Discrete Applied Mathematics 355 (2024) 200–222

Contents lists available at ScienceDirect

Discrete Applied Mathematics

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

Graph coloring approaches for a production planning problem with makespan and setup penalties in a product-wheel context

Jocelin Cailloux [a], Nicolas Zufferey [a,*], Olivier Gallay [b]

[a] Geneva School of Economics and Management, GSEM - University of Geneva, Uni-Mail, 1211 Geneva 4, Switzerland
[b] Department of Operations, Faculty of Business and Economics (HEC Lausanne), University of Lausanne, Quartier UNIL-Chamberonne, 1015 Lausanne, Switzerland

ARTICLE INFO

Article history:
Received 15 November 2022
Received in revised form 18 April 2024
Accepted 26 April 2024
Available online xxxx

Keywords:
Combinatorial optimization
Metaheuristics
Graph coloring for production planning
Makespan and setup penalties
Traveling salesman problem
Scheduling

ABSTRACT

In this paper, we introduce a clustering and scheduling problem on a production line modeled as a single machine. A set of jobs (some of them being urgent) must be partitioned into clusters, and a robust (with respect to a min–max criterion) cyclic sequencing of the clusters must be determined (i.e., the product-wheel paradigm is employed). Each cluster has to satisfy two constraints: the setup constraint (i.e., only jobs with small setup times between them are allowed in the cluster) and the capacity constraint (i.e., the setup and processing times in the cluster cannot exceed a given shift duration). Three objective functions are minimized in a lexicographic fashion: (1) the number of urgent clusters (i.e., containing at least one urgent job); (2) the total number of clusters; (3) a worst-case scenario with respect to the setup among clusters. In other words, makespan and setup penalties are considered. Graph-coloring models and methods are designed for (1) and (2), whereas traveling-salesman approaches are introduced for (3). The considered problem was proposed by a micro-machining company located in Switzerland, named *DIXI polytool*. In order to cover their industrial needs, the company imposed very strict computing-time limitations (a few minutes only), and was able to provide realistic instances with different characteristics. Three methods are compared in our experiments: an integer linear model (with CPLEX), a constructive heuristic that represents a current-practice rule, and a metaheuristic relying on various tabu-search procedures. Results show the efficiency (with respect to quality and speed) of our metaheuristic, and managerial insights are provided.

1. Introduction

In this paper, we consider a partitioning and scheduling problem arising in a production planning context encountered by the micro-machining company *DIXI polytool* (www.dixipolytool.ch) located in Switzerland (*DIXI* for short). *DIXI* produces saws and drill bits, and their core business stands in the cutting of the point and the blade. The company offers thousands of different products, and a job corresponds to a fabrication order of a single product (along with the number of units to produce and its associated processing time). We propose to generate a partition of the jobs into clusters, which then have to be scheduled. The first priority of *DIXI* is to reduce the risk of late deliveries in order to avoid shortages

* Corresponding author.
E-mail addresses: jocelin.cailloux@unige.ch (J. Cailloux), n.zufferey@unige.ch (N. Zufferey), olivier.gallay@unil.ch (O. Gallay).

https://doi.org/10.1016/j.dam.2024.04.015

(lost sales) at the client level. The second priority is to reduce the setup times in their plant, as they have a direct impact on production capacity and costs. A job with a higher risk of shortage (at the client level) is called urgent and has to be considered with special attention. The urgency of a job is determined by *DIXI*.

A cluster is a set of jobs that will be produced sequentially in the production line (we do not have to schedule the jobs within a cluster). Two constraints are associated with each cluster. First, in the setup constraint, only jobs with the smallest possible setup times (5 min in this work) between them are allowed to belong to the same cluster. Second, the capacity constraint imposes that the total work duration in a cluster (i.e., processing times + setup times) cannot exceed a given shift duration (8 h in this work). A cluster is said to be urgent if it contains at least one urgent job. In order to reduce the risk of late deliveries, we first minimize the number of urgent clusters, and then the total number of clusters (which is strongly correlated with the minimization of the overall makespan). In this regard, we will show that generating an optimal job partition has strong connections with the graph-coloring problem (GCP) [38]. Therefore, we naturally consider a graph-coloring approach to model the minimization of the number of clusters.

The setup time between two clusters is estimated by *DIXI* as the average setup time between their jobs. In an ideal situation and in line with the product-wheel paradigm [51], once the jobs are dispatched into clusters (i.e., when the job partition is built), *DIXI* wants to repeat the production of the same sequence of clusters while maximizing the robustness of the cluster sequence (as a cluster sequence can be perturbed, for instance, by the introduction of an unexpected cluster containing either specific products or suddenly new urgent jobs). In this regard, it was decided to generate a job partition that minimizes the longest Hamiltonian cycle over the clusters. This corresponds to minimizing the setup times (among clusters) in the worst-case scenario for the production planner at *DIXI*. We will show that the traveling salesman problem (TSP) can model this feature of the problem [35]. Therefore, we naturally chose to evaluate a cluster sequence using a TSP approach.

The considered problem (P) has thus the originality to unify, for an industrial application, two of the most studied problems in the combinatorial-optimization literature, namely the GCP (for building clusters of jobs) and the TSP (for scheduling clusters). These two problems are NP-hard [3,38], and metaheuristics are thus appropriate methods for tackling large-sized instances. We consider the minimization of the three following objectives, in a lexicographic order (i.e., no lower objective can be improved if a degradation is encountered for a higher objective), which directly contributes to the minimization of the overall makespan: (f_1) the number of urgent clusters; (f_2) the total number of clusters; (f_3) the longest Hamiltonian cycle considering setup times among clusters.

The main contributions of this paper are the following. First, we propose an original partitioning and scheduling problem (P) applied to an industrial production planning context. This problem helps to directly reduce the shortages (at the client level) and the setup times (at the plant level) in real industrial situations. Moreover, this partitioning problem can be useful in any production planning problem where job scheduling cannot be accurately determined. Second, we design an integer linear program which captures all the features of (P). Third, GCP and TSP models and methods are designed and combined to generate efficient solutions. Fourth, we propose various sensitivity analyses and managerial insights that could help a decision maker when planning and scheduling the production of the involved plant.

The paper is organized as follows. In Section 2, we present an overview of the literature on related topics. We describe problem (P) in Section 3. A nonlinear program and its linearization are developed in Section 4. In Section 5 we propose GCP and TSP solution methods for generating efficient solutions for large instances of (P). The numerical results are discussed in Section 6, along with managerial insights. Finally, conclusions and possible future works are given in Section 7.

2. Literature review

In this section, we first position the considered problem (P) with respect to the production planning literature in a makespan minimization context, and we highlight the links between (P) and the GCP. Second, we discuss the setup minimization feature of (P), and its analogies with the TSP. Finally, we show the relevance of lexicographic optimization and finding robust solutions in industry.

Minimizing the makespan is one of the most common objectives in the scheduling literature [41]. To minimize the makespan, jobs can be clustered using the family scheduling model, which leads to reducing the setup times in the schedule [42]. Problem (P) presents visible similarities with some batching and scheduling problems which are efficiently tackled using various solution methods [27] (e.g., variable neighborhood search [39], simulated annealing [53], recovering beam search [10]). For modeling a partitioning problem, graph coloring formulations are commonly used [38] and are often tackled efficiently with local search metaheuristics (e.g., tabu search [9,28]). Graph coloring problems have been often used to model scheduling problems [19,23], and tabu search was shown to be efficient in solving such problems within a few minutes [47,48].

Setup is one of the most widely studied feature of lot sizing and scheduling problems [41]. (P) can be reduced to the single machine scheduling problem with sequence dependent setup times [5]. This scheduling problem has already been shown to be efficiently solved with tabu search [6], genetic algorithms [34], and ant colony optimization [36]. Because of the employed product-wheel paradigm [11,26,51] (i.e., the same sequence of clusters must be ideally produced in a cyclic way), (P) shares similarities with the TSP [5]. Moreover, the product-wheel paradigm is commonly considered in job scheduling problems from different industries, such as the automotive industry [33], the chemical industry [49], and in wafer manufacturing [52]. Such problems are efficiently tackled using various metaheuristics, such as tabu search [12], and evolutionary algorithms [18,29].

Industrial optimization problems have often to face different, conflicting objective functions. In this context, lexicographic optimization is an efficient and relevant approach. It consists in ranking the objectives once and for all (i.e., a lower-level objective function can only be improved without deteriorating a higher-level objective function). This approach has been employed in various fields [22], such as the automotive industry [30,46], the luxury goods industry [44], the electricity industry [1], the electronics industry [45], and in job-shop scheduling contexts [8]. Robust solutions [4] are also expected and favored in industry. Informally and generally speaking, a solution is robust if it is not perturbed drastically in case of uncertain events. In this context, minimizing the worst-case scenario corresponds to the min–max criterion, which is a common way to determine robust solutions [2]. In the TSP context, it is applied in [14] where an exact method is compared with a heuristic based on the combination of various relaxations of the problem. Min–max criteria for generating robust solutions are also employed in manufacturing industries [15,31], where a branch-and-bound method and a greedy heuristic are proposed.

Therefore, from the existing literature, it appears that (P) is a unique combination of the GCP, the TSP, and the job scheduling problem with sequence dependent setup times. For each of these problems, Mixed-Integer Linear Programming (MILP) models and local search algorithms have been successfully applied. It also appears that, in contrast with evolutionary algorithms (that are population-based methods), tabu search is particularly efficient for such problems when computing time is limited to a few minutes (which is the case for *DIXI*). This motivates us to propose the same kind of approaches for (P).

3. Presentation of problem (P)

In this section, we first present problem (P) such that it can be captured by practitioners. Next, we give a formal presentation of (P), along with its connections with the GCP and the TSP.

3.1. Informal description of (P)

Considering a single production line, we have to perform sequentially a set of n jobs with known processing times, and some of them are urgent (i.e., they have the priority over the other jobs). A setup time is encountered between each pair of jobs performed consecutively. Each setup time is a multiple of 5 min belonging to the interval [5, 120] min.

A solution to the problem is a partition of the jobs into clusters (or groups), where two jobs can belong to the same cluster if the setup time between them is 5 min. The satisfaction of this setup constraint favors the overall setup-cost reduction. Moreover, the working time (i.e., processing time plus setup time) contained in a cluster cannot exceed 8 h (this is called the capacity constraint), which corresponds to a shift duration. *DIXI* will then simply process the jobs cluster by cluster. A setup time is also encountered between each pair of clusters performed sequentially, and its value is computed as the average setup time occurring between the jobs of the two involved clusters.

DIXI proposes to optimize three objectives (f_1, f_2, f_3) in a lexicographic fashion. (f_1) Distribute the urgent jobs into a minimum number k^* of urgent clusters (a cluster is urgent if it contains at least one urgent job). (f_2) Distribute the other jobs into a minimum number k of clusters. The k^* urgent clusters are also taken into account here (i.e., a non-urgent job can also be integrated into an urgent cluster as long as the setup and capacity constraints are satisfied). Moreover, it is possible to reallocate the urgent jobs to other urgent clusters (i.e., k^* is not increased). In other words, after planning the urgent jobs in the previous step, we aim here at minimizing the number of additional clusters created for inserting the non-urgent jobs into the production schedule. (f_3) Reassign the jobs into the k existing clusters (without augmenting k^*) in order to minimize the worst-case scenario for the total setup time among clusters, assuming there is also a setup time between the last cluster and the first cluster of the cluster sequence (this is a standard assumption in a product-wheel production context, where the production of the same cluster sequence is repeated again and again). Minimizing the worst-case scenario is useful at it allows the production planner to better face unexpected events (see below for more explanations). The joint consideration of these three objectives contributes to reduce the overall production time, and it favors on-time deliveries to clients.

The following main simplifications are made with respect to the real problem faced by *DIXI*. First, we consider deterministic durations (e.g., processing times, setup times). Second, sudden and unexpected demands (i.e., jobs) are ignored. For instance, an important client might place an order on short notice, which means that the associated jobs must be incorporated in the already planned solution. In other words, uncertainty is not taken into account. However, these simplifications do not have major impacts on the proposed methods for the studied problem (P). Indeed, the following measures can be taken, and some compensation phenomena can also occur.

- A longer-than-expected job (or setup time) can be simply compensated by a shorter-than-expected one. Moreover, the planned clusters are not perfectly filled with jobs and setup times (remind that a cluster corresponds to a shift of 8 h). For each cluster, the remaining buffer time (i.e., the time interval during which no setup or job processing operations take place) makes it possible to cope with some longer-than-expected jobs and setup times.
- The sudden and unexpected additional jobs can be scheduled either during the buffer times, or by building new clusters that are then scheduled somewhere in the planned solution (i.e., between two already-planned but non-already-performed clusters). In this context, a simulation–optimization algorithm could also be easily derived from the below-proposed methods (e.g., anytime additional jobs have to be scheduled, or when the processing of some

clusters is completed, the proposed algorithms are employed again for updating the solution), relying on a rolling-window simulation procedure. In this context the window size is typically one week, and the planning horizon one month.

- Minimizing the largest total setup time among clusters (i.e., f_3) contributes to reduce the stress of the production planner. Indeed, the proposed algorithms provide the decision maker with the job clusters and the value of f_3. But then, s/he can determine the cluster sequence according to her/his priorities at that time (which could also include non-modeled features). The good news is that the selected cluster sequence is likely to be better when compared to the provided total setup time among clusters (i.e., the provided value of f_3). Indeed, the decision maker is likely to not select the worst-case scenario. This also increases her/his flexibility for incorporating new clusters into the on-hand solution.

Although relying on more formalism, the next section also discusses other aspects that are important from a practical standpoint.

3.2. Formal description of (P) and connections with the GCP and the TSP

Let J be the set of n jobs to be performed, and $J^* \subset J$ be the set of urgent jobs. The sets J and J^* are given as input. For each job j, we know its processing time p_j. For each pair (j, j') of jobs, we know the setup time $s_{jj'} = s_{j'j}$ between them if they are processed consecutively on the production line (which can be modeled as a single machine in this work). We assume that the setup times are in the range $\left[s^{min}, s^{max}\right]$ (with steps of 5 min), where s^{min} and s^{max} are equal to 5 min and 120 min in this study, respectively.

A solution C is a partition of the jobs of J into clusters $C_1, C_2, C_3, \ldots$ (i.e., each C_i contains a subset of jobs). Two jobs j and j' can be in the same cluster C_i if $s_{jj'} = s^{min}$. This constraint (called here the setup constraint) is imposed by *DIXI*. The satisfaction of this constraint favors the overall setup-cost reduction, which is a crucial issue for *DIXI* as their hourly setup cost (with respect to the involved workers) is significant. *DIXI* will then simply process the jobs cluster by cluster. Moreover, the working time (i.e., processing time plus setup time) contained in a cluster cannot exceed the shift duration $\overline{p}$ (which is equal to 8 h in this study). Formally, for each cluster C_i, $\sum_{j \in C_i} p_j + (|C_i| - 1)s^{min} \leq \overline{p}$ (this is called here the capacity constraint). Between each pair $(C_i, C_{i'})$ of clusters, $S_{ii'}$ corresponds to the average setup time occurring between the jobs of the two clusters. More precisely, $S_{ii'} = \frac{\sum_{j \in C_i} \sum_{j' \in C_{i'}} s_{jj'}}{|C_i| \times |C_{i'}|}$.

Given a graph $G = (V, E)$, with vertex set V and edge set E, the k-coloring problem consists in assigning an integer (called color) in $\{1, \ldots, k\}$ to every vertex such that two adjacent vertices have different colors. The GCP consists in finding a k-coloring with the smallest possible k. From the input data of problem (P), we can build a graph $G = (V, E)$ as follows. We associate a vertex j with each job j, a color i with each cluster C_i, and an edge $\left[j, j'\right]$ with each pair (j, j') of incompatible jobs. Two jobs j and j' are incompatible if $s_{jj'} > s^{min}$ or $p_j + p_{j'} + s^{min} > \overline{p}$ (otherwise, they are compatible). Coloring G with k colors is equivalent to assigning a cluster C_i (with $i \in \{1, \ldots, k\}$) to each job. The capacity constraint (as defined above) of the clusters is considered in (P) but not in the GCP.

The TSP is defined on a complete undirected graph $G = (V, E)$, with vertex set V and edge set E. A symmetric cost matrix M is defined on E. The TSP consists in determining a cycle of minimum length, visiting each vertex once. Such a cycle is called a Hamiltonian cycle. With respect to problem (P), we can build a graph $G = (V, E)$ as follows. We associate a vertex i with each cluster C_i, and an edge $\left[i, i'\right]$ with each pair $(C_i, C_{i'})$ of clusters. For each edge $\left[i, i'\right]$, its cost is defined as $M_{ii'} = S_{ii'}$. Finding the smallest Hamiltonian cycle in G is equivalent to finding a cyclic sequence of clusters minimizing the overall setup among them. (P) is however an extension of the TSP. Indeed, in (P), we have to build clusters to minimize the longest Hamiltonian cycle. Consequently, in order to employ state-of-the-art TSP solution methods for (P), it is possible to adapt the matrix $(S_{ii'})$ of setup times by building a new matrix S' such that $S'_{ii'} = S^{max} - S_{ii'}$, where $S^{max} = \max_{i,i'} S_{ii'}$. Indeed, without such an adaptation, any TSP algorithm will find the shortest (and not the longest) Hamiltonian cycle.

