Computers & Industrial Engineering 151 (2021) 106962

Contents lists available at ScienceDirect

Computers & Industrial Engineering

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

Customer order scheduling problem to minimize makespan with sequence-dependent setup times

Bruno de Athayde Prata [a,*], Carlos Diego Rodrigues [b], Jose Manuel Framinan [c]

[a] Department of Industrial Engineering, Federal University of Ceara, Ceará, Brazil
[b] Department of Statistics and Applied Mathematics, Federal University of Ceara, Ceará, Brazil
[c] Department of Industrial Organization and Business Management I, University of Sevile, Sevile, Spain

ARTICLE INFO

Keywords:
Production sequencing
Assembly scheduling problems
Combinatorial optimization
Matheuristics

ABSTRACT

In this paper, we study a variant of the customer order scheduling problem when sequence-dependent setup times cannot be ignored. The performance measure adopted is the makespan minimization. The existence of sequence-dependent setup times makes this problem to be NP-hard. Furthermore, the solution encoding usually employed for other variants of the customer order scheduling problem does not guarantee finding optimal solutions. For this problem, we present some properties and develop two Mixed Integer Linear Programming (MILP) formulations to analyze the structure of the solutions. Using these properties and models, we propose two matheuristics based on fixing some integer decision variables in the MILP models, denoted as Fixed Variable List Algorithm (FVLA) and Clustering Sequence Algorithm (CSA), respectively. The computational experiments carried out prove the ability of these matheuristics to find high-quality solutions in reasonable CPU time. More specifically, the FVLA matheuristic stands out as the most efficient for the problem.

1. Introduction

Nowadays, the high exigency level of consumers coupled with the fierce competition among companies is changing the existing production paradigms in a continuous search for efficiency, thus allocating to different facilities or production lines the manufacturing of components of customized products, which are subsequently assembled in another facility. As a result, in the last few years, researchers in the production scheduling area are paying an increasing attention to assembly scheduling problems (see e.g. Framinan, Perez-Gonzalez, & Fernandez-Viagas, 2019 for a survey of the state-of-the-art in the topic, together with a unified notation for this class of problems).

Among the assembly problems, the so-called *customer order scheduling* problem arises when there are n customer orders – each one consisting of different products – to be processed in m dedicated parallel machines. In this problem, the machines are capable to perform only a single type of operation for a specific product but, since the products belong to a single customer, the order is completed only when the manufacturing of all the corresponding products has been completed. This is tantamount to considering a final (zero processing time) assembly operation for the completed orders and it is thus denoted $DPm \rightarrow 0$ according to the notation in Framinan et al. (2019). This scheduling environment arises in several real-world applications, such as the paper industry, the pharmaceutical industry, as well as in assembly operations (Leung, Li, & Pinedo, 2005).

In the existing literature on the customer order scheduling problem, the setup times required to prepare the machinery are considered as a part of the processing times. However, this may not be a realistic assumption in many cases, as most automated and semi-automated machinery presents some degree of flexibility in processing different products, for which they usually require some set-up operations. Thus, explicitly considering setup times is an important topic in the scheduling literature (Abreu, Cunha, Prata, & Framinan, 2020; Allahverdi, Ng, Cheng, & Kovalyov, 2008; Moccellin, Nagano, Pitombeira Neto, & Prata, 2018). However, this has not been the case in the customer order scheduling problem where, despite its theoretical and practical importance, according to the recent literature survey (Framinan et al., 2019), the problem with sequence-dependent setup times – denoted as $DPm \rightarrow 0|ST_{sd}|C_{max}$ – has not been addressed so far. Note that this problem is substantially different than the same problem without setups, which is known to be polynomial (see e g. Wagneur & Sriskandarajah, 1993), and therefore requires an specific analysis.

