v_i : e \in \mathcal{C}_i\}$. We set distance A_{ij} to 0 for any pair (i, j), and $\mathcal{J}^2 = \mathcal{J}^3 = \emptyset$. The number of machines is equal to M, and for each machine $m = 1, \dots, M$, $r_m = 0$.

If there is a solution to the obtained feasibility problem using M machines, then one can obtain a solution for set-covering by taking the M sets that are related to those corresponding with the configuration value of each machine.

If there is a solution to the set covering problem, then a solution for the feasibility problem can be obtained by collecting the elements $m = 1, \dots, M$ used in the solution, assigning the corresponding value to the configuration for machine m, and assigning to m the jobs corresponding with the elements included in $\mathcal{C}_m$ (and removing duplicate values). □

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 for $P||\sum_i U_i$ do not apply anymore. We prove below 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 solutions where no job can be scheduled earlier without modifying other parts of the solution.

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). Moreover under the assumption that a machine is horizon-unconstrained, the two following propositions are true (we omit the proofs, which just use the fact that the two jobs i and j can be swapped with no impact on other parts of the solution).

Proposition 2. *If machine m is horizon-unconstrained, for any two jobs $i \in \mathcal{J}$, $j \in \mathcal{J} \setminus \mathcal{J}^+(i)$ such that $A_{ik} = A_{jk}$ and $A_{ki} = A_{kj}$ for all $k \neq i, j$, and $d_i > d_j$ and i, there is an optimal solution where i is not assigned just before j on machine m.*

We can also benefit from the fact that data are structured (i.e. matrix A reflects distances between facilities, and thus jobs assigned to the same facility are equivalent in terms of distances with other jobs).

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 $k \neq i, j$ and $d_j < d_i$ then there is an optimal solution 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.*

2.4. Simple parameters

We now study the structure of the configuration constraints, and characterize the case 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, we say that two jobs are incompatible when $\mathcal{V}_i(g) \cap \mathcal{V}_j(g) = \emptyset$. For some parameters, checking pairwise compatibility is sufficient to ensure that the configuration constraint is satisfied. 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 ensure 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 useful to detect simple parameters, because it provides an alternative way of ensuring compatibility by simply 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 in the remainder of this section

we consider only one parameter. 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 lines 1, 2 and 3 represent values a, b, and c. We also 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 1s.*

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. By assumption, $Q^g(\hat{S})$ contains a line of ones. Let v be the index of this line. Parameter v is compatible with all jobs of S^m. Therefore the assignment of jobs in $\hat{S}$ is feasible with respect to parameter g, and thus g is simple.

Now assume that there exists a set of pairwise compatibles jobs $\hat{S}$ for which $Q^g(\hat{S})$ does not have a line of ones. In this case, if only pairwise conflicts are considered, all jobs of $\hat{S}$ can be assigned to the same machine. By assumption, there is no value for parameter g that is compatible with all elements of $\hat{S}$. Therefore, the pairwise compatibility is not sufficient to ensure the satisfaction of the configuration constraint related to parameter g, which is not simple. □

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

Definition 3 (see e.g. [5])**.** *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 1s.*

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 $\hat{S}$ of pairwise compatible jobs, and let i and j be respectively the smallest and the 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 $\hat{S}$, and thus all elements of line v of $Q^g(\hat{S})$ 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 [27], the author shows that a matrix has the consecutive ones property if and only if it does not contain the following submatrices (Figure 3).

$M_I(k), k \geq 3$

$$\begin{array}{c} 0 \\ 1 \\ 2 \\ \vdots \\ k-2 \\ k-1 \end{array} \begin{pmatrix} 1 & 1 & & & & \\ & 1 & 1 & & & \\ & & 1 & 1 & & \\ & & & \ddots & \ddots & \\ & & & & 1 & 1 \\ 1 & 0 & \cdots & 0 & 0 & 1 \end{pmatrix}$$

$M_{II}(k), k \geq 4$

$$\begin{array}{c} 0 \\ 1 \\ 2 \\ \vdots \\ k-2 \\ k-1 \end{array} \begin{pmatrix} 0 & 1 & 1 & \cdots & 1 & 1 \\ 1 & 1 & & & & \\ & 1 & 1 & & & \\ & & \ddots & \ddots & & \\ & & & 1 & 1 & 0 \\ 1 & \cdots & 1 & 1 & 0 & 1 \end{pmatrix}$$