We define three objectives, which are optimized in a lexicographic fashion. Let k be the number of employed clusters, and $k^* \leq k$ the number of clusters containing at least one urgent job. First, a clustering (i.e., a partition) of the urgent jobs is determined such that k^* is minimized (this is denoted as problem (P1)). Second, a clustering of all the jobs is determined such that k is minimized without increasing k^* (problem (P2)). Third, we modify the clustering such that the longest Hamiltonian cycle considering setup times among clusters is minimized, without increasing k and k^* (problem (P3)). Other methods to handle multi-objective optimization problems exist (e.g., utility or goal programming, simultaneous or interactive approaches) [37]. Even if no perfect approach comes out and each one has its own advantages and drawbacks, *DIXI* imposed the considered lexicographic approach because their natural priorities are in this order: (1) deliver on time to the client level (minimizing the number of urgent clusters obviously contributes to this priority); (2) use the available production resources (i.e., workers and machines) as much as possible (minimizing the total number of clusters directly contributes to this goal, as it favors having fully loaded clusters); (3) reduce the risk of augmenting the overall makespan because of the setup times among clusters (minimizing the worst-case scenario for the total setup time among clusters contributes to this objective). Moreover, the consideration of these three objectives also contributes to the minimization of the overall makespan, which is a very important point, given the significant hourly resource cost encountered by *DIXI*. Such client-first-company-second (and sometimes setup-third) way of prioritizing objectives is also encountered in other industry-based studies [32,43,50].

4. Mathematical model

In this section, we provide first a nonlinear integer programming formulation for (P). Next, a procedure to linearize the previous model is given. Three preprocessing steps and model improvements are finally proposed. Note that indexes j and j' refer to jobs, whereas indexes i and i' refer to clusters (i.e., C_i and $C_{i'}$). We define the following sets and parameters.

Sets:

- J: set of jobs
- $J^* \subset J$: set of urgent jobs
- $\mathcal{C}$: set of possible cluster indexes

Parameters:

- $n \in \mathbb{N}$: number of jobs
- $\bar{c} \in \mathbb{N}$: time capacity of each cluster, in minutes (here, a shift duration of 480 min = 8 h)
- $b_{jj'} = 1$ if jobs j and j' can be in the same cluster (i.e., they are compatible); $b_{jj'} = 0$ otherwise
- $s_{jj'} \in \mathbb{N}$: setup time occurring between jobs j and j', in minutes
- $p_j \in \mathbb{N}$: processing time of job j, in minutes
- $k \in \mathbb{N}$: number of clusters
- $s^{min} \in \mathbb{N}$: minimum possible setup time between jobs
- $s^{max} \in \mathbb{N}$: maximum possible setup time between jobs
- $\bar{q} \in \mathbb{N}$: upper bound for the number of jobs that a cluster can contain

4.1. Nonlinear model

The decision variables are the following.

- $A_{ij} = 1$ if job j is in cluster C_i; $A_{ij} = 0$ otherwise
- $U_i = 1$ if cluster C_i contains at least one urgent job; $U_i = 0$ otherwise
- $E_i = 1$ if cluster C_i is not empty; $E_i = 0$ otherwise
- $P_{ii'} = 1$ if cluster C_i is the direct predecessor of cluster $C_{i'}$ (in the cluster sequence, i.e., in the Hamiltonian cycle); $P_{ii'} = 0$ otherwise
- $L_i \in \mathbb{R}$: setup time occurring after cluster C_i with respect to the considered cluster sequence
- $S_{ii'} \in \mathbb{R}$: setup time between clusters C_i and $C_{i'}$
- $W_i \in \mathbb{N}$: a unique number in $\{1, \ldots, n\}$ assigned to each cluster C_i in order to avoid subtours
- $T_{ii'} \in \mathbb{N}$: sum of the job-related setup times between jobs of cluster C_i and jobs of cluster $C_{i'}$
- $N_i \in \mathbb{N}$: number of jobs in cluster C_i
- $D_{ii'} \in \mathbb{N}$: number of possible distinct pairs of jobs between clusters C_i and $C_{i'}$
- $Q_{ii'} \in \mathbb{R}$: variables that help to compute the average setup time between jobs of clusters C_i and $C_{i'}$ such that $Q_{ii'} = \frac{1}{D_{ii'}}$

The set of constraints can be divided into three categories. The first constraint set is associated with a GCP formulation (for the clustering part), whereas the second constraint set is connected with a TSP formulation (for computing the value of the clustering according to setup times). The third constraint set is used to compute the average setup time between the jobs of two different clusters. The proposed model is an integer linear program, except for the computation of the average setup time between two clusters.

The first two objective functions (1) and (2) correspond to minimizing the number of urgent clusters and the total number of clusters, respectively. Objective function (3) minimizes the maximum Hamiltonian cycle length with respect to the setup times occurring among clusters. In line with row-generation techniques [21,25], to tackle the non-linear objective (3) with CPLEX, a series of max problems is solved iteratively. The following two steps (a) and (b) are repeated until no feasible solution can be obtained anymore or the computation time limit is reached: (a) CPLEX is employed to solve the on-hand max problem; (b) a constraint is added to the on-hand problem to forbid CPLEX to return again the same solution. At the end of the process, either an optimal solution to the min–max problem is found (because no feasible solution can be found anymore in step (a)), or a message of non-optimality is returned (because the computation time limit is reached). Typically, at each iteration i of this process, an optimal value for the on-hand max problem is returned, with a smaller (or equal) value than the one found at iteration $(i-1)$ (as the max problem tackled at iteration i is more constrained).

$$\min \quad \sum_{i \in \mathcal{C}} U_i \tag{1}$$

$$\min \quad \sum_{i \in \mathcal{C}} E_i \tag{2}$$

$$\min \max \quad \sum_{i \in \mathcal{C}} L_i \tag{3}$$

s.t.

$$\text{(GCP)} \begin{cases} \sum_{i \in \mathcal{C}} A_{ij} = 1 & \forall j \in J \quad \text{(a)} \\ \sum_{j \in J} A_{ij}(p_j + s^{min}) \leq \bar{c} + s^{min} & \forall i \in \mathcal{C} \quad \text{(b)} \\ U_i \geq A_{ij} & \forall i \in \mathcal{C}, \quad \forall j \in J^* \quad \text{(c)} \\ E_i \geq A_{ij} & \forall i \in \mathcal{C}, \quad \forall j \in J \quad \text{(d)} \\ A_{ij} + A_{ij'} \leq 1 + b_{jj'} & \forall i \in \mathcal{C}, \quad \forall j, j' \in [\![1, n]\!] \mid j < j' \quad \text{(e)} \\ N_i = \sum_{j \in J} A_{ij} & \forall i \in \mathcal{C} \quad \text{(f)} \end{cases} \tag{4}$$

$$\text{(TSP)} \begin{cases} L_i \geq S_{ii'} - s^{max}(1 - P_{ii'}) & \forall i, i' \in \mathcal{C} \mid i \neq i' \quad \text{(a)} \\ L_i \leq S_{ii'} + s^{max}(1 - P_{ii'}) & \forall i, i' \in \mathcal{C} \mid i \neq i' \quad \text{(b)} \\ \sum_{i \in \mathcal{C} \setminus \{i'\}} P_{ii'} = 1 & \forall i' \in \mathcal{C} \quad \text{(c)} \\ \sum_{i' \in \mathcal{C} \setminus \{i\}} P_{ii'} = 1 & \forall i \in \mathcal{C} \quad \text{(d)} \\ W_i - W_{i'} + kP_{ii'} \leq k - 1 & \forall i, i' \in \mathcal{C} \setminus \{1\} \mid i \neq i' \quad \text{(e)} \end{cases} \tag{5}$$

$$\text{(Setup times)} \begin{cases} T_{ii'} = \sum_{\substack{j,j' \in [\![1,n]\!] \\ j \neq j'}} s_{jj'} \left(A_{ij} \times A_{i'j'}\right) & \forall i, i' \in [\![1, k]\!] \mid i < i' \quad \text{(a)} \\ S_{ii'} = S_{i'i} & \forall i, i' \in [\![1, k]\!] \mid i < i' \quad \text{(b)} \\ D_{ii'} = N_i \times N_{i'} & \forall i, i' \in [\![1, k]\!] \mid i < i' \quad \text{(c)} \\ Q_{ii'} = \dfrac{1}{D_{ii'}} & \forall i, i' \in [\![1, k]\!] \mid i < i' \quad \text{(d)} \\ S_{ii'} = T_{ii'} \times Q_{ii'} & \forall i, i' \in [\![1, k]\!] \mid i < i' \quad \text{(e)} \end{cases} \tag{6}$$

$$\text{(Domain constraints)} \begin{cases} A_{ij} \in \{0, 1\} & \forall i \in \mathcal{C}, \quad \forall j \in J \quad \text{(a)} \\ U_i \in \{0, 1\} & \forall i \in \mathcal{C} \quad \text{(b)} \\ E_i \in \{0, 1\} & \forall i \in \mathcal{C} \quad \text{(c)} \\ P_{ii'} \in \{0, 1\} & \forall i, i' \in \mathcal{C} \quad \text{(d)} \\ L_i \geq 0 & \forall i \in \mathcal{C} \quad \text{(e)} \\ S_{ii'} \geq 0 & \forall i, i' \in \mathcal{C} \quad \text{(f)} \\ W_i \in \{1, \ldots, n\} & \forall i \in \mathcal{C} \quad \text{(g)} \\ 1 \leq W_i \leq k - 1 & \forall i \in \mathcal{C} \quad \text{(h)} \\ T_{ii'} \in \mathbb{N} & \forall i, i' \in \mathcal{C} \quad \text{(i)} \\ N_i \in \mathbb{N} & \forall i \in \mathcal{C} \quad \text{(j)} \\ D_{ii'} \in \mathbb{N} & \forall i, i' \in \mathcal{C} \quad \text{(k)} \\ 0 \leq Q_{ii'} \leq 1 \in \mathbb{R} & \forall i, i' \in \mathcal{C} \quad \text{(l)} \end{cases} \tag{7}$$

Constraints (4)(a)–(4)(f) are connected with the GCP, where constraints (4)(a) ensure that each job is assigned to exactly one cluster. The capacity constraint of each cluster is respected thanks to constraints (4)(b). A cluster C_i is set as urgent if it contains at least an urgent job thanks to constraints (4)(c) and is set in this case as non-empty thanks to constraints (4)(d). Last, constraints (4)(e) make sure that two jobs can be in the same cluster only if they are compatible. Constraints (4)(f) compute the number of jobs per cluster.

Constraints (5)(a)–(5)(e) relies on the proposed TSP formulation. Constraints (5)(a) and (5)(b) allow to linearize the costs. It is ensured that the clusters are sequenced one after the other thanks to constraints (5)(c) and (5)(d). Constraints (5)(e) are necessary to break the subtours.

Constraints (6)(a)–(6)(e) compute the average setup time between each pair of clusters. The sum of the setup times occurring between the jobs of a pair of clusters is computed by constraints (6)(a), where $A_{ij} \times A_{i'j'}$ allows to compute the pairs of jobs for which a job-related setup time occurs between a given pair of clusters (i.e., $A_{ij} \times A_{i'j'} = 1$ if jobs j and j' are

in clusters i and i', respectively, and $A_{ij} \times A_{i'j'} = 0$ otherwise). Constraints (6)(b) allow to compute only half of the setup times by enforcing the symmetry. Constraints (6)(c) and (6)(d) compute the number of setup times among jobs between each pair of clusters. This corresponds to the denominator written as a fraction. Finally, constraints (6)(e) compute the average setup time between each pair of clusters.

Constraints (7)(a)–(7)(l) are domain constraints.

4.2. Linearized model

The above formulations (1)–(5)(e), (6)(b) and (7)(a)–(7)(l) remain unchanged here. The linearization of constraints (6)(a) is determined by fixing the value of $V_{ii'jj'}$ accordingly to all possible values for A_{ij} and $A_{i'j'}$ (i.e., by setting $V_{ii'jj'} = A_{ij} \times A_{i'j'}$). Based on a mechanism presented in [7], we determine linearized constraints (6)(c)–(6)(e) as follows. For each quadratic constraint, one of the two involved variables is translated in its binary writing for linearizing the multiplication. For each variable $Q_{ii'}$, many binary variables are created as there are several possible values for $D_{ii'}$ to compute the fraction. To this end, we need the following new decision variables, where $q \in [\![1, \bar{q}^2]\!]$ corresponds to a possible value for variable $D_{ii'}$, and g indicates the position of the bit of each binary variable employed in the binary writing of an integer variable. For example, the binary writing in $\bar{g}$ bits of a variable $X \in \mathbb{N}$ can be represented with the set of variables $B_g \in \{0, 1\}, \forall g \in \{0, 1, \ldots, \bar{g}\}$, where B_g is 2^g if $B_g = 1$, and B_g is 0 otherwise. With such a binary representation of X, we can compute $X = \left(2^0 B_0\right) + \left(2^1 B_1\right) + \cdots + \left(2^{\bar{g}} B_{\bar{g}}\right)$. Using such a binary representation, the multiplication $Z = X \times Y$ can be written with the sum $Z = \left(2^0 B_0 \times Y\right) + \left(2^1 B_1 \times Y\right) + \ldots + \left(2^{\bar{g}} B_{\bar{g}} \times Y\right)$, where the multiplications involved in each term of the sum can be more easily linearized as it involves binary variables. Another detailed illustration of this type of linearization technique can be found in [20].

- $V_{ii'jj'} = 1$ if clusters C_i and $C_{i'}$ contain jobs j and j', respectively; $V_{ii'jj'} = 0$ otherwise
- $D'_{ii'q} = 1$ if $D_{ii'} \leq q$; $D'_{ii'q} = 0$ otherwise
- $D''_{ii'q} = 1$ if $D_{ii'} = q$; $D''_{ii'q} = 0$ otherwise
- $B^N_{ig} \in \{0, 1\}$: binary writing for variables N_i
- $B^T_{ii'g} \in \{0, 1\}$: binary writing for variables $T_{ii'}$
- $Z^D_{ii'g} \in \mathbb{N}$: variables to linearize $B^N_{ig} \times N_{i'}$
- $Z^S_{ii'g} \in \mathbb{R}$: variables to linearize $B^T_{ii'g} \times Q_{ii'}$

$$(T_{ii'}) \begin{cases} V_{ii'jj'} \geq A_{ij} + A_{i'j'} - 1 & \forall i, i' \in [\![1,k]\!] \mid i < i', \quad \forall j, j' \in [\![1,n]\!] \mid j \neq j' \quad \text{(a)} \\ V_{ii'jj'} \leq A_{ij} & \forall i, i' \in [\![1,k]\!] \mid i < i', \quad \forall j, j' \in [\![1,n]\!] \mid j \neq j' \quad \text{(b)} \\ V_{ii'jj'} \leq A_{i'j'} & \forall i, i' \in [\![1,k]\!] \mid i < i', \quad \forall j, j' \in [\![1,n]\!] \mid j \neq j' \quad \text{(c)} \\ T_{ii'} = \sum_{\substack{j,j' \in [\![1,n]\!] \\ j \neq j'}} s_{jj'} V_{ii'jj'} & \forall i, i' \in [\![1,k]\!] \mid i < i' \quad \text{(d)} \end{cases} \tag{8}$$

$$(Q_{ii'}) \begin{cases} D'_{ii'q} \geq \dfrac{1}{\bar{q}^2}(D_{ii'} + 1 - q) & \forall i, i' \in [\![1,k]\!] \mid i < i', \quad \forall q \in [\![1, \bar{q}^2]\!] \quad \text{(a)} \\ D'_{ii'q} \leq \dfrac{1}{q} D_{ii'} & \forall i, i' \in [\![1,k]\!] \mid i < i', \quad \forall q \in [\![1, \bar{q}^2]\!] \quad \text{(b)} \\ D''_{ii'q} = D'_{ii'q} - D'_{ii'q+1} & \forall i, i' \in [\![1,k]\!] \mid i < i', \quad \forall q \in [\![1, \bar{q}^2 - 1]\!] \quad \text{(c)} \\ D''_{ii'\bar{q}^2} = D'_{ii'\bar{q}^2} & \forall i, i' \in [\![1,k]\!] \mid i < i' \quad \text{(d)} \\ Q_{ii'} \geq \dfrac{1}{q} D''_{ii'q} & \forall i, i' \in [\![1,k]\!] \mid i < i', \quad \forall q \in [\![1, \bar{q}^2]\!] \quad \text{(e)} \\ Q_{ii'} \leq \dfrac{1}{q} + \left(1 - \frac{1}{q}\right) \times (1 - D''_{ii'q}) & \forall i, i' \in [\![1,k]\!] \mid i < i', \quad \forall q \in [\![1, \bar{q}^2]\!] \quad \text{(f)} \end{cases} \tag{9}$$