In this paper we address the customer order scheduling problem with sequence-dependent setup times. We first prove that the problem is NP-hard, and present two mixed-integer linear problem (MILP)

* Corresponding author.
E-mail addresses: baprata@ufc.br (B.d.A. Prata), diego@lia.ufc.br (C.D. Rodrigues), framina@us.es (J.M. Framinan).

https://doi.org/10.1016/j.cie.2020.106962
Received 14 July 2020; Received in revised form 23 September 2020; Accepted 2 November 2020
Available online 5 November 2020

models to solve it optimally. Furthermore, we analyze some problem properties and show that, in contrast to most related assembly problems, maintaining the same order of jobs for all machines (i.e. a permutation encoding of the solutions) does not guarantee reaching the optimal solution. In view of these problem features, we develop two matheuristics, and report an extensive computational experimentation to show their efficiency, also by comparing them with the state-of-the-art methods from related problems.

The remainder of this paper is organized as follows: in Section 2 the related literature is reviewed; in Section 3, the scheduling problem treated in this paper is stated together with some properties. In Section 4, the MILP models are presented, while in Section 5, the proposed algorithms are described; in Section 6, we discuss some results from computational experiments; finally, in Section 7 we draw some conclusions and suggestions for future works.

2. Literature review

As mentioned in Section 1, to the best of our knowledge, the order scheduling problem considering sequence-dependent setup times for makespan minimization has not been previously reported in the revised literature. Therefore, we review the state-of-the-art looking for related approaches, starting with the problem without setup considerations. Julien and Magazine (1990) consider a flexible manufacturing environment in which customer requirements for each of the possible product types are known in advance. The customer order scheduling was formally described by Wagneur and Sriskandarajah (1993), as a variant of the open shop scheduling problem with jobs overlaps. They analyze the complexity of several objective functions, and show that, for the objective of the makespan minimization, it is a trivial problem.

Leung et al. (2005) consider the customer order scheduling with total weighted completion time minimization. They presented an important result: there exists an optimal solution consisting of a fixed permutation in all dedicated parallel machines, an assumption that it is either implicitly or explicitly made in most contributions on different variants of the problem. Furthermore, two constructive heuristics were proposed. For the same objective function, Shi, Wang, Liu, and Shi (2017) propose a quadratic formulation, which can be converted into a mixed-integer linear programming model. Also, a hybrid nested partitions algorithm is developed to solve large-sized instances.

Lee (2013) addresses the order scheduling with total tardiness minimization. In this case it is also shown that an optimal solution can be found with the same sequence of the orders on all machines. Four constructive heuristics, as well as a branch-and-bound algorithm, are proposed. This problem is also studied by Framinan and Perez-Gonzalez (2018), which propose a mixed-integer linear programming formulation, a new constructive heuristic, as well as two matheuristics. Xu et al. (2016) introduce a customer order scheduling environment with a position-based learning effect, in which the objective function is the total tardiness minimization. A lower bound, some dominance relations, a branch-and-bound algorithm, as well as two metaheuristics (simulated annealing and particle swarm optimization) are presented.

Xu, Ma, Zhou, and Zhao (2015) address the customer order scheduling problem with unrelated parallel machines for the total completion time minimization. A lower bound and three constructive heuristics are developed for this problem. Framinan and Perez-Gonzalez (2017) addressed the customer order scheduling environment for the total completion time objective. A constructive heuristic with a look-ahead mechanism is proposed. Furthermore, a greedy search algorithm is also presented. For this same variant, Riahi, Newton, Polash, and Sattar (2019) present eight dispatching rules, which constitute the initial solutions of a constructive heuristic that evaluates several partial schedules. Also, a perturbative search metaheuristic is proposed. Lin et al. (2017) study a two-agent multi-facility customer order scheduling with ready times for the total completion time minimization. A branch-and-bound algorithm with dominance rules is proposed. Besides, two evolutionary algorithms (particle swarm optimization and opposite-based particle swarm optimization) are developed.

Table 1
Processing times for order scheduling example.

$i\backslash k$	O_1	O_2	O_3
M_1	3	4	2
M_2	4	1	2

Table 2
Setup times for the first machine of the order scheduling example.

$j\backslash k$	O_1	O_2	O_3
J_1	–	3	4
J_2	5	–	1
J_3	6	2	–

Table 3
Setup times for the second machine of the order scheduling example.