$M_{III}(k), k \geq 3$

$$\begin{array}{c} 0 \\ 1 \\ 2 \\ \vdots \\ k-2 \\ k-1 \end{array} \begin{pmatrix} 1 & 1 & & & & & \\ & 1 & 1 & & & & \\ & & 1 & 1 & & & \\ & & & \ddots & \ddots & & \\ & & & & 1 & 1 & 0 \\ 0 & 1 & \ldots & 1 & 1 & 0 & 1 \end{pmatrix}$$

M_{IV}

$$\begin{array}{c} 0 \\ 1 \\ 2 \\ 3 \end{array} \begin{pmatrix} 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 \end{pmatrix}$$

M_V

$$\begin{array}{c} 0 \\ 1 \\ 2 \\ 3 \end{array} \begin{pmatrix} 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 & 1 \end{pmatrix}$$

Figure 3: Five forbidden structures for the consecutive ones property [27].

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 and the last columns do not have any common value, so they cannot be obtained by selecting pairwise compatible jobs either. 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 and 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.

2.5. 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, while constraints (2c) ensure that there cannot be two first jobs on any machine. Constraints (2d) ensure that if a job is on a machine m, either it has a predecessor, or it 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, and constraints (2g) indicate that u_i has to be 1 if i is late. Constraints (2h) say that for each parameter and each machine, exactly one value has to be selected. Constraints (2i) forbid to assign job i to machine m if the configuration of the latter is not compatible with i. Constraints (2j) ensure that a job cannot be followed by a forbidden successor.

When the 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 use to propose valid inequalities for the models above.

Observation 3. *For two jobs $i, j \in \mathcal{J}$, if $d_j < d_i$ and i and j are such that $A_{ik} = A_{jk}$ and $A_{ki} = A_{kj}$ for all $k \neq i, j$, the following constraints are valid.*

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

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

The following observation states that if no job is assigned to a machine, then the machine does not need to be configurated.

Observation 4. *It is possible to replace constraint* (2h) *by the following equality.*

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

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 (taking into account the availability date of m, the computing time of j, and the lag between j and i).

Proposition 4. *The following constraint is valid.*

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

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). Each variable involved in the right-hand side corresponds with a choice that make job i late. □

3. A path formulation

In this section, we propose a reformulation of the problem, where sequences of jobs are modelled by paths in a graph. An originality of our reformulation is that configurations are not considered explicitly,

even if they are not simple. They are deduced *a posteriori* by the sequence of jobs chosen on each machine. At the beginning of the time horizon, all parameters are possible, and each time a job is selected, it forbids a set of possible values for each parameter. If no more values are allowed for a parameter, then 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 have been forbidden by previous jobs, which would lead to a graph of exponential size. Therefore we use a path-flow formulation, and we solve it using a column-and-cut generation algorithm. In this section, we show how a state-of-the-art labelling algorithm can be extended to account for the specific configuration constraints in the pricing procedure.

To describe our approach, 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 labelling algorithm.

3.1. 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 of the network. 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 in 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 $j(n_j) \in \mathcal{J}^+(j(n_i))$. 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 for such a network, for 3 machines and 4 jobs.

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 an assignment of a sequence of jobs to a machine. 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) there exists a configuration for the machine that is compatible with all selected jobs. We use resources to formally verify the feasibility of paths.

To compute the accumulated time along the path, we use a so-called *disposable resource* [20]. The time resource consumption q_a of arcs in $\mathcal{A}$ is equal to

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

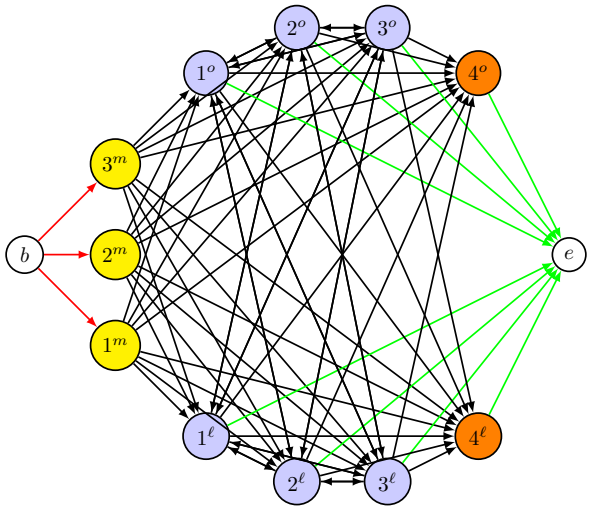

Figure 4: An example of network with three machines and four jobs . The source b and the sink e are colored in white, machine nodes, standard job nodes, and damaging job nodes are respectively colored in yellow, blue and orange. Nodes $1^o, \ldots, 4^o$ correspond with selecting these jobs on time, while 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 not possible 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.