$$(D_{ii'}) \begin{cases} N_i = \sum_{g=0}^{\lfloor \log(\bar{q}) \rfloor} 2^g B^N_{ig} & \forall i \in C \quad \text{(a)} \\ Z^D_{ii'g} \leq \bar{q} B^N_{ig} & \forall i, i' \in [\![1,k]\!] \mid i < i', \quad g \in [\![0, \lfloor \log(\bar{q}) \rfloor]\!] \quad \text{(b)} \\ Z^D_{ii'g} \leq N_{i'} & \forall i, i' \in [\![1,k]\!] \mid i < i', \quad g \in [\![0, \lfloor \log(\bar{q}) \rfloor]\!] \quad \text{(c)} \\ Z^D_{ii'g} \geq N_{i'} - \bar{q}(1 - B^N_{ig}) & \forall i, i' \in [\![1,k]\!] \mid i < i', \quad g \in [\![0, \lfloor \log(\bar{q}) \rfloor]\!] \quad \text{(d)} \\ D_{ii'} = \sum_{g=0}^{\lfloor \log(\bar{q}) \rfloor} 2^g Z^D_{ii'g} & \forall i, i' \in [\![1,k]\!] \mid i < i' \quad \text{(e)} \end{cases} \tag{10}$$

$$(S_{ii'})\begin{cases} T_{ii'} = \sum_{g=0}^{\lfloor \log(s^{max}) \rfloor} 2^g B^T_{ii'g} & \forall i, i' \in \mathcal{C} \mid i \neq i' \quad \text{(a)} \\ Z^S_{ii'g} \leq s^{max} B^T_{ii'g} & \forall i, i' \in \mathcal{C} \mid i \neq i', \quad g \in [\![0, \lfloor \log(s^{max}) \rfloor]\!] \quad \text{(b)} \\ Z^S_{ii'g} \leq Q_{ii'} & \forall i, i' \in \mathcal{C} \mid i \neq i', \quad g \in [\![0, \lfloor \log(s^{max}) \rfloor]\!] \quad \text{(c)} \\ Z^S_{ii'g} \geq Q_{ii'} - s^{max}(1 - B^T_{ii'g}) & \forall i, i' \in \mathcal{C} \mid i \neq i', \quad g \in [\![0, \lfloor \log(s^{max}) \rfloor]\!] \quad \text{(d)} \\ S_{ii'} = \sum_{g=0}^{\lfloor \log(s^{max}) \rfloor} 2^g Z^S_{ii'g} & \forall i, i' \in \mathcal{C} \mid i \neq i' \quad \text{(e)} \end{cases} \tag{11}$$

$$\text{(Domain constraints)}\begin{cases} V_{ii'jj'} \in \{0,1\} & \forall i, i' \in \mathcal{C}, \quad \forall j, j' \in J \quad \text{(a)} \\ D'_{ii'q} \in \{0,1\} & \forall i, i' \in \mathcal{C}, \quad \forall q \in [\![1, \bar{q}^2]\!] \quad \text{(b)} \\ D''_{ii'q} \in \{0,1\} & \forall i, i' \in \mathcal{C}, \quad \forall q \in [\![1, \bar{q}^2]\!] \quad \text{(c)} \\ B^N_{ig} \in \{0,1\} & \forall i \in \mathcal{C}, \quad \forall g \in [\![1, \log(s^{max})]\!] \quad \text{(d)} \\ B^T_{ig} \in \{0,1\} & \forall i \in \mathcal{C}, \quad \forall g \in [\![1, \log(s^{max})]\!] \quad \text{(e)} \\ Z^D_{ii'g} \in \mathbb{N} & \forall i, i' \in \mathcal{C}, \quad \forall g \in [\![1, \log(s^{max})]\!] \quad \text{(f)} \\ Z^S_{ii'g} \in \mathbb{R} & \forall i, i' \in \mathcal{C}, \quad \forall g \in [\![1, \log(s^{max})]\!] \quad \text{(g)} \end{cases} \tag{12}$$

Constraints (8)(a)–(8)(d) allow to compute the setup time occurring between two clusters i and i' using variables $V_{ii'jj'}$. As all three variables are binary, we use constraints (8)(a) to ensure that $V_{ii'jj'} = 1$ if both $A_{ij} = 1$ and $A_{i'j'} = 1$. Otherwise (i.e., if A_{ij} or $A_{i'j'}$ is equal to zero) constraints (8)(b)–(8)(c) enforce $V_{ii'jj'} = 0$. Constraints (8)(d) use the variables $V_{ii'jj'}$ to compute the sum of the jobs related setup times occurring between each pair of clusters.

Constraints (9)(a)–(9)(f) allow to compute $Q_{ii'}$. Constraints (9)(a)–(9)(d) store the values of $D_{ii'}$ in the corresponding binary variables $D'_{ii'q}$ and $D''_{ii'q}$. Constraints (9)(e) and (9)(f) allow the computation of $\frac{1}{D_{ii'}}$.

Constraints (10)(a)–(10)(f) and (11)(a)–(11)(e) work in the same way and stand for the linearization of a multiplication. Constraints (10)(a) and (11)(a) store the binary writing of one of the two terms of the multiplication by computing the binary value of each bit. Constraints (10)(b)–(10)(e) and (11)(b)–(11)(e) linearize the multiplication between two variables. To provide a more precise explanation, constraints (10)(b)–(10)(d) and (11)(b)–(11)(d) perform a computation for each bit. This computation involves multiplying one of the members of the multiplication (which can be a real or an integer variable) by the corresponding bit of the other member's binary representation. To illustrate this linearization technique, consider the multiplication $Z = X \times Y$ where $X \in \mathbb{N}$. By the use of variables B representing the binary writing of X such that $X = \left(2^0 B_0\right) + \left(2^1 B_1\right) + \ldots + \left(2^{\bar{g}} B_{\bar{g}}\right)$, we can write the computation of Z as $Z = \left(2^0 B_0 \times Y\right) + \left(2^1 B_1 \times Y\right) + \ldots + \left(2^{\bar{g}} B_{\bar{g}} \times Y\right)$. The considered constraints help with the computation of each term $2^g B_g \times Y$ by computing $B_g \times Y$. At the end, constraints (10)(e) and (11)(e) calculate the final result by summing up the previously computed multiplications for all bits (where each computed multiplication is multiplied by the corresponding value 2^g of the bit).

Constraints (12)(a)–(12)(g) are domain constraints.

4.3. Preprocessing and model improvements

We propose preprocessing steps to make the previous integer linear model easier to solve. First, we find the maximum number of jobs that are pairwise incompatible, and we assign them to different clusters. This helps to reduce the search space as part of the solution is fixed (without loss of generality). Second, we propose to compute $\bar{q}$, the largest number of jobs that a cluster can contain. Indeed, as the number of variables $D'_{ii'q}$ and constraints in $(Q_{ii'})$ follow a quadratic augmentation with $\bar{q}$, we want to find its smallest value. Third, we propose to compute first k^*, and then k. This allows to find more accurate values for both k^* and k by using a very restricted model. Also, this enables to obtain a feasible solution to use as a starting point for the entire model when computing the min–max Hamiltonian cycle with fixed k^* and k values.

4.3.1. Determination of the maximum number of pairwise incompatible jobs

Consider a graph $G = (V, E)$ with vertex set V and edge set E. A vertex set $V' \subset V$ is a clique if all its vertices are pairwise connected. A clique V' is maximal if there is no other clique V'' having a larger cardinality (i.e., more elements) than V'. Based on the correspondence between the GCP and (P) described in Section 3, finding a clique V' in G is equivalent to finding a set of pairwise incompatible jobs $J' \subset J$. Thus, determining the largest set J' of jobs that must belong to different clusters is equal to finding a maximal clique V' in G. Therefore, we can add the constraints $A_{jj} = 1$, $\forall j \in J'$ (i.e., we put each job j in the cluster C_j) to the (GCP) model. Let $X_j^{clique} = 1$ if job j is in the clique; $X_j^{clique} = 0$ otherwise ($\forall j \in J$). The integer linear model for the maximal clique is presented below. As mentioned above, all the jobs of the returned maximal clique will then be put in different clusters of the solution. In this regard, the first constraint set below

forbids two compatible jobs j and j' that can be put in the same cluster of the solution to belong to the clique. Two jobs j and j' can be put in the same cluster of the solution if $b_{jj'} = 1$ and their processing times plus their corresponding setup time do not exceed a shift duration of $\bar{c} = 8$ h.

$$\text{(Clique)} \begin{cases} \max & \sum_{j \in J} X_j^{clique} \\ \text{s.t.} & \\ X_j^{clique} + X_{j'}^{clique} \leq 1 & \forall j, j' \in [\![1, n]\!] \mid j < j', \text{ with } b_{jj'} = 1 \text{ and } p_j + p_{j'} + s^{min} \leq \bar{c} \\ X_j^{clique} \in \{0, 1\} & \forall j \in J \end{cases}$$

4.3.2. Computation of k^* and k

We propose to compute separately the two first objectives (1) and (2), and to fix their values to facilitate the resolution of the entire model. We determine first the value k^* by considering objective (1) and constraints (4)(a)–(4)(c), (4)(e) and (7)(a)–(7)(b). Once k^* is computed, the constraint $\sum_{i \in \mathcal{C}} U_i = k^*$, which we call $(ck*)$, is added to the set of constraints of the (GCP) model. To determine the value of k, we consider objective (2) with constraints (4)(a)–(4)(e) and (7)(a)–(7)(c) along with constraint $(ck*)$. Once k is computed, the constraint $\sum_{i \in \mathcal{C}} E_i = k$, which we call (ck), can also be added to the set of constraints of the (GCP) model.

4.3.3. Reduction of the number of constraints in $(Q_{ii'})$

The largest number of jobs $\bar{q}$ that a cluster can contain is an upper bound for the index q used in several variables of our model. To find the smallest possible value for $\bar{q}$, we solve an integer linear program which can be summarized as the maximization of the number of jobs in the largest cluster, considering all the clustering rules of our problem. We introduce two new types of variable: $Y \in \mathbb{N}$, storing the largest number of jobs contained in the largest cluster, and $X_i = 1$ if cluster C_i is the cluster with the largest number of jobs ($X_i = 0$ otherwise). We maximize the number of jobs contained in the largest cluster using the objective $\max Y$. We consider the constraint set of the above (GCP) model to which we add the constraints $(ck*)$ and (ck) defined in Section 4.3.2 (these two constraints are added in order to further constrain the problem by fixing the two first objectives, which will help to find a more accurate value for $\bar{q}$), along with the following set of constraints:

$$\text{(Largest number of jobs in cluster)} \begin{cases} \bar{q} = \max Y & \\ \text{s.t.} & \\ \sum_{i \in \mathcal{C}} X_i = 1 & \\ Y \geq N_i & \forall i \in \mathcal{C} \\ Y \leq n(1 - X_i) + N_i & \forall i \in \mathcal{C} \\ X_i \in \{0, 1\} & \forall i \in \mathcal{C} \\ Y \in \mathbb{N} & \end{cases}$$

The first constraint ensures that only one cluster is determined as the largest cluster thanks to the variables X_i. The two next constraints make sure that Y stores the number of jobs contained in the largest cluster (i.e. $Y = \max_{i \in \mathcal{C}}(N_i)$). The last two constraints are domain constraints.

5. Solution methods

In this section, we propose solution methods to solve problem (P), which is made of the three subproblems (P1), (P2) and (P3). (P1) and (P2), modeled as a GCP, are solved using an adaptation of the PartialCol algorithm [9] (see Section 5.2). Solving (P3) (i.e., minimizing the longest Hamiltonian cycle) is made using a derivative of the TabuCol algorithm [28] (see Section 5.3). Both PartialCol and TabuCol are state-of-the-art tabu-search algorithms for the GCP, which are considered as efficient (with respect to solution quality) and fast (with respect to the computing time they need to find efficient solutions) metaheuristics able to tackle large instances [38]. A constructive heuristic (denoted as DSATUR, see Section 5.1) is employed for generating initial solutions for PartialCol. As presented in Algorithm 1, the three algorithms (namely, DSATUR, PartialCol and TabuCol) are combined in a single method for tackling the entire problem (P). In Section 5.1, another constructive heuristic (DSAT) is also presented for tackling (P). DSAT represents a good bound on the current-practice rule, as it can be considered as the best possible algorithm that a decision maker would apply manually in the involved company *DIXI*.

In the three-step Algorithm 1, for tackling (P1), DSATUR builds a partial solution C considering the urgent jobs only (i.e., J^*). The number of clusters is then iteratively reduced by removing a cluster C_{i^*} from C and, using PartialCol, by trying to reinsert the jobs of C_{i^*} into the remaining clusters of C. The cluster C_{i^*} to be removed from C is selected such that the

processing time of the longest job in C_{i^*} is minimum (ties are broken randomly). This rule is motivated by the fact that it is generally easier to redistribute (into the remaining clusters) many short jobs instead of a few long jobs. This iterative process stops when a time limit T_1 (parameter) is reached, and the resulting solution for (P1) is the last feasible partition of the jobs of J^* into k^* clusters. For tackling (P2), DSATUR is employed for completing C (with jobs from $J \backslash J^*$) such that it becomes a partition of all the jobs (J) into $k \geq k^*$ clusters. Next, without increasing k^*, we reproduce the same iterative process: the total number k of clusters is iteratively reduced by removing a cluster C_{i^*} from C (with the above rule for selecting C_{i^*}) and by using PartialCol (for redistributing the removed jobs into the remaining clusters). A solution for (P2) with k clusters overall, k^* of which are urgent clusters, is thus found. To tackle (P3), without increasing neither k^* nor k, the length of the longest Hamiltonian cycle over all the clusters is minimized, by redistributing the jobs into the clusters of C with the use of TabuCol.

Algorithm 1 Solution method for problem (P)

input: set of jobs J, set of urgent jobs $J^* \subset J$, processing times and setup times

output: a clustering solution C (i.e., a partition of the jobs into clusters)

▷ Resolution of (P1)

Use DSATUR to build a clustering solution $C = (C_1, C_2, \ldots, C_{k^*})$ of the urgent jobs (J^*)

While a time limit T_1 (parameter) is not met, **do**:

Remove a cluster C_{i^*} from C, and use PartialCol to try to build a solution C' with $k^* - 1$ clusters

If C' **is feasible**, set $C = C'$ and $k^* = \mid C' \mid$

▷ Resolution of (P2)

Use DSATUR to build a clustering solution $C = (C_1, C_2, \ldots, C_k)$ of all the jobs (J) by completing the above solution $(C_1, C_2, \ldots, C_{k^*})$ made of the urgent jobs (J^*) only (with $k \geq k^*$)

While a time limit T_2 (parameter) is not met, **do**:

Remove a cluster C_{i^*} from C, and use PartialCol to try to build a solution C' with $k - 1$ clusters, and not more than k^* urgent clusters

If C' **is feasible**, set $C = C'$ and $k = \mid C' \mid$

▷ Resolution of (P3)

While a time limit T_3 (parameter) is not met, **do**:

Use TabuCol for modifying the clustering solution $C = (C_1, C_2, \ldots, C_k)$ in order to reduce its associated longest Hamiltonian cycle, without augmenting k nor k^*

5.1. Constructive heuristics: DSATUR for (P1) and DSAT for (P)

The constructive heuristic used for generating a job clustering relies on the well-known DSATUR algorithm for the GCP [13]. The saturation degree $dsat(j)$ of a vertex j is defined as the number of different colors that are adjacent to j. The degree $deg(j)$ of vertex j is the number of adjacent vertices to j.

Starting from an empty solution (where no vertex has a color), DSATUR selects at each iteration the next vertex j to be colored, and assigns to it the smallest possible color (i.e., without giving the same color to two adjacent vertices). The selected vertex j has the largest saturation degree; if necessary, ties are broken with the largest degree (in the subgraph made of non-colored vertices only); and the possible remaining ties are broken randomly (which means that two runs of DSATUR may generate two different solutions). Regarding (P), DSATUR is adapted such that the smallest color (cluster) available for each vertex (job) j corresponds to the first cluster C_i for which the setup and capacity constraints are satisfied for $C_i + \{j\}$. DSATUR is used to generate initial solutions for PartialCol in Algorithm 1.

In order to benchmark the results of Algorithm 1, we propose another greedy heuristic (denoted as DSAT) that simulates the best that a decision maker of *DIXI* could do. DSAT is very similar to DSATUR, but it has more priority rules for selecting the next job j to put in the smallest possible cluster. The selected vertex (job) j has the largest processing time (which is a very important criterion in practice, as the longest jobs are the most difficult to schedule). If necessary, ties are broken with the largest saturation degree and next with the largest degree (in the subgraph made of non-colored vertices only).

If necessary, before being broken randomly (which means that two runs of DSAT may generate two different solutions), the remaining ties are broken with the smallest augmentation of the longest Hamiltonian cycle. In our work, the longest Hamiltonian cycle of each clustering is computed using the state-of-the-art ILP (Integer-Linear-Programming) formulation proposed in [16], with iterative subtour elimination constraints [40].

It is important to note that DSATUR and DSAT do not have the same roles and capabilities, even though they are both able to generate a clustering solution $C = (C_1, C_2, \ldots, C_k)$ of jobs in a constructive fashion, based on a set of priority rules. The role of DSATUR is to quickly generate solutions to problems (P1) and (P2), which are then used as initial solutions by PartialCol. The rule set employed by DSATUR aims at minimizing the number of used clusters based on the incompatibility graph, which is important for (P1) and (P2). However, DSATUR ignores the objective function of (P3) when building a solution. In contrast, the role of DSAT is to simulate an efficient decision maker for solving the entire problem (P), i.e., including (P3) as well. In this regard, DSAT has two additional priority rules: one to capture what a decision maker will do in practice (this is why the first priority rule is based on the largest processing times), and the other to cope with the objective function of (P3) (this is why, in line with the lexicographic hierarchy of the objective functions, the last priority rule is linked to f_3). In other words, DSAT is an extension of DSATUR. Given its limited and very specific role, no results will be presented for DSATUR.

5.2. PartialCol for solving (P1) and (P2)

In this section, we present the tabu search PartialCol for the GCP, and its adaptation to (P), where the number of colors (clusters) is fixed to k.

Tabu search [24] starts from an initial solution, and it goes at each iteration from a current solution C to a neighbor solution C'. The modification applied to C leading to C' is called a move. The set of the neighbor solutions of C that can be obtained with the possible moves is denoted as $N(C)$. Once a move is performed, the reverse move is set tabu (i.e., forbidden) and cannot be applied for a certain number of iterations. At each iteration, the best non-tabu move (with respect to an objective function f) is selected in $N(C)$. In general, tabu search stops if a predefined computing time is reached or if an optimal solution is found (assuming the value of an optimal solution is known).

PartialCol for the GCP [9] considers a solution $C = (C_1, C_2, \ldots, C_k; OUT)$, where C_i is the set of non-conflicting vertices with color i, and OUT is the set of uncolored vertices. For the GCP, a conflict occurs if two adjacent vertices have the same color. The solution C corresponds to a partial k-coloring. The function to minimize is $f(C) =\mid OUT \mid$. The algorithm stops either if $f(C) = 0$ (in such a case, the process can be restarted with $k - 1$ colors, and so on, until no feasible k-coloring is found) or if a computing time is reached. A move $m(j, OUT, C_i)$ consists in moving vertex j from OUT to C_i (phase 1 of the move), and if necessary (phase 2 of the move, called the repair phase), in putting in OUT the vertices of C_i that are in conflict with j. The repair phase preserves the feasibility of the resulting neighbor solution (i.e., there is no conflict in each cluster). When such a move is applied, the reverse move $m(j, C_i, OUT)$ is set tabu and cannot be performed for a given number of iterations.

To adapt PartialCol to solve (P1) and (P2), the following modifications are made. First, the repair phase associated with move $m(j, OUT, C_i)$ consists in putting in OUT all the jobs that are incompatible with j (either because of the setup or the capacity constraints). Second, the considered objective function is $\sum_{j \in OUT} p_j$, which corresponds to the remaining workload to dispatch into the clusters. Furthermore, when considering (P2), a move $m(j, OUT, C_i)$ is forbidden if it leads to a solution where the number of urgent clusters is larger than k^* (as it would correspond to a degradation of a higher-level objective). When a move $m(j, OUT, C_i)$ is performed, the reverse move $m(j, C_i, OUT)$ becomes tabu for tab (parameter) iterations. After preliminary experiments, we have decided that the duration tab is generated with a uniform distribution in the interval $[2, \max(5, 1.5c)]$, where c is the current number of clusters. The value $f(C')$ of a neighbor solution C' of the current solution C can be evaluated incrementally by computing the variation $\Delta f(C, C') = f(C') - f(C)$. More precisely, once a move $m(j, OUT, C_i)$ is selected (along with the set J' of jobs to be removed from C_i in the repair phase), we compute $\Delta f(C, C') = \sum_{j' \in J'} p_{j'} - p_j$. In each iteration, the best neighbor candidate solution is selected. Ties are broken randomly, which means that if we run PartialCol two times on the same instance, the sequence of visited solutions is likely to be different.

The pseudo-code of PartialCol is presented in Algorithm 2.

Algorithm 2 PartialCol for (P1) and (P2)

input: a partial solution $C = (C_1, \ldots, C_k; OUT)$, where OUT is not empty

output: a partition of the jobs into k clusters, i.e., a clustering solution $C = (C_1, \ldots, C_k)$ (or an infeasiblility message)

While $OUT \neq \emptyset$ **and** the time limit T_2 (parameter) is not met, **do**:

Perform the best non-tabu move $m(j, OUT, C_i)$

Update the tabu status: move $m(j, C_i, OUT)$ is forbidden for the next tab (parameter) iterations

5.3. TabuCol for solving (P3)

In this section, we present TabuCol for the GCP and its adaptation to (P3), where the number of urgent clusters and the total number of clusters are fixed to k^* and k, respectively.

TabuCol is a tabu search for the GCP [28]. A k-coloring is formulated as $C = (C_1, C_2, \ldots, C_k)$, where C_i is the set of vertices with color i in which conflicts are possible. A move $m(j, C_i, C_{i'})$ consists in moving a vertex j from C_i to $C_{i'}$. When such a move is applied, C_i is tabu for j for *tab* (parameter) iterations. The function to minimize is the number of conflicts in C, where a conflict occurs when two adjacent vertices have the same color (i.e., they belong to the same cluster C_i). If the number of conflicts in C is zero, the process is restarted with $k - 1$ colors, and so on until no feasible solution is found.

To adapt TabuCol to solve (P3), we only consider solutions without conflicts, and the following modifications are made. The repair phase associated with a move $m(j, C_i, C_{i'})$ consists in greedily redistributing (without creating conflicts) into the clusters of $C \backslash C_i$ all the jobs of $C_{i'}$ that are incompatible with j (either because of the setup or the capacity constraint). The objective function $f(C)$ to minimize is the longest (with respect to the setup times among clusters) Hamiltonian cycle covering the clusters of C. In each iteration, the first improving move is selected among a randomly generated proportion $q \in [0, 1]$ (parameter) of neighbor solutions. If there is no improving move, the best of these moves is performed. As for PartialCol, ties are broken randomly. After preliminary experiments, we have decided to set $q = \min\left(1, \frac{5000}{100n}\right)$ and to generate the tabu duration *tab* with a uniform distribution in interval $\left[2, \min\left(2, \frac{c}{10}\right)\right]$ (where c is the current number of clusters).

The pseudo-code of TabuCol is presented in Algorithm 3.

Algorithm 3 TabuCol for (P3)

input: a feasible clustering solution $C = (C_1, \ldots, C_k)$ including k^* urgent clusters

output: the best (according to the minimization of the longest Hamiltonian cycle) encountered solution including k^* urgent clusters among k clusters

While the time limit T_3 (parameter) is not met, **do**:

Perform a non-tabu move $m(j, C_i, C_{i'})$ selected according to some predefined rules

Update the tabu status: moving j back to C_i is forbidden for the next *tab* (parameter) iterations

In order to avoid evaluating multiple times equivalent clusterings, we propose to associate an identifier with each solution C, defined as $H(C) = \sum_{i \in C} \left(\prod_{j \in C_i} j + \sum_{j \in C_i} j\right) |C_i|$. Consequently, when we generate a neighbor solution C', we first compute its identifier $H(C')$. Next, we only evaluate C' if $H(C')$ does not already belong to the dictionary of all the previous identifiers. Otherwise, it means that the value of C' is already known (and there is thus no need to recompute it).

6. Experiments and managerial insights

We present the computational experiments performed on realistic instances generated together with *DIXI* in order to capture the different situations they have to face in practice. First, the instances are presented in Section 6.1. Second, the results for small (resp. large) instances are presented in Section 6.2 (resp. 6.3). Third, aggregated results, sensitivity analyses and managerial insights are presented in Section 6.4. The reader interested in reading the aggregated results first (or only) can simply skip Sections 6.2 and 6.3. The algorithms were implemented in Julia. The MILP solver is CPLEX 20.1.0.0 (with default setting) and the model was coded using Julia for Mathematical Programming [17]. Computational experiments were performed on an Intel Xeon Gold 6240 at 2.60 GHz with a limit of 32 GB of RAM.

In order to cover their needs, *DIXI* has imposed a computing-time limit of 30 min on the considered computer. Both our Algorithm 1 and DSAT have thus such a time limit. However, in order to better assess the performance of the proposed solution methods, we have fixed the time limit for the MILP to 2 h per objective (i.e., 2 h for (P1), (P2) and (P3), which results in 6 h), whereas DSAT is simply restarted for 30 min and the best solution is returned at the end. Regarding Algorithm 1, preliminary experiments showed that the following computing-time setting is appropriate: $(T_1, T_2, T_3) = (5, 5, 20)$ min. Note that, when considering (P1), PartialCol is stopped before the time limit if a solution with $k^* = \left\lceil \frac{\sum_{j \in J^*} p_j}{\bar{p}} \right\rceil$ is found (i.e., a lower bound on k^* is reached), where $\bar{p}$ is the largest processing time of a job (with respect to the considered instance). In such a case, the remaining computing time is added to the time limit of the next objective. Similarly, when considering (P2), PartialCol is stopped if k reaches the lower bound $\left\lceil \frac{\sum_{j \in J} p_j}{\bar{p}} \right\rceil$ and the remaining

computing time is added to the time limit of the next objective. The results reported for Algorithm 1 (which is referred to as PartialCol and TabuCol in the tables) are averages over 10 runs.

In all the proposed meta/heuristics (i.e., DSATUR, DSAT, PartialCol, and TabuCol), equivalent options may be encountered in some iterations (e.g., the best neighbor solutions could have the same value). In such cases, and as indicated in Section 5 when presenting each method, ties are broken randomly. This means that two runs of each of these meta/heuristics may return different solutions with different values. In our experiments, the standard deviations of all the methods except TabuCol are systematically equal to (or very close to) zero. For this reason, only the standard deviations of TabuCol (as well as its best results) are reported and analyzed. In summary, if we run Algorithm 1 ten times, the following behavior is likely to happen: the solution values associated with (P1) and (P2) are likely to be similar (because of the robustness of DSATUR and PartialCol, in particular for the smaller instances), but the solutions values associated with (P3) are likely to be more diverse.

6.1. Instance generation

The following instance set was generated based on the real data provided by *DIXI*. We consider the following four parameters:

- $n \in \{15, 20, 25, 30, 50, 100, 150, 200\}$: the number of jobs,
- $u \in \{0.1, 0.3, 0.5\}$: the ratio of urgent jobs ($u = 0.1$ means that 10% of the jobs are urgent),
- $d \in \{0.2, 0.5, 0.8\}$: the density of the incompatibility graph (as defined below),
- $\bar{p} \in \{240, 360, 450\}$: the largest processing time of a job (in minutes).

The density $d(G)$ of a graph G is defined as the average number of edges between two vertices, where an edge corresponds to an incompatibility (with respect to the setup constraint). Formally, $d(G) = \frac{\sum_{j,j' \in J, j \neq j'} (1 - b_{jj'})}{n(n-1)}$. The incompatibilities among jobs are randomly generated such that the density d is met. As mentioned above, if two jobs j and j' are compatible, the setup time $s_{jj'} = s^{min} = 5$ min. If j and j' are incompatible, the setup time $s_{jj'}$ (in minutes) follows a uniform distribution in the interval [15, 120] min, with steps of 5 min. Note that a setup time of 10 min is not possible because of the machine configurations (i.e., we have either s^{min} or at least 15 min between two jobs). For each job j, the processing time p_j follows a uniform distribution in the interval $[15, \bar{p}]$ min, with steps of 5 min. Therefore, we consider $8 \times 3 \times 3 \times 3 = 216$ instances, labeled from I1 to I216 in the result tables. Note for example (e.g., see Table 1) that instances I1, I2 and I3 have the same values for n, u and d, but different values for $\bar{p}$ (which means that the jobs have different durations).

6.2. Results for the small instances ($n \leq 30$)

The results for instances of size $n \in \{15, 20, 25, 30\}$ are presented in Tables 1–4, respectively. In these tables, we compare DSAT, ILP and PartialCol for (P1) and (P2), and DSAT, ILP and TabuCol for (P3). The left columns indicate the instance ID and its parameters (i.e., u, d, and $\bar{p}$). For (P1) (resp. (P2)), the obtained value of k^* (resp. k) is provided. Regarding (P3), the obtained objective values are denoted by H (which holds for "Hamiltonian"). As results are averaged over 10 runs for TabuCol, we also indicate the best-encountered value (denoted as $H^{(best)}$) and the standard deviation (denoted as σ). The times are indicated in seconds. To only present interesting information, we do not indicate the times in the tables for $n \leq 30$ for (P1) and (P2) as they are consistently near 0 s. Note that each indicated time refers to the considered instance, subproblem (among (P1), (P2), and (P3)), and stopping condition (as explained at the beginning of this section, there is a time limit for each method and subproblem). For example, considering instance I1, the time at which the optimal solution is found is 0 s for (P1) (knowing that 7200 s were available), 0 s also for (P2), and 7196.36 s were employed to find the best solution for (P3) (without proving optimality). Optimal values are indicated in bold characters. The average results for $n = 30$ (i.e., the last line of Table 4) do not consider instance I93 for (P3) as ILP could not find a solution.

The following observations can be made.

- For (P1) and (P2), optimal values are found instantaneously by ILP and PartialCol. DSAT finds 84.6% of the optimal values for (P1) and 36.11% for (P2).
- For (P3), the performance of ILP is the following. The percentage of instances for which ILP proves optimality quickly decreases with the augmentation of n (i.e., 37.04% for $n = 15$, 22.22% for $n = 20$, 3.70% for $n = 25$, and 0% for $n = 30$). Over all the small instances, ILP proves optimality for only 15.74% of the instances (17 out of 108). The time needed for ILP to find its last solution is large and increases with the augmentation of n. On average, it is around 26 min for $n = 15$ and it goes up to 64 min for $n = 30$. From $n = 30$, ILP starts not to be able to find a feasible solution for all instances (see instance I93 in Table 4).
- For (P3), the performance of TabuCol is the following. On average, TabuCol finds results improved by 12% and 15% compared to ILP and DSAT, respectively. TabuCol is able to find at each run all optimal solutions found by ILP, except for instances I31 and I52 for which TabuCol reaches the optimal solution only 1 and 2 times over the 10 runs, respectively. As presented in Table 5 (which gives aggregated results with respect to various robustness indicators), the standard deviation σ remains low and increases slightly with the augmentation of n (see column 2 labeled as "average σ"). Unsurprisingly, the percentage of instances for which $\sigma = 0$ decreases quickly with the augmentation

Table 1
Results for the instances with $n = 15$.