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 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

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

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

Elementarity constraints are imposed by binary resources, one resource for every 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

$$l_{kj}^P = l_{k_1,j}^P + \begin{cases} 1, & a_k^P = (n', n) : \ j(n) = j, \\ 0, & \text{otherwise}, \end{cases} \quad k \in \{1, \ldots, k(P)\}, \ \text{ and } l_{0j}^P = 0. \tag{9}$$

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

Finally, we introduce so-called *selection resources* to impose the constraint that a machine configuration exists that is 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 $g \in H$ after visiting the k-th

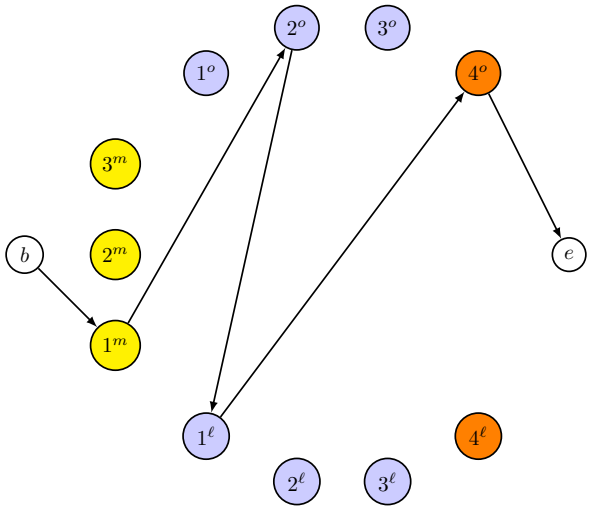

Figure 5: A path in the network of Figure 4, corresponding with job sequence $(2, 1, 4)$ on machine 1, where 2 and 4 are on-time and 1 is late. The path is obviously elementary. It is feasible if the sequence $(2, 1, 4)$ allows to schedule job 2 and jobs 4 on-time (accounting for the release date of 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.

node of path P:

$$C^P_{kg} = C^P_{k-1,g} \bigcap \begin{cases} \mathcal{V}_j(g), & j = j(n^P_k) \in J, \\ \mathcal{V}(g), & \text{otherwise,} \end{cases} \quad k \in \{1, \ldots, k(P)\}, \text{ and } C^P_{0g} = \mathcal{V}(g). \tag{10}$$

A path P satisfies the selection resources if $C^P_{0g} \neq \emptyset$ for all $k \in \{1, \ldots, k(P)\}$ and for all $g \in \mathcal{G}$.

An alternative way to define selection resources uses the fact that many machine parameters in our instances are simple (see definition 2). Selection resources are defined as before for non-simple machine parameters. For simple resources we instead define a conflict undirected 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. For each bipartite subgraph found, the corresponding conflicts are removed, and replaced by a single selection. 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 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, a cover of the conflict graph by complete bipartite subgraphs is computed using a simple greedy heuristic.

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

3.2. Master problem

For each path $P \in \mathcal{P}$, we define constant $\alpha^P_a \in \mathbb{Z}_+$, which is equal to the number of times arc $a \in \mathcal{A}$ is followed in path P. Our formulation is based on a unique set of variables. For each path $P \in \mathcal{P}$, binary variable λ_P is equal to one if and only if path h is used in the solution. With a slight abuse of notation,

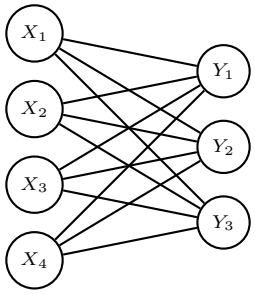

Figure 6: A complete bipartite subgraph of the conflict graph, where $\mathcal{X}^d = \{X_1, X_2, X_3, X_4\}$ and $\mathcal{Y}^d = \{Y_1, Y_2, Y_3\}$. In this case, one selection resource is sufficient to check that if a job of $\mathcal{X}^d$ is selected, no job from $\mathcal{Y}^d$ can be selected, and vice-versa.