ID	u	d	$\bar{p}$	ILP	DSAT	PartialCol	ILP	DSAT	PartialCol	ILP		DSAT	TabuCol			
				k^*	k^*	k^*	k	k	k	H	Time	H	H	$H^{(best)}$	Time	σ
I1	0.1	0.2	240	**1**	**1**	**1**	**4**	**4**	**4**	145.6	7196.36	114.67	87.7	87.7	0.66	0
I2			360	**1**	**1**	**1**	**7**	**7**	**7**	331	5921.73	279.58	184.05	183.78	0.28	0.46
I3			450	**1**	**1**	**1**	**9**	**9**	**9**	413.8	3385.92	296.25	248.65	240	0.05	3.83
I4		0.5	240	**1**	**1**	**1**	**6**	**6**	**6**	393.3	5870.6	342.08	280.3	280.3	0.26	0
I5			360	**1**	**1**	**1**	**10**	**10**	**10**	717.4	965.15	625	579.61	573.7	0.02	4.15
I6			450	**1**	**1**	**1**	**10**	**10**	**10**	767.5	178.6	727.5	696.3	696.3	0.03	0
I7		0.8	240	**1**	**1**	**1**	**8**	**8**	**8**	**613.7**	1987.99	665	**613.7**	**613.7**	0.03	0
I8			360	**1**	**1**	**1**	**8**	9	**8**	**661.7**	9.17	819.38	**661.7**	**661.7**	0.02	0
I9			450	**1**	**1**	**1**	**8**	**8**	**8**	**661.2**	1.45	669.58	**661.2**	**661.2**	0.02	0
I10	0.3	0.2	240	**1**	**1**	**1**	**4**	**4**	**4**	75.3	7183.02	89.37	65	65	0.05	0
I11			360	**2**	**2**	**2**	**6**	**6**	**6**	261	1643.81	152.08	140.1	140.1	0.1	0
I12			450	**4**	**4**	**4**	**8**	9	**8**	252.4	1558.26	336.25	177.21	176.4	0.05	2.56
I13		0.5	240	**2**	**2**	**2**	**5**	7	**5**	240.3	0.99	367.5	240.47	240.47	0	0
I14			360	**2**	**2**	**2**	**9**	10	**9**	626.8	233.02	727.5	575	575	0.03	0
I15			450	**3**	**3**	**3**	**11**	**11**	**11**	942.5	262.84	842.5	786.5	785	0.04	1.29
I16		0.8	240	**3**	**3**	**3**	**7**	8	**7**	484.5	1.41	593.75	484.5	484.5	0.01	0
I17			360	**4**	**4**	**4**	**9**	10	**9**	**729.2**	163.26	851.25	**729.2**	**729.2**	0.03	0
I18			450	**3**	**3**	**3**	**10**	11	**10**	**911.3**	3.91	1007.5	**911.3**	**911.3**	0	0
I19	0.5	0.2	240	**2**	**2**	**2**	**4**	5	**4**	**89.8**	403.21	123.99	**89.8**	**89.8**	0.03	0
I20			360	**3**	**3**	**3**	**8**	**8**	**8**	397.5	587.11	285.83	232	232	0.02	0
I21			450	**3**	4	**3**	**8**	**8**	**8**	269.5	3279.52	251.25	238	235.8	0.03	2.84
I22		0.5	240	**4**	**4**	**4**	**5**	**5**	**5**	**233.5**	7.98	233.83	**233.5**	**233.5**	0	0.07
I23			360	**4**	**4**	**4**	**9**	**9**	**9**	566.3	575.23	538.75	458.8	458.8	0.03	0
I24			450	**5**	**5**	**5**	**9**	10	**9**	622	67.36	660	533.7	533.7	0	0
I25		0.8	240	**5**	**5**	**5**	**8**	9	**8**	**557.6**	88.85	670	**557.6**	**557.6**	0.02	0
I26			360	**5**	**5**	**5**	**9**	**9**	**9**	**786.2**	3.95	797.5	**786.2**	**786.2**	0.02	0
I27			450	**5**	**5**	**5**	**10**	**10**	**10**	**900**	21.16	911.25	**900**	**900**	0.03	0
Average results				2.56	2.59	2.56	7.74	8.15	7.74	505.59	1540.81	517.75	450.08	449.36	0.07	0.56

of n (see column 3 labeled "$\sigma = 0$"). The average standard deviation corresponds to less than 1% of the average objective-function values (see column 4). We can also observe (see the last column) that the worst-case variations of H remain reasonable and they do not suffer from the augmentation of n.

- DSAT is the quickest method to generate a relatively good solution, but it is less efficient than ILP and PartialCol/TabuCol. For (P1) (resp. (P2)), DSAT finds objective-function values that are on average 2.51% (resp. 6.36%) higher than the ones obtained by both ILP and PartialCol. For (P3), such percentage gaps increase to 4.39% when compared to ILP, and to 26.74% when compared to TabuCol. DSAT is not able to propose better results than PartialCol/TabuCol, except for three instances (I50, I75 and I78) for which the DSAT solutions obtained for a previous objective are not optimal and thus the resolution of (P2) and (P3) is less restricted.

6.3. Results for the large instances ($n \geq 50$)

The results for instances of size $n \in \{50, 100, 150, 200\}$ are presented in Tables 6–9, respectively, which have globally the same structure as the previous tables. To only present interesting information, we note however the following straightforward format modifications: (1) the results for (P3) obtained with ILP are shown only for $n = 50$ (as no solution at all has been found before the time limit for $n \geq 100$); (2) for $n \geq 100$, the computing times for ILP and PartialCol are now indicated for (P1) and (P2), as they are not consistently near 0 s anymore. Two lines are proposed to present the average results: the first one considers only the instances for which ILP found a solution, whereas the second one considers all instances. It is important to remind here that ILP can have the following outputs.

- An optimal solution is found. In such a case, the objective-function value is indicated in bold character. If a time is indicated, it corresponds to the computing time needed to find the optimal solution (without considering the time to prove optimality).
- A non-necessarily optimal solution is found (i.e., either the solution is not optimal, or the solution is optimal but CPLEX was not able to prove it). If a time is indicated, it refers to the computing time needed by CPLEX to find its best solution (while considering a time limit of 7200 s per subproblem). It can thus happen that the best solution is found very early (e.g., after 2.65 s for instance I140), and then no better solution is found (i.e., for the remaining 7197.35 s regarding instance I140).

Table 2
Results for the instances with $n = 20$.

ID	u	d	$\bar{p}$	ILP	DSAT	PartialCol	ILP	DSAT	PartialCol	ILP		DSAT	TabuCol			
				k^*	k^*	k^*	k	k	k	H	Time	H	H	$H^{(best)}$	Time	σ
I28	0.1	0.2	240	**1**	**1**	**1**	**7**	8	**7**	217.6	1910.76	229.86	140.47	140.47	0.7	0
I29			360	**1**	**1**	**1**	**9**	**9**	**9**	364.1	614.82	296.24	208.3	208.3	5.47	0
I30			450	**1**	**1**	**1**	**12**	**12**	**12**	545	157.85	467.5	418.8	418.8	0.06	0
I31		0.5	240	**2**	**2**	**2**	**6**	8	**6**	**257.6**	837.4	437.91	273.62	**257.6**	0.02	13.76
I32			360	**2**	**2**	**2**	**9**	10	**9**	593.8	7116.98	595.42	416.6	413.15	4.6	4.69
I33			450	**2**	**2**	**2**	**11**	**11**	**11**	819.5	3761.2	707.5	649.68	633	0.05	17.58
I34		0.8	240	**2**	**2**	**2**	**9**	10	**9**	**632.9**	29.17	780.83	**632.9**	**632.9**	0.02	0
I35			360	**2**	**2**	**2**	**11**	**11**	**11**	991.3	115.51	963.75	847.35	843.7	0.04	5.67
I36			450	**2**	**2**	**2**	**13**	14	**13**	1267	3331.86	1305	1192.6	1192.6	0.04	0
I37	0.3	0.2	240	**3**	**3**	**3**	**6**	7	**6**	194.5	5176.03	210.55	113.4	113.4	0.18	0
I38			360	**4**	**4**	**4**	**11**	**11**	**11**	562.1	5351.32	490.41	354.44	345.65	0.41	12.85
I39			450	**3**	**3**	**3**	**9**	11	**9**	283.3	92.44	436.11	240.7	240.7	0.02	0
I40		0.5	240	**3**	**3**	**3**	**6**	**6**	**6**	327.9	7124.77	296.41	284.73	277.47	0.04	7.66
I41			360	**3**	**3**	**3**	**9**	**9**	**9**	545.5	1487.32	468.19	411.45	411.45	0.26	0
I42			450	**4**	**4**	**4**	**12**	**12**	**12**	1018	1147.5	870	793	793	0.08	0
I43		0.8	240	**5**	**5**	**5**	**10**	12	**10**	795.9	39.14	1021.25	732.4	732.4	0.03	0
I44			360	**4**	**4**	**4**	**10**	**10**	**10**	**749.5**	2251.62	788.75	**749.5**	**749.5**	0.03	0
I45			450	**5**	**5**	**5**	**14**	**14**	**14**	1321.3	89.58	1251.25	1218.7	1218.7	0.06	0
I46	0.5	0.2	240	**3**	4	**3**	**6**	6	**6**	204.2	7169.15	165.31	142.9	142.9	0.04	0
I47			360	**4**	**4**	**4**	**9**	**9**	**9**	450.4	1250.36	360	213.06	207.2	0.24	6.82
I48			450	**7**	**7**	**7**	**11**	**11**	**11**	549.8	344.72	518.34	401.28	395	0.08	10.19
I49		0.5	240	**5**	6	**5**	6	8	**6**	**295.1**	0.9	440.28	**295.1**	**295.1**	0	0
I50			360	**5**	6	**5**	**8**	8	**8**	483.6	93.72	419.3	421.6	421.6	0.03	0
I51			450	**6**	**6**	**6**	**11**	12	**11**	685.8	2200.67	792.5	636.2	636.2	0.03	0
I52		0.8	240	**6**	**6**	**6**	**10**	11	**10**	**696.2**	9.97	878.75	726.72	**696.2**	0.02	15.98
I53			360	**8**	**8**	**8**	**13**	14	**13**	**1127.5**	529.62	1306.25	**1127.5**	**1127.5**	0	0
I54			450	**8**	**8**	**8**	**15**	**15**	**15**	1507.5	306.56	1493.75	1432.6	1432.6	0.04	0
Average results				3.74	3.85	3.74	9.74	10.33	9.74	647.66	1945.96	666.35	558.36	554.71	0.47	3.53

- No solution is found (which corresponds to empty cells in the result tables).
- It should be noted that the bounds obtained by CPLEX are not shown as they do not provide relevant insights for the analysis (i.e., when optimal solutions are not found, the quality of the bounds is here usually very poor).

The following observations can be made.

- For (P1), ILP is not always able to prove optimality for $n \geq 150$. With respect to the augmentation of u and n, the percentage of instances solved to optimality decreases quickly (see the left part of Table 10), whereas the time needed by ILP to find its last solution increases quickly (see the right part of Table 10).
- For (P1) again, PartialCol is able to find all the optimal solutions found by ILP. The time needed by PartialCol is usually below 1 s (except for six instances). The results obtained by DSAT are on average 11.69% higher than the results of ILP, whereas PartialCol averagely improves the results of ILP by 0.64% (more precisely, 0.00% for $n \leq 100$, 0.31% for $n = 150$, and 2.25% for $n = 200$). We can see that the time needed by PartialCol to find its best solutions is relatively constant with the augmentation of n.
- For (P2), the performance of ILP is the following. It is able to find 24.07% of optimal solutions (for 26 out of 108 instances). ILP finds 92.59% (25 out of 27) of optimal solutions for $n = 50$, and 3.70% (1 out of 27) for $n = 100$. From $n = 150$, no more optimal solution is found by ILP and a solution is found for only 22.22% of the instances (12 out of 54). More precisely, for $n = 150$, ILP is able to find a solution for 29.63% (8 out of 27) of the instances (all of them are obtained for instances with $u = 0.1$, which can be explained by the fact that it is easier for ILP to find a feasible solution when the problem is less constrained). For $n = 200$, a solution is found for 14.81% (4 out of 27) of the instances. The average time needed by ILP to find its last solution is 38.3 s for $n = 50$ and 4252.58 s for $n = 100$. The average time needed for $n = 150$ and $n = 200$ is not relevant because of the low number of instances solved.
- For (P2) again, PartialCol is able to find all the optimal solutions found by ILP and finds on average its last solution in 25.63 s (more precisely, 21.07 s for $n = 50$, 17.05 s for $n = 100$, 30.26 s for $n = 150$, and 31.14 s for $n = 200$). On average, PartialCol improves the results of ILP by 12.99% (i.e., 0.00% for $n = 50$, 5.95% for $n = 100$, 54.77% for $n = 150$, and 62.88% for $n = 200$), whereas the objective-function values obtained by DSAT are averagely 5.22% higher than the ones of ILP for $n \in \{50, 100\}$ (i.e., 9.86% for $n = 50$, and 0.39% for $n = 100$), but 52.22% lower for $n \geq 150$ (i.e., 51.56% for $n = 150$, and 53.55% for $n = 200$).

Table 3
Results for the instances with $n = 25$.

ID	u	d	$\bar{p}$	ILP	DSAT	PartialCol	ILP	DSAT	PartialCol	ILP		DSAT	TabuCol			
				k^*	k^*	k^*	k	k	k	H	Time	H	H	$H^{(best)}$	Time	σ
I55	0.1	0.2	240	**1**	**1**	**1**	**9**	10	**9**	299.4	4910.21	323.33	179.33	179.2	47.12	0.13
I56			360	**2**	**2**	**2**	**9**	**9**	**9**	313.2	1626.75	255.9	190	190	6.83	0
I57			450	**1**	**1**	**1**	**12**	**12**	**12**	603.3	4577.73	461.24	365.21	364	2.13	1.64
I58		0.5	240	**2**	**2**	**2**	**7**	8	**7**	354.2	6700.39	411.48	308.53	305.68	0.03	8.99
I59			360	**2**	**2**	**2**	**10**	11	**10**	600	424.92	656.68	473.2	473.2	1.22	0
I60			450	**1**	**1**	**1**	**13**	**13**	**13**	942.4	4093.11	930.01	701.44	695.45	0.24	12.63
I61		0.8	240	**2**	**2**	**2**	**11**	12	**11**	877.9	1662.06	965.84	745.72	745.72	0.07	0
I62			360	**2**	**2**	**2**	**16**	17	**16**	1384.9	3966.9	1510	1321.3	1321.3	0.06	0
I63			450	**2**	**2**	**2**	**17**	**17**	**17**	1624.9	2191.25	1550.42	1485.8	1485.8	0.06	0
I64	0.3	0.2	240	**3**	**3**	**3**	**6**	7	**6**	152.7	6951.62	160.11	93	93	33.62	0
I65			360	**4**	**4**	**4**	**12**	**12**	**12**	545.1	5659.13	447.08	368.8	368.8	0.55	0
I66			450	**3**	**3**	**3**	**11**	**11**	**11**	492.1	6743.95	451.88	281.88	263.08	27.12	8.07
I67		0.5	240	**4**	**4**	**4**	**8**	9	**8**	435.5	5757.36	472.51	357.17	357.17	1.86	0
I68			360	**3**	**4**	**3**	**11**	12	**11**	724.3	1126.84	778.54	578.23	577.85	114.55	1.21
I69			450	**4**	**4**	**4**	**15**	**15**	**15**	1272.7	4981.56	1176.66	1034.74	1033.7	0.08	1.34
I70		0.8	240	**5**	**5**	**5**	**12**	14	**12**	1033.8	5661.6	1262.5	894.2	894.2	0.05	0
I71			360	**6**	**6**	**6**	**13**	14	**13**	1127.6	53.57	1187.5	1095.83	1060.1	0.02	30.61
I72			450	**5**	**5**	**5**	**13**	14	**13**	1084.7	7006.79	1163.75	958.43	954.55	0.07	12.25
I73	0.5	0.2	240	**3**	4	**3**	**6**	7	**6**	178.1	104.64	219.58	128.66	128	0.13	0.57
I74			360	**6**	**6**	**6**	**13**	**13**	**13**	612.1	2759.05	572.64	467.22	463.5	0.33	4.63
I75			450	**6**	7	**6**	**13**	13	**13**	669.6	6600.8	500.42	504.3	504.3	0.16	0
I76		0.5	240	**5**	**5**	**5**	**8**	9	**8**	471.5	6633.18	509.72	350.15	347.7	0.26	7.75
I77			360	**7**	**7**	**7**	**12**	**12**	**12**	818.2	194.65	757.51	695.8	695.8	0.1	0
I78			450	**6**	7	**6**	**16**	15	**16**	1280	2849	1050	1183.8	1183.8	0.07	0
I79		0.8	240	**7**	**7**	**7**	**11**	12	**11**	856.9	7198.55	982.22	776.3	776.3	0.07	0
I80			360	**8**	**8**	**8**	**13**	14	**13**	**1070**	37.76	1250	**1070**	**1070**	0.02	0
I81			450	**11**	**11**	**11**	**20**	**20**	**20**	1951.6	1545.9	1959.16	1912.5	1912.5	0.03	0
	Average results			4.11	4.26	4.11	11.74	12.3	11.74	806.54	3778.49	813.58	685.98	683.14	8.77	3.33

- For (P3), ILP is only able to find 11.11% of feasible solutions for $n = 50$ (3 out of 27 instances) and the average time needed to find its last solutions is very high (6505.05 s). For $n \geq 100$, ILP is not able to find any solution within the time limit of two hours.
- For (P3) again, TabuCol finds on average its last solution in 416.58 s (i.e., 139.11 s for $n = 50$, 379.92 s for $n = 100$, 566.17 s for $n = 150$, and 581.13 s for $n = 200$). We can see that the average time needed by TabuCol increases for the largest instances but still remains around 10 min. As ILP finds only three solutions, the performance of TabuCol is only compared with the performance of DSAT: TabuCol averagely improves the results of DSAT by 22.60% (i.e., 23.43% for $n = 50$, 18.42% for $n = 100$, 13.54% for $n = 150$, and 35.03% for $n = 200$). The aggregated results with respect to various robustness indicators are given in Table 11 (which is similar in its structure to Table 5). We can observe that the standard deviation σ is on average low even for the higher values of n (see the second column), whereas the percentage of instances for which $\sigma = 0$ is 7.41% for $n = 50$ and is null for $n \geq 100$ (see the third column labeled "$\sigma = 0$"). The value of σ corresponds on average to 1.03% of the average objective value (see the fourth column). It seems that a peak is reached for $n = 100$ and that for lower values of n, solutions are somewhat easier to find, whereas for higher values of n, the convergence to local optima is more stable. The biggest variation of σ is relatively low and corresponds to 6.09% of the objective-function value for $n = 100$ (see the last column).