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 at most 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 should be equal to 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 $\sum_{P \in \mathcal{P}} \sum_{a \in \mathcal{A}^\ell} \alpha_a^P \lambda_P \leq z^*$.

3.3. 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 with a negative reduced cost.

The pricing problem consists in finding in graph H a resource-feasible path of smallest reduced cost. For that we use a labelling algorithm, in which 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. In order 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

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

Even when the *bucket graph* variant of the labelling algorithm [23] is used to implement our method, the running time of the algorithm happens to be very large for the real-life instances we used. 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).

In order to improve the lower bound generated by the master problem, we use the dynamic ng-path technique as well as rank-one Chvátal-Gomory cuts, based on the set-partitioning constraints. In the ng-path relaxation, introduced by [3] for a vehicle routing problem, a memory consisting of neighbour clients is defined for every client, and the latter cannot be visited again 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 [22], memories are augmented dynamically by analysing 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. Afterwards, 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):

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

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 [18]. The cut separation problem is solved independently for each such vector of multipliers. Again, at most seven rounds of cut separation is performed, and the column generation

algorithm is run again between separation rounds. Each cut (12) introduces one additional resource into the pricing problem. Thus, a large number of active rank-one cuts may slow down considerably the pricing. In order to mitigate the impact of active rank-one cuts on the pricing problem difficulty, we use the limited-memory technique introduced in [17].

3.4. The diving heuristic

We now present a heuristic method based on the column-and-cut generation algorithm presented above. As it is not required anymore to calculate a valid lower bound, we heuristically reduce the graph H by considering only a subset of arcs in order to accelerate the solution of the pricing problem. 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 application of the dynamic ng-path relaxation is not necessary anymore.

To produce integer solutions for formulation (11), we use the diving heuristic with Limited Discrepancy Search (LDS) [24]. 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 we repeat this process until the solution produced by column generation is integer. The alternation of column generation algorithm and the fixing procedure corresponds to the simple diving heuristic or one dive in the branch-and-bound tree. To diversify the search, we apply ten dives in the branch-and-bound tree using LDS. The diving heuristic with LDS uses a tabu list of columns which are forbidden to be fixed to one. For details of this heuristic please refer to [24]. We use the same parameter set as in this reference.

4. Computational experiments

The main 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. We also estimate the gap between these results and the optimal values, using lower bounds produced by our MIP models.

4.1. Instance description and practical setting

For our computational experiments, we use industrial instances provided by Renault. For each instance, there is a unique facility that produces the prototypes (machines), and several facilities located in different other countries in which tests (jobs) are performed. It takes generally two days to move from one facility to another in the same country, and from 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 numbered from 3 to

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)

11, the only differences between i and im are the possible values for parameters, all other data are the same. We report in Tables 1 and 2 a summary of the instance sizes. For each instance of our test bed, we report the number of jobs (N), the number of machines (M), the number of parameters (G), and the total number of parameter values. The highest values are reported in bold.

As a reference, we use the results obtained by a CP model previously designed in the company (see Appendix A). The CP solver handles lexicographic optimization. It was run only once with the two ordered objectives and thus provides an upper bound for both objective function. We used IBM CP optimizer [2] to solve the model. The time needed by the solver to get stable objective values was two hours, and the solver was not able to deduce any lower bound.

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 value as a constraint in the second stage, where the minimum number of vehicles is sought. If the solver has not converged to an optimal solution for the first objective function after one hour of computing time, we stop the search and use the best solution value found by the solver to parameterize the second. The maximum computing time given for the second objective function is two hours minus the time spent for the first objective function. For the compact MIP formulation, when the second objective function is minimized, the solution minimizing the first objective function is given as MIP start to the solver. If some machines are not used in the input solution, then they are removed from the problem. IBM CPLEX version 22.1.1.0 [1] was used to solve this formulation. All computational experiments were run on an Intel(R) Core(TM) i7-9700K CPU @ 3.60GHz.

The path formulation was solved by BapCod [25], a generic branch-and-price library developed in our team. We use VRPSolver [20], a state-of-the-art BCP solver for vehicle routing and related problems, as a basis to implement our labelling algorithm. We needed to implement the selection resource type introduced above, which ensures that a suitable machine configuration can be chosen for the set of jobs assigned to a same machine. Cplex is used to solve the LP relaxation in the column generation process, and the process is stabilized using the in-out technique [19]. We recall in table 3 the different methods

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

tested.