6.4. Aggregated results, sensitivity analyses, and managerial insights

Table 12 presents the aggregated results of all the proposed methods for problems (P1), (P2), and (P3) (built from the last lines of Tables 1 to 9). Each line represents averaged results over 27 instances (i.e., covering different values for the parameters u, d, and $\bar{p}$) for the considered number n of jobs. The following information is displayed.

- k^*: number of clusters containing at least one urgent job, for each method related to problem (P1).
- k: total number of clusters, for each method related to problem (P2).
- Time: computing time of the corresponding algorithm (in seconds) to provide its best solution. "Time" is not given for DSAT as it is consistently near zero.
- SOL: number of instances (among 27) for which ILP is able to generate a solution (to the considered problem), but without proving its optimality.

Table 4
Results for the instances with $n = 30$.

ID	u	d	$\bar{p}$	ILP	DSAT	PartialCol	ILP	DSAT	PartialCol	ILP		DSAT	TabuCol			
				k^*	k^*	k^*	k	k	k	H	Time	H	H	$H^{(best)}$	Time	σ
I82	0.1	0.2	240	**2**	**2**	**2**	**8**	9	**8**	239.8	242.27	281.63	163.13	162.9	513.04	0.5
I83			360	**2**	**2**	**2**	**12**	13	**12**	431.1	6985.35	491.35	269.15	268.77	19.06	1.2
I84			450	**3**	**3**	**3**	**18**	**18**	**18**	836.8	4629.36	797.5	685.61	671.7	1.04	11.11
I85		0.5	240	**2**	**2**	**2**	**9**	11	**9**	446.1	6614.14	638.63	391.93	391.6	1.08	0.53
I86			360	**2**	**2**	**2**	**15**	16	**15**	1053.8	571.24	1140	1018.51	1012.35	0.03	3.25
I87			450	**2**	**2**	**2**	**17**	**17**	**17**	1299.2	6697.7	1342.5	1109.64	1097.6	0.28	8.81
I88		0.8	240	**3**	**3**	**3**	**13**	15	**13**	1066.6	3208.25	1310.41	991.24	987.55	0.09	7.9
I89			360	**2**	**2**	**2**	**18**	**18**	**18**	1616.3	1281.73	1580	1507.84	1506.4	0.07	2.32
I90			450	**2**	**2**	**2**	**20**	21	**20**	1937.1	3600.22	2071.25	1851.4	1851.4	0.15	0
I91	0.3	0.2	240	**3**	**3**	**3**	**8**	9	**8**	251.1	1456.33	298.35	174.4	174.4	40.01	0
I92			360	**6**	**6**	**6**	**14**	**14**	**14**	562.5	5729.73	524.31	399.72	392.97	4.05	5.63
I93			450	**5**	**5**	**5**	**20**	**20**	**20**	–	–	1085.42	1007.37	1000	0.53	7.53
I94		0.5	240	**4**	**4**	**4**	**8**	9	**8**	428.9	6904.41	464.05	357.72	352.23	0.78	9.7
I95			360	**5**	**5**	**5**	**12**	**12**	**12**	690.6	610.33	686.17	578.33	578.3	2.45	0.06
I96			450	**7**	**7**	**7**	**18**	**18**	**18**	1339.9	6029.58	1385	1190.03	1183.7	0.62	4.55
I97		0.8	240	**5**	**5**	**5**	**13**	15	**13**	1078.8	5327.01	1199.58	956.72	951.8	0.13	6.35
I98			360	**5**	**5**	**5**	**16**	**16**	**16**	1319.1	986.26	1327.09	1247.9	1247.9	0.06	0
I99			450	**7**	**7**	**7**	**23**	**23**	**23**	2342.5	6170.26	2342.5	2320	2320	0.02	0
I100	0.5	0.2	240	**5**	6	**5**	**9**	10	**9**	281.8	6541.2	314.92	191.83	191.83	4.29	0
I101			360	**6**	**6**	**6**	**12**	**12**	**12**	459.3	1952.8	453.21	327.17	326.8	25.21	1.17
I102			450	**9**	**9**	**9**	**16**	**16**	**16**	815.1	4369.98	754.58	594.67	584.9	0.83	12.11
I103		0.5	240	**6**	7	**6**	**9**	10	**9**	432.2	3543.73	509.78	425.8	425.8	0.01	0
I104			360	**6**	7	**6**	**12**	14	**12**	752.5	1801.76	970.84	714.8	714.8	0.02	0
I105			450	**9**	10	**9**	**18**	**18**	**18**	1226.3	6094.52	1259.16	1215.1	1215.1	0.15	0
I106		0.8	240	**8**	**8**	**8**	**13**	14	**13**	1103.5	7200.03	1163.74	991.9	991.9	0.19	0
I107			360	**10**	**10**	**10**	**17**	18	**17**	1622.1	494.91	1687.09	1538.6	1538.6	0.12	0
I108			450	**9**	**9**	**9**	**16**	17	**16**	1418.2	370.16	1529.17	1376.63	1362.5	0.03	22.75
Average results				5	5.15	5	14.22	14.93	14.22	963.51	3823.59	1020.11	868.84	865.53	23.61	3.77

Table 5
For TabuCol on (P3), evolution of σ and other robustness indicators depending on n.

n	Average σ	$\sigma = 0$	Average variation of H	Biggest variation of H
15	0.56	74.07%	0.20%	1.54%
20	3.53	66.67%	0.88%	5.03%
25	3.33	55.56%	0.60%	2.91%
30	3.91	37.04%	0.54%	2.71%

- OPT: number of instances (among 27) for which ILP is able to generate a solution (to the considered problem) and to prove its optimality.
- H (which holds for "Hamiltonian"): objective-function value for (P3).
- $H^{(best)}$ (for TabuCol and (P3) only): average of the best objective-function values. This can be computed as for each instance, Algorithm 1 (and thus its TabuCol component) is run 10 times.
- σ: average standard deviation for TabuCol on (P3).

One can easily observe the good behavior of PartialCol/TabuCol with respect to the increase of the instance size n. More precisely, in contrast with ILP and DSAT, PartialCol and TabuCol do not suffer with the increase of the number of jobs, as such metaheuristics can still generate efficient solutions (with respect to objective-function values) quickly (with respect to the allowed computing times). The following more detailed observations can be made regarding the performance of each method for each problem.

- (P1) All the methods are comparable when $n < 150$. However, from $n = 150$, ILP requires much more computing time, and it cannot always prove the optimality of the provided solutions. ILP and PartialCol slightly outperform DSAT regarding solution quality, in particular from $n = 100$.
- (P2) ILP is not efficient anymore from $n = 150$ regarding solution quality, and from $n = 100$ regarding speed. The results of DSAT are close to those of PartialCol, except for $n = 200$ where PartialCol is obviously much better regarding solution quality, while still having relatively small computing times (roughly half a minute).
- (P3) For the large instances (i.e., from $n = 100$), ILP cannot generate any solution, except for three (out of 27) instances with $n = 50$ while having huge computing times. The solutions of DSAT are clearly outperformed by those

Table 6
Results for the instances with $n = 50$.

				ILP	DSAT	PartialCol	ILP		DSAT	PartialCol	ILP		DSAT	TabuCol			
ID	u	d	$\bar{p}$	k^*	k^*	k^*	k	Time	k	k	H	Time	H	H	$H^{(best)}$	Time	σ
I109			240	**2**	**2**	**2**	**14**	20.65	15	**14**	–	–	504.45	318.39	315.27	650.25	1.87
I110		0.2	360	**2**	**2**	**2**	**18**	17.75	19	**18**	–	–	718.47	505.34	504.7	480.24	1.35
I111			450	**3**	**3**	**3**	**25**	9.67	26	**25**	–	–	1313.34	1055.23	1044.4	3.95	17.69
I112			240	**2**	3	**2**	**13**	9.97	17	**13**	–	–	1052.02	616.55	587.05	122.67	21.81
I113		0.5	360	**4**	**4**	**4**	**18**	16.02	21	**18**	–	–	1330.97	875.24	869.75	513.06	3.49
I114	0.1		450	**3**	**3**	**3**	**26**	19.17	27	**26**	–	–	2060.84	1688.88	1672.4	6.79	16.07
I115			240	**4**	**4**	**4**	**19**	79.19	23	**19**	1536.1	6409.71	1935.97	1468.57	1439.15	0.96	31.63
I116		0.8	360	**4**	**4**	**4**	**20**	7.06	25	**20**	–	–	2243.33	1642.81	1563.2	0.12	29.4
I117			450	**4**	**4**	**4**	**29**	7.68	30	**29**	–	–	2925	2677.46	2671.3	3.35	6.3
I118			240	**4**	5	**4**	**13**	8.5	14	**13**	–	–	468.33	290.39	286.2	539.9	2.08
I119		0.2	360	**7**	8	**7**	**22**	11.64	23	**22**	–	–	980.84	781.3	781.3	1.07	0
I120			450	**9**	**9**	**9**	**26**	11.56	**26**	**26**	–	–	1240.28	1027.82	998.6	14.55	16.68
I121			240	**6**	7	**6**	14	19.73	15	14	–	–	818.96	638.96	635.6	592.36	2.91
I122		0.5	360	**6**	8	**6**	**25**	14.11	25	**25**	–	–	1905.85	1621.98	1581.75	4.36	29.39
I123	0.3		450	**7**	8	**7**	**29**	15.11	29	**29**	–	–	2434.16	2207.23	2185	3.86	19.45
I124			240	**7**	8	**7**	**20**	3.57	24	**20**	1562.3	6768.32	2047.09	1509.9	1509.9	2.68	0
I125		0.8	360	**9**	10	**9**	**21**	3.48	23	**21**	–	–	1942.3	1657.54	1639.45	0.28	29.3
I126			450	**10**	**10**	**10**	**28**	3.24	30	**28**	–	–	2912.91	2426.54	2418.3	4.31	10.68
I127			240	**7**	8	**7**	**14**	11.11	15	**14**	–	–	474.52	312.84	311.13	640.54	1.2
I128		0.2	360	**10**	11	**10**	**21**	11.76	22	**21**	–	–	888.68	647.06	641.3	108.75	2.56
I129			450	**13**	**13**	**13**	**25**	17.2	**25**	**25**	–	–	1215.01	953.77	930.9	28.59	27.19
I130			240	**7**	10	**7**	15	646.25	18	15	–	–	1076.04	755.89	754.35	3.74	1.99
I131		0.5	360	**12**	13	**12**	**20**	22.08	22	**20**	–	–	1443.88	1148.87	1120.4	15.97	32.86
I132	0.5		450	**13**	14	**13**	**25**	23.61	26	**25**	–	–	1973.54	1573.57	1548.2	10.25	18.57
I133			240	**11**	14	**11**	**20**	3.12	25	**20**	1537.5	6340.13	2154.16	1474.67	1452.8	2.26	20.15
I134		0.8	360	**12**	13	**12**	**22**	2.69	26	**22**	–	–	2356.24	1822.9	1803.7	0.02	20.17
I135			450	**15**	**15**	**15**	**27**	18.27	28	**27**	–	–	2532.5	2359.73	2331.5	1.14	27.26
	Average results (ILP)			7.15	7.89	7.15	21.07	38.3	22.93	21.07	1545.3	6506.05	2045.74	1484.38	1467.28	1.97	17.26
	Average results (all)			7.15	7.89	7.15	21.07	38.3	22.93	21.07	1545.3	6506.05	1590.73	1261.46	1244.36	139.11	14.52

Table 7
Results for the instances with $n = 100$.

				ILP		DSAT	PartialCol		ILP		DSAT	PartialCol		DSAT	TabuCol			
ID	u	d	$\bar{p}$	k^*	Time	k^*	k^*	Time	k	Time	k	k	Time	H	H	$H^{(best)}$	Time	σ
I136			240	**3**	0	**3**	**3**	0.04	28	2614.73	29	28	0	1037.37	775.7	767.45	642.66	5.37
I137		0.2	360	**4**	0.01	**4**	**4**	0.03	40	1739.68	40	40	0.02	1860.37	1449.67	1432.3	561.61	9.81
I138			450	**6**	0	**6**	**6**	0	55	2326.21	55	55	0	3576.5	2979.5	2882.3	131.24	113.33
I139			240	**4**	0	5	**4**	0.06	29	6455.42	32	27	4.37	1950.44	1357.04	1346.85	667.98	6.01
I140		0.5	360	**6**	0	**6**	**6**	0	96	2.65	44	41	0.28	3193.34	2586.91	2536.7	445.36	22.25
I141	0.1		450	**6**	0	**6**	**6**	0	96	2.82	57	55	0.03	5094.58	4437.58	4395	133.44	33.31
I142			240	**6**	0	**6**	**6**	0.02	34	4113.06	39	33	3.69	3314.3	2469.65	2453.27	684.64	8.26
I143		0.8	360	**7**	0	**7**	**7**	0.06	41	4359.32	47	41	1	4269.3	3304.1	3279.5	475.93	15.43
I144			450	**7**	0	**7**	**7**	0	52	5975.27	54	51	0.01	5214.58	4643.55	4618	227.89	20.95
I145			240	**9**	0.22	11	**9**	0.04	29	2123.2	28	28	0	939.62	760.02	754	599.28	4.23
I146		0.2	360	**13**	1.35	**13**	**13**	0.06	39	2540.48	40	39	0	1794.23	1421.8	1407.7	492.39	6.89
I147			450	**15**	0.67	**15**	**15**	0.06	55	4424.41	56	55	0.01	3698.75	3038.4	2910.9	156.18	185.03
I148			240	**9**	2.72	10	**9**	0.06	26	7111.21	30	26	0.18	1779.66	1287.07	1276	524.27	5.81
I149		0.5	360	**14**	2.4	**14**	**14**	0.06	–	–	46	44	0.02	3393.88	2905.98	2856.9	464.98	23.95
I150	0.3		450	**20**	0.85	21	**20**	0.06	51	4608.26	52	51	0.48	4214.86	3766.25	3704	124.11	30.24
I151			240	**12**	0.34	15	**12**	0.07	36	5106.51	40	34.9	104.18	3381.14	2645.45	2632.15	561.99	18.22
I152		0.8	360	**15**	1.34	17	**15**	0.07	46	4683.05	49	45	16.41	4549.02	3744.68	3713.65	91.19	29.71
I153			450	**19**	1.05	20	**19**	0.06	56	2255.32	57	56	0	5713.61	5401.86	5375.1	74.77	19.8
I154			240	**14**	14.85	16	**14**	0.06	27	1395.05	27	26	0.22	928.82	687.32	683.6	732.2	3.51
I155		0.2	360	**16**	20.55	18	**16**	26.31	**36**	5329.68	37	36.5	203.4	1625.59	1238.31	1200.3	612.83	52.48
I156			450	**23**	19.39	24	**23**	0.06	52	4573.61	52	52	0.1	2974.58	2647.04	2609.8	65.25	33.54
I157			240	**15**	39.71	18	**15**	0.08	29	6615.74	32	29	0.49	2014.4	1513.7	1501.89	586.06	7.4
I158		0.5	360	**21**	52.39	23	**21**	0.06	41	5914.37	43	39	37.99	2978.81	2389.03	2377	546.71	8.67
I159	0.5		450	**24**	52.17	**24**	**24**	0.06	50	7116.83	47	47	0.05	3523.19	3315.51	3278.4	70.71	30.06
I160			240	**19**	27.28	24	**19**	0.06	39	7198.19	40	33.1	83.43	3323.48	2553.61	2496.6	347.56	44.99
I161		0.8	360	**25**	14.38	**25**	**25**	0.06	51	7129.48	51	48	0.74	4860.43	4229.09	4178.2	134.55	31.84
I162			450	**28**	21.36	30	**28**	0.08	54	4852.52	57	54	3.37	5627.09	5118.14	5092.4	101.93	13.65
	Average results (ILP)			13.33	10.11	14.37	13.33	1.02	45.69	4252.58	43.65	41.17	17.71	–				
	Average results (all)			13.33	10.11	14.37	13.33	1.02	45.69	4252.58	43.74	41.28	17.05	3216	2691.37	2657.78	379.92	29.06

of TabuCol from $n = 15$ already, and the gap grows with the increase of n. Regarding TabuCol, the standard deviation and the computing times augment reasonably with the increase of n.