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 completion of column-and-cut generation. In our experiments, only one bound was improved by one unit after one hour, therefore we do not report the results obtained by the exact branch-and-price method.

4.2. Numerical results

A summary of our experiments is reported in Table 4. We report the reference value obtained by CP optimizer in 7200 seconds (column CP ref / UB-2h), and in 600 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 approaches based on the compact MIP formulation, we report the best lower and upper bounds found in 7200 seconds (the CPU time, always equal to 7200 is omitted). For the approaches based on 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 slightly more than two hours to obtain a lower bound.

The improvements on the compact MIP formulation significantly increase the lower bound obtained by the method. However this does only reflect marginally on the quality of the primal solutions. Overall, applying them improves one solution (from 62 to 43 for instance 3), produces an additional feasible solution (instance 4m), and worsen the result for one instance (from 20 to 21 for instance 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 clearly outperforms the compact formulation for producing dual and primal bounds. For dual bound, there are some slight improvements for the first objective function (for a smaller 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 is able to find bounds in two hours for all instances except instance 11, which takes 2h and 24 minutes. When, it comes to computing primal bounds, the difference is even more important. The path formulation is able to 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, it is able to find solutions that are better than those obtained by the reference CP solver after two hours of computing time (although the latter method already produces optimal solutions for 13 instances of 20). For the second objective, the path formulation 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 clearly outperforms this method, which is not always able to 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 used can exclude optimal solutions in some cases.

Using the bipartite clique cover does not have a significant impact on the results. The same bounds are found, and the computing time is only marginally reduced. The largest improvement occurs for instance 5, where the total time goes from 1087 seconds to 687 seconds.

Finally, we draw some conclusions on the problem itself. First it transpires that the second objective function (min machines) leads to a clearly harder problem than minimizing the number of late jobs. This is true for all methods. For the exact column generation method, the gap is larger, 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 really make the problem harder for the column-generation method (although it seems to have a negative impact on 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.

	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. Symbol "-" means that the model was not able to compute any value. All times are reported in seconds (rounded up). The computing time for CP and both versions of the compact model are always equal to 7200 seconds. Bold faces indicate optimal primal solutions.

5. Conclusion

In this paper, we have introduced a new scheduling problem that arises in the car industry. An original feature of the problem is the fact that machines must be configured to accept the different jobs, and a machine has a unique configuration, which cannot be changed over time. 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 labelling algorithm to exclude solutions containing subsets of jobs for which there is no feasible machine configuration. Computational experiments show that the path-flow formulation is able to close the gap for all instances when the number of late jobs has to be minimized. When the number of machines is optimized, our method is able to find feasible solutions whose values are better than those obtained by a CP model used by the company. The exact resolution of the problem through branch-and-cut-and-price for the industrial instances is out of reach for our methodologies right now, but our path-flow model is also able to produce dual bounds for all test cases. Although the gap is generally not large, the time needed for each pricing phase requires a different approach to solve optimally the largest instances.

References

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

[2] *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`.

[3] Roberto Baldacci, Aristide Mingozzi, and Roberto Roberti, *New route relaxation and pricing strategies for the vehicle routing problem*, Operations Research **59** (2011), no. 5, 1269–1283.

[4] Hans L. Bodlaender, Klaus Jansen, and Gerhard J. Woeginger, *Scheduling with incompatible jobs*, Discrete Applied Mathematics **55** (1994), no. 3, 219–232.

[5] S. Booth and 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.

[6] Peter Brucker, *Scheduling algorithms*, Springer Berlin, Heidelberg, 2007.

[7] Grissele Centeno and Robert L. Armacost, *Minimizing makespan on parallel machines with release time and machine eligibility restrictions*, International Journal of Production Research **42** (2004), no. 6, 1243–1256.

[8] Jeng-Fung Chen and Tai-Hsi Wu, *Total tardiness minimization on unrelated parallel machine scheduling with auxiliary equipment constraints*, Omega **34** (2006), no. 1, 81–89.

[9] Rosiane de Freitas Rodrigues, Mitre Costa Dourado, and Jayme Luiz Szwarcfiter, *Scheduling problem with multi-purpose parallel machines*, Discrete Applied Mathematics **164** (2014), 313–319, Combinatorial Optimization.

[10] Vinícius L de Lima, Cláudio Alves, François Clautiaux, Manuel Iori, and José M Valério de Carvalho, *Arc flow formulations based on dynamic programming: Theoretical foundations and applications*, European Journal of Operational Research **296** (2022), no. 1, 3–21.

[11] Mengying Fu, Moeed Haghnevis, Ronald Askin, John Fowler, and Muhong Zhang, *Machine qualification management for a semiconductor back-end facility*, Proceedings of the 2010 Winter Simulation Conference, 2010, pp. 2486–2492.

[12] Jacques Lamothe, Francois Marmier, Matthieu Dupuy, Paul Gaborit, and Lionel Dupont, *Scheduling rules to minimize total tardiness in a parallel machine problem with setup and calendar constraints*, Computers & Operations Research **39** (2012), no. 6, 1236–1244, Special Issue on Scheduling in Manufacturing Systems.

[13] Joseph Y.-T. Leung and Chung-Lun Li, *Scheduling with processing set restrictions: A survey*, International Journal of Production Economics **116** (2008), no. 2, 251–262.

[14] ———, *Scheduling with processing set restrictions: A literature update*, International Journal of Production Economics **175** (2016), 1–11.

[15] Lars Mönch and Claude Yugma, *Scheduling jobs on parallel machines with qualification constraints*, 2015 IEEE International Conference on Automation Science and Engineering (CASE), 2015, pp. 657–658.

[16] Margaux Nattaf, Stéphane Dauzère-Pérès, Claude Yugma, and Cheng-Hung Wu, *Parallel machine scheduling with time constraints on machine qualifications*, Computers & Operations Research **107** (2019), 61–76.

[17] Diego Pecin, Artur Pessoa, Marcus Poggi, and Eduardo Uchoa, *Improved branch-cut-and-price for capacitated vehicle routing*, Mathematical Programming Computation **9** (2017), no. 1, 61–100.

[18] Diego Pecin, Artur Pessoa, Marcus Poggi, Eduardo Uchoa, and Haroldo Santos, *Limited memory rank-1 cuts for vehicle routing problems*, Operations Research Letters **45** (2017), no. 3, 206 – 209.

[19] Artur Pessoa, Ruslan Sadykov, Eduardo Uchoa, and Francois Vanderbeck, *In-out separation and column generation stabilization by dual price smoothing*, Experimental Algorithms (Berlin, Heidelberg) (Vincenzo Bonifaci, Camil Demetrescu, and Alberto Marchetti-Spaccamela, eds.), Springer Berlin Heidelberg, 2013, pp. 354–365.

[20] Artur Pessoa, Ruslan Sadykov, Eduardo Uchoa, and François Vanderbeck, *A generic exact solver for vehicle routing and related problems*, Mathematical Programming **183** (2020), 483–523.

[21] Michael L. Pinedo, *Scheduling*, Springer New York, NY, 2012.

[22] Roberto Roberti and Aristide Mingozzi, *Dynamic ng-path relaxation for the delivery man problem*, Transportation Science **48** (2014), no. 3, 413–424.

[23] Ruslan Sadykov, Eduardo Uchoa, and Artur Pessoa, *A bucket graph–based labeling algorithm with application to vehicle routing*, Transportation Science **55** (2021), no. 1, 4–28.

[24] Ruslan Sadykov, François Vanderbeck, Artur Pessoa, Issam Tahiri, and Eduardo Uchoa, *Primal heuristics for branch-and-price: the assets of diving methods*, INFORMS Journal on Computing **31** (2019), no. 2, 251–267.

[25] Ruslan Sadykov and François Vanderbeck, *BaPCod - a generic branch-and-price code*, Technical report, Inria Bordeaux Sud-Ouest, November 2021.

[26] Oscar Tellez, Samuel Vercraene, Fabien Lehuédé, Olivier Péton, and Thibaud Monteiro, *The fleet size and mix dial-a-ride problem with reconfigurable vehicle capacity*, Transportation Research Part C: Emerging Technologies **91** (2018), 99–123.

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

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 are considered, and some additional constraints such as

time windows on some jobs. Our simplifications do not change the main structure of the problem, and simplify the exposition. The solver used was CP Optimizer (CPO), and OPL (Optimization Programming Language) was used as a modelling language. We report this model using the vocabulary of CP. We use *optional job variables* to model the fact 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 size of the interval 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 disactivated. Binary variable α_{mgv} has the same meaning as the one used in our linear integer program. We use two sets of expressions that are 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 to assign 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 on 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 value of each parameter to each machine.