Table 13 presents the aggregated results for all the instances that can be solved by ILP. We compare DSAT, ILP, and PartialCol (for (P1) and (P2)) or TabuCol (for (P3)) according to the variation of each instance parameter (namely, u, d, and $\bar{p}$). We provide the improvement gaps (in percent) brought by ILP, PartialCol and TabuCol, when compared to the average objective-function values of DSAT (i.e., a positive value corresponds to an improvement).

The following observations can be made according to the increase of u, which corresponds to an augmentation of the instance size for (P1), but to a kind of reduction of the instance size for (P2). Indeed, for example, if 10% of the jobs are urgent (such jobs belong to (P1)), it means that 90% of them are not urgent (and such jobs belong to (P2)) and that 10% of the jobs will already be integrated in a given partial solution before solving (P2).

Table 8
Results for the instances with $n = 150$.

				ILP		DSAT	PartialCol		ILP		DSAT	PartialCol		DSAT	TabuCol			
ID	u	d	$\bar{p}$	k^*	Time	k^*	k^*	Time	k	Time	k	k	Time	H	H	$H^{(best)}$	Time	σ
I163			240	**5**	0	**5**	**5**	0.01	140	7.84	44	42	0	1721.37	1313.55	1293.9	611.17	11.5
I164		0.2	360	**8**	0.01	**8**	**8**	0.05	143	6.92	62	62	0.08	3056.13	2725.33	2703.27	468.48	19.23
I165			450	**11**	0.01	12	**11**	0.06	146	6.51	79	79	0.01	4660.83	4488.85	4406	446.7	36.55
I166			240	**5**	0.07	6	**5**	0.05	140	14.39	47	42	1.8	2996.47	2281.58	2254.4	687.57	11.92
I167		0.5	360	**7**	0.01	**7**	**7**	0.06	142	14.81	63	59	0.72	4675.25	3952.52	3915.7	694.03	19.57
I168	0.1		450	**11**	0.01	**11**	**11**	0	146	14.36	73	72	0.01	5622.5	5372.66	5294.6	394.43	37.46
I169			240	**8**	0.01	9	**8**	0.04	143	33.36	58	47.6	66.13	4898.06	3590.5	3528.98	666.86	48.07
I170		0.8	360	**9**	0.01	10	**9**	0	81	5267.87	73	65	25.09	6842.78	5644.78	5626.6	489.42	12.27
I171			450	**10**	0.01	11	**10**	0.06	–	–	87	88	0.01	8991.66	8935.61	8893.7	560.79	22.84
I172			240	**14**	10.41	**14**	**14**	0.05	–	–	44	43	0.92	1656.62	1370.6	1339.3	865.1	16.58
I173		0.2	360	**19**	21.29	20	**19**	0.06	–	–	63	63	0.15	3044.13	2785.19	2757.6	737.24	19.52
I174			450	**28**	11.24	**28**	**28**	0.04	–	–	74	74	0.05	4337.08	3929.25	3879.5	395.3	24.94
I175			240	12	120.35	14	12	0.06	–	–	45	40.9	12.7	2797.15	2220.49	2162	608.05	21.78
I176		0.5	360	19	37.35	22	19	0.06	–	–	63	61	45.56	4636.6	4144.48	4115.65	619.7	19.64
I177	0.3		450	**23**	35.21	24	**23**	0.06	–	–	75	74	40.66	6207.71	5793.32	5768	574.54	19.1
I178			240	**17**	116.48	20	**17**	0.06	–	–	58	49	82.01	4988.94	3778.17	3756.87	471.06	12.84
I179		0.8	360	**26**	16.37	27	**26**	0.06	–	–	70	65	3.08	6481.26	5660.6	5647.2	505.3	11.7
I180			450	**27**	22.92	28	**27**	0.06	–	–	80	77	0.66	7900.42	7313.81	7289.6	726.43	16.79
I181			240	**19**	465.07	**19**	**19**	0.05	–	–	41	39.3	90.72	1494.21	1160.61	1135.97	549.8	19.79
I182		0.2	360	29	360.82	30	29	0.06	–	–	61	60	1.18	3065.6	2552.32	2532.7	718.06	11.89
I183			450	**40**	437.78	**40**	**40**	0.06	–	–	83	83	0	5456.26	5016.66	4912.1	384.27	79.32
I184			240	21	742.27	24	20	9.97	–	–	45	41.1	157.92	2828.86	2237.55	2212	454.42	23.05
I185		0.5	360	31	859.12	33	30.1	61.81	–	–	63	65.2	4.41	4602.02	4637.17	4212.2	412.69	279.49
I186	0.5		450	40	2970.63	40	40	0.06	–	–	78	76	3.13	6639.86	6132.51	6082.6	412.31	26.69
I187			240	24	1042.21	30	23.7	1.35	–	–	57	48.1	169.47	5013.98	3701.83	3674.9	682.26	46.69
I188		0.8	360	30	5472.41	36	30.2	69.31	–	–	68	62	23.5	6241.75	5342.27	5074.2	659.55	154.4
I189			450	35	1238.61	39	35	0.26	–	–	78	75.8	86.99	7648.42	7198.09	7107.4	490.98	50.22
	Average results (ILP)			19.56	517.8	21	19.48	5.32	135.13	670.76	62.38	58.58	11.73	–				
	Average results (all)			19.56	517.8	21	19.48	5.32	135.13	670.76	64.15	61.26	30.26	4759.48	4195.57	4132.48	566.17	39.77

Table 9
Results for the instances with $n = 200$.

				ILP		DSAT	PartialCol		ILP		DSAT	PartialCol		DSAT	TabuCol			
ID	u	d	$\bar{p}$	k^*	Time	k^*	k^*	Time	k	Time	k	k	Time	H	H	$H^{(best)}$	Time	σ
I190			240	**6**	0.09	**6**	**6**	0.05	186	17.85	63	55	0.01	3039.96	1836.05	1809.3	611.56	11.32
I191		0.2	360	**10**	0.2	13	**10**	0.06	–	–	103	79	0.02	8109.42	3612.05	3552.95	852.99	30.42
I192			450	**9**	0.29	11	**9**	0.06	–	–	118	94	41.92	10570.43	4969.59	4921.5	598.89	27.13
I193			240	**6**	0.08	7	**6**	0.04	186	44.67	61	52	9.87	4109.69	2867.8	2846.4	655.18	12.04
I194		0.5	360	**10**	0.04	11	**10**	0.06	190	51.83	101	74.9	12.09	9614.95	5080.58	4999.3	489.63	31.1
I195	0.1		450	**14**	0.2	16	**14**	0	194	57.76	128	100	0.19	13541.18	8127.26	8071.7	687.57	38.66
I196			240	**9**	0.03	11	**9**	0.04	–	–	71	61	40.8	6049.75	4705.63	4687.22	535.27	11.62
I197		0.8	360	**10**	0.06	13	**10**	0.01	–	–	107	80.8	42.64	11130.35	6987.02	6901.4	587.75	44.69
I198			450	**16**	0.11	**16**	**16**	0	–	–	136	112	0.06	15211.62	11337.44	11311.1	566.26	21.2
I199			240	**17**	80.63	20	**17**	0.06	–	–	63	58	0.15	2743.58	2009.82	1984.48	528.33	11.42
I200		0.2	360	**26**	135.98	35	**26**	0.06	–	–	97	76	22.03	7194.04	3393.4	3358.6	601.25	21.69
I201			450	**27**	92.6	33	**27**	0.06	–	–	124	102.1	99.17	11337.73	6079.53	6005.3	528.96	66.81
I202			240	**19**	328.21	22	**19**	0.07	–	–	63	55.6	72.44	4231.57	3148.19	3096.3	644.68	36.3
I203		0.5	360	26	542.47	35	26	0.06	–	–	102	80.2	75.11	9690.13	5684.99	5626.4	712.56	55.76
I204	0.3		450	**28**	188.12	36	**28**	0.06	–	–	122	103	1.94	12618.48	8794.3	8717.3	492.53	38.39
I205			240	**20**	1136.05	27	**20**	0.54	–	–	71	64.5	109.42	6066.79	5088.54	4692.9	623.36	204.83
I206		0.8	360	**27**	158.08	37	**27**	0.08	–	–	110	87	99.6	11597.87	7804.1	7757.95	567.82	22.34
I207			450	**38**	308.22	44	**38**	0.06	–	–	132	108	8.36	14655.69	10759.43	10737.7	595.05	16.83
I208			240	**28**	4624.11	33	**28**	0.06	–	–	63	56	0.02	2842.4	1873.52	1854.5	562.34	11.35
I209		0.2	360	39	5501.48	54	39	0.06	–	–	99	78	1.11	7403.16	3515.72	3491.9	488.85	16.82
I210			450	**51**	3040.16	64	**51**	0.06	–	–	122	104	0.01	11243.78	6391.8	6356	647.52	36.98
I211			240	32	4085.59	34	28	0.07	–	–	66	58	2.06	4513.4	3364.06	3343.22	413.02	14.05
I212		0.5	360	46	3526.35	54	39	0.08	–	–	97	79	2.34	9051.51	5514.49	5484.5	500.29	17.36
I213	0.5		450	56	6599.12	66	54	0.06	–	–	122	103	2.41	12450.86	8839.17	8783.2	583.11	48.15
I214			240	34	5814.42	38	31.8	9.57	–	–	73	65.5	102.94	6231.49	5168.15	4975.13	644.48	235.08
I215		0.8	360	50	4630.98	58	42	0.27	–	–	102	82.6	164.3	10312.33	7227.24	7097.8	494.31	120.92
I216			450	57	1544.43	66	53	0.06	–	–	124	102	10.72	13529.58	9948.68	9919.5	476.87	18.99
	Average results (ILP)			26.33	1568.08	31.85	25.33	0.43	189	43.03	88.25	70.48	5.54	–				
	Average results (all)			26.33	1568.08	31.85	25.33	0.43	189	43.03	97.78	80.41	34.14	8855.25	5708.46	5643.84	581.13	45.27

Table 10
For (P1), percentage of optimal solutions found by ILP and average time needed to find its last solution, depending on u and n.

	Percentage of optimal solutions				Average computing time in seconds		
	n = 50	n = 100	n = 150	n = 200	n = 100	n = 150	n = 200
u = 0.1	100%	100%	100%	100%	0	0.02	0.12
u = 0.3	100%	100%	77.78%	88.89%	1.22	43.51	330.04
u = 0.5	100%	100%	22.22%	22.22%	29.12	1509.88	4374.07

Table 11
For TabuCol on (P3), evolution of σ and other robustness indicators depending on n.

n	Average σ	$\sigma = 0$	Average variation of H	Biggest variation of H
50	14.52	7.41%	1.14%	3.54%
100	29.06	0%	1.11%	6.09%
150	39.77	0%	1.00%	6.03%
200	45.27	0%	0.86%	4.55%

Table 12
Aggregated results in function of the instance size n.

Size	Problem (P1)							Problem (P2)							Problem (P3)									
n	ILP				DSAT	PartialCol		ILP				DSAT	PartialCol		ILP				DSAT	TabuCol				
	k^*	Time	SOL	OPT	k^*	k^*	Time	k	Time	SOL	OPT	k	k	Time	H	Time	SOL	OPT	H	H	$H^{(best)}$	Time	σ
15	2.56	0	27	27	2.59	2.56	0	7.74	0	27	27	8.15	7.74	0	505.6	1541	27	10	517.8	450.1	449.4	0.07	0.56
20	3.74	0	27	27	3.85	3.74	0	9.74	0	27	27	10.33	9.74	0	647.7	1946	27	5	666.4	558.4	554.7	0.47	3.53
25	4.11	0	27	27	4.26	4.11	0	11.74	0	27	27	12.3	11.74	0	806.5	3778	27	1	813.6	686	683.1	8.77	3.33
30	5	0	27	27	5.15	5	0	14.22	0	27	27	14.93	14.22	0	963.5	3824	26	0	1020	868.8	865.5	23.61	3.77
50	7.15	0	27	27	7.89	7.15	0	21.07	38.3	27	25	22.93	21.07	0	1545	6506	3	0	1591	1261	1244	139.1	14.52
100	13.33	10.11	27	27	14.37	13.33	1.02	45.69	4253	26	1	43.74	41.28	17.05	–	–	0	0	3216	2691	2658	379.9	29.06
150	19.56	517.8	27	18	21	19.48	5.32	135.1	670.8	6	0	64.15	61.26	30.26	–	–	0	0	4759	4196	4132	566.2	39.77
200	26.33	1568	27	19	31.85	25.33	0.43	189	43.03	4	0	97.78	80.41	34.14	–	–	0	0	8855	5708	5644	581.1	45.27

Table 13
Sensitivity analyses with respect to the instance parameters u, d and $\bar{p}$.

	k^*			k			H		
	DSAT	ILP	PartialCol	DSAT	ILP	PartialCol	DSAT	ILP	TabuCol
$u = 0.1$	4.46	6.54%	6.54%	28.23	−54.64%	9.04%	783.17	3.87%	17.10%
$u = 0.3$	11.64	10.98%	10.98%	18.36	4.93%	5.25%	782.65	5.00%	16.93%
$u = 0.5$	18.01	10.41%	12.66%	18.59	4.28%	5.72%	795.80	4.98%	13.85%
$d = 0.2$	10.22	10.19%	10.19%	19.93	−29.76%	2.98%	354.62	−7.56%	26.84%
$d = 0.5$	11.08	10.15%	12.02%	23.75	−40.69%	9.36%	700.87	2.09%	15.77%
$d = 0.8$	12.81	9.98%	11.53%	22.91	−1.01%	8.68%	1255.24	9.00%	13.26%
$\bar{p} = 240$	8.79	12.01%	13.19%	17.93	−41.68%	12.04%	648.38	16.64%	25.16%
$\bar{p} = 360$	11.75	11.47%	13.32%	22.32	−20.60%	7.36%	751.14	1.46%	15.10%
$\bar{p} = 450$	13.57	7.68%	8.29%	26.49	−14.83%	3.71%	979.12	−1.76%	9.81%

- *Performance of ILP.* The gaps of ILP are higher with $u \in \{0.3, 0.5\}$ than with $u = 0.1$. For (P1), the smaller gap of 6.54% (when compared to 10.98% and 10.41%) can be explained by the good performance of DSAT for the involved small, unconstrained problems. In contrast, ILP has a very low, negative gap (of more than 50%) for (P2) with $u = 0.1$. This can be explained by the fact that a smaller value of u results in a larger combinatorial complexity for solving (P2), as more non-urgent jobs have to be scheduled while satisfying the constraint of not augmenting a smaller number of urgent clusters. Moreover, from $n \geq 150$, the bad performance on (P2) with small values of u, along with the inability to find feasible solutions for higher values of u, highlights again the limitations of ILP with respect to the instance-size augmentation.
- *Performance of PartialCol/TabuCol.* First, such methods have always positive gaps (i.e., they are always able to improve the DSAT results). Moreover, such gaps are always higher than the ILP gaps (which shows again the superiority of such metaheuristics over ILP). For (P1) (resp. (P2)), the gaps of PartialCol augment (resp. somewhat decrease) with the augmentation of u. In other words, PartialCol reacts better than DSAT when augmenting the instance size for (P1) (i.e., when increasing u) and when decreasing the size of the partial solution given for (P2) (i.e., when decreasing u). Regarding (P3), we can observe that the performance of TabuCol decreases with the augmentation of u. This might be explained by the fact that more feasible moves can be tested by TabuCol with smaller values of u. Indeed, the solution space is less constrained when fewer jobs are restricted to the urgent clusters only. Furthermore, it is on average 19.75% faster to evaluate a solution with $u = 0.1$ than with $u = 0.5$ (i.e., TabuCol can explore more solutions with smaller values of u).

The following observations can be made according to the increase of d, which corresponds to augmenting the incompatibility percentage among jobs, which also corresponds to augmenting the number of constraints and thus reducing the size of the solution space.

- *Performance of ILP.* For (P1), ILP can roughly improve the results of DSAT by 10%, independently from the variation of d. Regarding (P2), we can again observe that ILP obtains poor results when the combinatorial complexity is larger (i.e., with $d \in \{0.2, 0.5\}$). The worst results are however obtained with $d = 0.5$ and not with $d = 0.2$. This is in line with the findings of the graph-coloring literature [38], in which the experiments have shown that among the random graphs, the ones with a density of $d = 0.5$ are the hardest to color. Regarding (P3), and as expected, ILP performs better with higher values of d. Indeed, positive gaps are obtained when $d \in \{0.5, 0.8\}$.

- *Performance of PartialCol/TabuCol.* For both (P1) and (P2), the improvements brought by PartialCol are the highest for the hardest configuration (i.e., for $d = 0.5$). The lowest gap (but still positive) is obtained for $d = 0.2$. This can be explained again by the fact that DSAT performs better when the problem is less constrained. Regarding (P3), a trend similar to the one observed with u can be made: when d is lower and more feasible moves can be tested, the improvements brought by TabuCol are higher. Also, evaluating a solution is 27.30% faster with $d = 0.2$ than with $d = 0.8$.

The following observations can be made according to the increase of $\bar{p}$, which corresponds to a reduction of the solution-space size. Indeed, if a job j has a higher processing time p_j, it can be combined with fewer jobs in the same cluster because of the capacity constraint (i.e., we cannot exceed 8 h per cluster). In other words, the combinatorial complexity decreases with the increase of $\bar{p}$.

- *Performance of ILP.* For (P1) and (P3), the gap of ILP decreases with the augmentation of $\bar{p}$. This shows again that DSAT is more efficient when the combinatorial complexity is small (i.e., when the number of job combinations is smaller). The reverse trend holds however for (P2). This might be explained by the lexicographic nature of the problem (i.e., the better a method is for a higher-level objective, the worse it is likely to be for a lower-level, conflicting objective).
- *Performance of PartialCol/TabuCol.* For each objective, PartialCol/TabuCol show higher improvements when the combinatorial complexity is larger (i.e., with smaller values of $\bar{p}$). As before for (P3), more feasible solutions can be tested with smaller values of $\bar{p}$. Evaluating a solution is 77.10% faster with $\bar{p} = 240$ than with $\bar{p} = 450$.

When comparing further DSAT with PartialCol/TabuCol, it is interesting to note that, considering all the instances (and not only the ILP-solved ones), the TabuCol positive gaps augment by roughly 7 points of percentage, on average. For example, the gap associated with $u = 0.1$ (resp. 0.3 and 0.5) is 17.10% (resp. 16.93% and 13.85%) when considering the ILP-solved instances, and it moves to 25.70% (resp. 22.63% and 21.80%) when considering all the instances. In other words, when considering the most difficult instances, the performance gap between TabuCol and DSAT increases significantly in favor of TabuCol. This positive trend is also observed for (P2) when comparing PartialCol and DSAT, where the PartialCol gaps augment roughly by 3%, on average. Unsurprisingly, the PartialCol gaps do not augment when considering (P1), as ILP finds a feasible solution for all the instances and thus the instance set does not differ.

These results lead to the following managerial insights.

- Local-search metaheuristics, such as PartialCol and TabuCol, are definitely recommended for instances with more than 30 jobs. This is in line with the job-scheduling literature.
- If computing time is an issue (typically, if the decision maker would like a solution within a few seconds because of an unexpected situation that has to be tackled immediately), DSAT could be employed for quickly generating sufficiently good solutions. Indeed, for each objective, its performance is roughly 15% away from the best-proposed metaheuristics.
- If the decision maker hesitates to determine whether a job j is urgent or not, s/he should consider j as urgent. Indeed, more efficient solutions are obtained for problem (P1) if the percentage u of urgent jobs is larger (i.e., PartialCol performs better for larger values of u).
- In this study, two jobs j and j' are incompatible if the setup between them is larger than $s^{min} = 5$ min. In such a case, vertices j and j' are connected with an edge in the associated incompatibility graph. If the production planner decides to increase s^{min} from 5 to 15 min, fewer pairs of jobs will be connected in the incompatibility graph (as the pairs of jobs for which the setup time is 5 or 15 min will not be connected). In other words, the density d of the graph will decrease, which has a positive impact on the performance of TabuCol for (P3). The decision maker would thus prefer setting $s^{min} = 15$ min if the number of (urgent) clusters does not increase when compared to the case with $s^{min} = 5$ min, as the returned solutions are likely to be better regarding (P3).
- Grouping together jobs having small setup times among them might be appealing to reduce the complexity of the problem (and the size of the incompatibility graph), as fewer jobs would have to be scheduled. For instance, one could merge jobs j_1, j_2 and j_3 into a single job j, which means that in practice, processing j corresponds to sequentially processing j_1, j_2 and j_3 (or the best permutation of these three jobs according to the overall setup time). However, performing such a preprocessing optimization step increases the value of $\bar{p}$ (i.e., the largest processing time of a job), which is likely to decrease the potential benefits of the proposed metaheuristics.

7. Conclusion

In this paper, we have studied a complex clustering and scheduling problem proposed by *DIXI polytool*, a micro-machining company based in Switzerland. Here, a cluster is an unsequenced group of jobs which must be processed one after the other before starting to produce the jobs of another cluster. Each cluster is subject to a capacity constraint (here, 8 h) and two incompatible jobs (with respect to setup times) cannot share the same cluster. To meet the company's goals, three objectives are considered in a lexicographic order. The two first objectives minimize the number of clusters (the first objective f_1 considers the subset of urgent jobs and the second objective f_2 considers all the jobs), whereas the third objective f_3 minimizes the worst-case scenario with respect to the setup times among clusters. Two well-known

models are used to capture the different features of the problem: the graph coloring problem and the traveling salesman problem. We decomposed the problem into three subproblems (one per objective). The proposed metaheuristic relies on two tabu-search algorithms, namely PartialCol (for f_1 and f_2) and TabuCol (for f_3). Note that a quick exact method for the traveling salesman problem is employed to compute the f_3-value of a solution.

We have considered realistic instances, generated with the help of the company to capture the different real situations they have to face in practice. A computing-time limit of 30 min is imposed to meet the requirements of the decision maker. PartialCol and TabuCol are favorably compared to an ILP model (solved using CPLEX) for which a time limit of 2 h per objective is allowed, and to a constructive heuristic (DSAT) that aims to reproduce how a decision maker could efficiently solve the problem in practice. The experiments show that our metaheuristic is efficient according to quality (i.e., value of the obtained solutions), speed (i.e., time to generate such solutions), and robustness indicators (e.g., deviation of the results if various runs are performed).

A possible extension of this work could be to increase the instance size such that the evaluation of f_3 cannot be performed with an exact method anymore. In this case, f_3 would have to be evaluated with a quick and efficient (meta)heuristic. Another avenue of research could be to consider a non-deterministic version of the problem, which would open the door to simulation.

Data availability

Data will be made available on request.

Acknowledgments

This study was funded in part by the company *DIXI polytool*, Switzerland (www.dixipolytool.ch). The authors would like to thank Marc Schuler, Simon Bournez and Benoit Van Schoors for their availability and advices. The computations were performed at the University of Geneva on the Baobab HPC cluster. Thanks are due to two anonymous reviewers for their constructive and valuable comments.

References

[1] J. Aghaei, N. Amjady, H.A. Shayanfar, Multi-objective electricity market clearing considering dynamic security by lexicographic optimization and augmented epsilon constraint method, Appl. Soft Comput. 11 (4) (2011) 3846–3858.
[2] H. Aissi, C. Bazgan, D. Vanderpooten, Min–max and min–max regret versions of combinatorial optimization problems: A survey, European J. Oper. Res. 197 (2) (2009) 427–438.
[3] C. Archetti, L. Bertazzi, M. Grazia Speranza, Reoptimizing the traveling salesman problem, Networks 42 (3) (2003) 154–159.
[4] A. Ben-Tal, L. El Ghaoui, A. Nemirovski, Robust Optimization, Princeton University Press, 2009.
[5] L.-P. Bigras, M. Gamache, G. Savard, The time-dependent traveling salesman problem and single machine scheduling problems with sequence dependent setup times, Discrete Optim. 5 (4) (2008) 685–699.
[6] Ü. Bilge, F. Kıraç, M. Kurtulan, P. Pekgün, A tabu search algorithm for parallel machine total tardiness problem, Comput. Oper. Res. 31 (3) (2004) 397–414.
[7] A. Billionnet, S. Elloumi, A. Lambert, Linear reformulations of integer quadratic programs, in: Communications in Computer and Information Science, Springer Berlin Heidelberg, 2008, pp. 43–51.
[8] D.C. Bissoli, N. Zufferey, A.R.S. Amaral, Lexicographic optimization-based clustering search metaheuristic for the multiobjective flexible job shop scheduling problem, Int. Trans. Oper. Res. 28 (5) (2021) 2733–2758.
[9] I. Blöchliger, N. Zufferey, A graph coloring heuristic using partial solutions and a reactive tabu scheme, Comput. Oper. Res. 35 (3) (2008) 960–975.
[10] R. Bollapragada, F. Della Croce, M. Ghirardi, Discrete-time, economic lot scheduling problem on multiple, non-identical production lines, European J. Oper. Res. 215 (1) (2011) 89–96.
[11] H. Bouchriha, M. Ouhimmou, S. D'Amours, Lot sizing problem on a paper machine under a cyclic production approach, Int. J. Prod. Econ. 105 (2) (2007) 318–328.
[12] W. Bożejko, A. Gnatowski, J. Pempera, M. Wodecki, Parallel tabu search for the cyclic job shop scheduling problem, Comput. Ind. Eng. 113 (2017) 512–524.
[13] D. Brélaz, New methods to color the vertices of a graph, Commun. ACM 22 (4) (1979) 251–256.
[14] A. Chassein, M. Goerigk, On the recoverable robust traveling salesman problem, Optim. Lett. 10 (7) (2015) 1479–1492.
[15] R.L. Daniels, P. Kouvelis, Robust scheduling to hedge against processing time uncertainty in single-stage production, Manage. Sci. 41 (2) (1995) 363–376.
[16] G. Dantzig, R. Fulkerson, S. Johnson, Solution of a large-scale traveling-salesman problem, J. Oper. Res. Soc. Am. 2 (4) (1954) 393–410.
[17] I. Dunning, J. Huchette, M. Lubin, JuMP: A modeling language for mathematical optimization, SIAM Rev. 59 (2) (2017) 295–320.
[18] A. Elmi, S. Topaloglu, Multi-degree cyclic flow shop robotic cell scheduling problem: Ant colony optimization, Comput. Oper. Res. 73 (2016) 67–83.
[19] L. Epstein, M.M. Halldórsson, A. Levin, H. Shachnai, Weighted sum coloring in batch scheduling of conflicting jobs, Algorithmica 55 (4) (2007) 643–665.
[20] A. Fleischhacker, A. Ninh, Y. Zhao, Positioning inventory in clinical trial supply chains, Prod. Oper. Manage. 24 (6) (2014) 991–1011.
[21] C. Fu, N. Zhu, S. Ma, R. Liu, A two-stage robust approach to integrated station location and rebalancing vehicle service design in bike-sharing systems, Eur. J. Oper. Res. 298 (2022) 915–938.
[22] O. Gallay, N. Zufferey, Metaheuristics for lexicographic optimization in industry, in: Proceedings of the 19th EU/ME Workshop on Metaheuristics for Industry, 2018.
[23] K. Giaro, M. Kubale, P. Obszarski, A graph coloring approach to scheduling of multiprocessor tasks on dedicated machines with availability constraints, Discrete Appl. Math. 157 (17) (2009) 3625–3630.
[24] F. Glover, M. Laguna, Tabu search, in: Handbook of Combinatorial Optimization, Springer US, 1998, pp. 2093–2229.

[25] M. Goerigk, J. Kurtz, M. Poss, Min-max-min robustness for combinatorial problems with discrete budgeted uncertainty, Discrete Appl. Math. 285 (2020) 707–725.
[26] A. Grigoriev, V.J. Kreuzen, T. Oosterwijk, Cyclic lot-sizing problems with sequencing costs, J. Sched. 24 (2) (2020) 123–135.
[27] S. Henn, V. Schmid, Metaheuristics for order batching and sequencing in manual order picking systems, Comput. Ind. Eng. 66 (2) (2013) 338–351.
[28] A. Hertz, D. de Werra, Using tabu search techniques for graph coloring, Computing 39 (4) (1987) 345–351.
[29] A. Jalilvand-Nejad, P. Fattahi, A mathematical model and genetic algorithm to cyclic flexible job shop scheduling problem, J. Intell. Manuf. 26 (6) (2013) 1085–1098.
[30] S. Khosravani, M. Jalali, A. Khajepour, A. Kasaiezadeh, S.-K. Chen, B. Litkouhi, Application of lexicographic optimization method to integrated vehicle control systems, IEEE Trans. Ind. Electron. 65 (12) (2018) 9677–9686.
[31] P. Kouvelis, R.L. Daniels, G. Vairaktarakis, Robust scheduling of a two-machine flow shop with uncertain processing times, IIE Trans. 32 (5) (2000) 421–432.
[32] H. Krim, N. Zufferey, J.-Y. Potvin, R. Benmansour, D. Duvivier, Tabu search for a parallel-machine scheduling problem with periodic maintenance, job rejection and weighted sum of completion times, J. Sched. 25 (2022) 89–105.
[33] W. Kubiak, Solution of the Liu–Layland problem via bottleneck just-in-time sequencing, J. Sched. 8 (4) (2005) 295–302.
[34] M.E. Kurz, R.G. Askin, Scheduling flexible flow lines with sequence-dependent setup times, European J. Oper. Res. 159 (1) (2004) 66–82.
[35] G. Laporte, A concise guide to the traveling salesman problem, J. Oper. Res. Soc. 61 (1) (2010) 35–40.
[36] C.-J. Liao, H.-C. Juan, An ant colony optimization for single-machine tardiness scheduling with sequence-dependent setups, Comput. Oper. Res. 34 (7) (2007) 1899–1909.
[37] T. Loukil, J. Teghem, D. Tuyttens, Solving multiobjective production scheduling problems using metaheuristics, Eur. J. Oper. Res. 161 (2005) 42–61.
[38] E. Malaguti, P. Toth, A survey on vertex coloring problems, Int. Trans. Oper. Res. 17 (1) (2010) 1–34.
[39] B. Menéndez, M. Bustillo, E.G. Pardo, A. Duarte, General variable neighborhood search for the order batching and sequencing problem, European J. Oper. Res. 263 (1) (2017) 82–93.
[40] U. Pferschy, R. Staněk, Generating subtour elimination constraints for the TSP from pure integer solutions, CEJOR Cent. Eur. J. Oper. Res. 25 (1) (2016) 231–260.
[41] M.L. Pinedo, Scheduling: Theory, Algorithms, and Systems (Fifth Edition), Springer International Publishing, 2016.
[42] C.N. Potts, M.Y. Kovalyov, Scheduling with batching: A review, European J. Oper. Res. 120 (2) (2000) 228–249.
[43] J. Respen, N. Zufferey, E. Amaldi, Metaheuristics for a job scheduling problem with smoothing costs relevant for the car industry, Networks 67 (3) (2016) 246–261.
[44] J. Respen, N. Zufferey, P. Wieser, Three-level inventory deployment for a luxury watch company facing various perturbations, J. Oper. Res. Soc. 68 (10) (2017) 1195–1210.
[45] T. Sawik, A lexicographic approach to bi-objective scheduling of single-period orders in make-to-order manufacturing, European J. Oper. Res. 180 (3) (2007) 1060–1075.
[46] C. Solnon, V.D. Cung, A. Nguyen, C. Artigues, The car sequencing problem: Overview of state-of-the-art methods and industrial case-study of the ROADEF'2005 challenge problem, European J. Oper. Res. 191 (3) (2008) 912–927.
[47] S. Thevenin, N. Zufferey, J.-Y. Potvin, Makespan minimisation for a parallel machine scheduling problem with preemption and job incompatibility, Int. J. Prod. Res. 55 (6) (2016) 1588–1606.
[48] S. Thevenin, N. Zufferey, J.-Y. Potvin, Graph multi-coloring for a job scheduling application, Discrete Appl. Math. 234 (2018) 218–235.
[49] N. Trautmann, C. Schwindt, A cyclic approach to large-scale short-term planning in chemical batch production, J. Sched. 12 (6) (2009) 595–606.
[50] M.-S. Vié, N. Zufferey, R. Leus, Aircraft landing planning under uncertainties, J. Sched. 25 (2022) 203–228.
[51] S. Wilson, N. Ali, Product wheels to achieve mix flexibility in process industries, J. Manuf. Technol. Manag. 25 (3) (2014) 371–392.
[52] F. Yang, N. Wu, Y. Qiao, R. Su, Polynomial approach to optimal one-wafer cyclic scheduling of treelike hybrid multi-cluster tools via Petri nets, IEEE/CAA J. Autom. Sin. 5 (1) (2018) 270–280.
[53] C. Yugma, S. Dauzère-Pérès, C. Artigues, A. Derreumaux, O. Sibille, A batching and scheduling algorithm for the diffusion area in semiconductor manufacturing, Int. J. Prod. Res. 50 (8) (2012) 2118–2132.