$j\backslash k$	O_1	O_2	O_3
J_1	–	5	8
J_2	2	–	10
J_3	1	3	–

Kung et al. (2018) address a customer order scheduling environment with unequal order ready times for total completion time minimization. Two lower bounds are developed, and some problem properties are explored. Eight metaheuristics are proposed to solve the problem: four simulated annealing (SA) algorithms, and four genetic algorithms (GAs). Also, a Branch-and-Bound (B&B) algorithm is presented. The computational results point out that the SA-based algorithms outperform the GAs. Lin et al. (2019) address a customer order scheduling environment with release dates to minimize the weighted number of tardy orders. Some dominance properties are shown and a lower bound is presented together with a B&B algorithm. Regarding approximate solution procedures, these authors present four standard bee colony algorithms, followed by four hybrid bee colony algorithms for finding near-optimal solutions within acceptable computational times. Wu et al. (2019) introduce a customer order scheduling variant with ready times. Several dominance rules are explored, and two lower bounds are developed. These elements are considered to develop a B&B algorithm. Furthermore, the authors adapt five existing constructive heuristics to the problem under study and propose an Iterated Greedy (IG) metaheuristic.

As it can be seen, the best of our knowledge, the explicit consideration of setup times in the customer order scheduling environment has not been considered so far in the revised literature. In the next section, we formalize the $DPm \rightarrow 0|ST_{sd}|C_{max}$ problem and present some properties.

3. Problem statement and properties

Let n be the number of customer orders to be produced in a set of m dedicated parallel machines. For each machine i, a permutation Π_i represents a feasible sequence, in which a given order k can be processed in position j of this permutation. Each order k presents a processing time p_{ik} on machine i, and requires a setup time s_{ijk} if processed after order j, which is sequence-dependent. Given these definitions, the problem under study is to find the best sequence for each available machine aiming to minimize the maximal completion time C_{max} for all the orders, or makespan.

Consider an illustrative example with three orders and two machines. Processing and setup times are presented in Tables 1, 2, and 3, respectively. The global optimal solution for this instance is obtained by the sequences $\Pi_1 = \{1, 2, 3\}$ in machine M_1 and $\Pi_2 = \{3, 2, 1\}$ in machine M_2, with a makespan of 16 time units, as illustrated in Fig. 1.

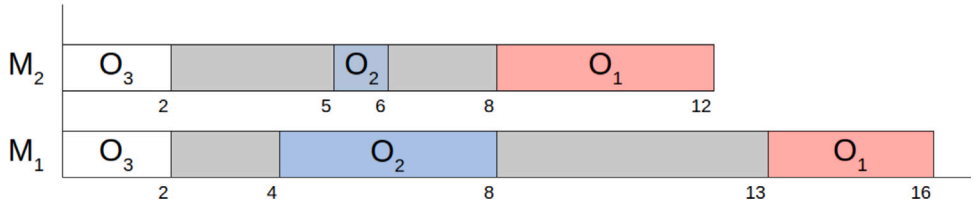

Fig. 1. Gantt chart for the presented solution.

Table 4
Setup times for the first machine of the FVLA example.

$j\backslash k$	O_1	O_2	O_3	O_4
J_1	–	2	4	5
J_2	1	–	2	4
J_3	6	3	–	9
J_3	2	5	1	–

In the next theorem, we show that the optimization problem of customer order scheduling with sequence-dependent setup times is NP-hard.

Theorem 1. *The customer order scheduling with sequence-dependent setup times and makespan objective is NP-Complete.*

Proof. The $DPm \rightarrow 0|ST_{sd}|C_{max}$ is a special case of the $1|ST_{sd}|C_{max}$ when the number of machines is equal to one and all the processing times are null. Since the $1|ST_{sd}|C_{max}$ problem is NP-complete, then it follows that the latter is NP-Complete (Pinedo, 2018). □

Another interesting property of the problem under consideration refers to the structure of the optimal solutions. In the following, we denote as *permutation* to one solution of the problem where the sequence in which the orders are processed *in all machines* is given (i.e. all orders are processed in the same sequence for all machine). The space of solutions of the problem consisting of all possible permutations is denoted as *permutation encoding*. Note that, in principle, the permutation encoding only represents a subset of all possible solutions for a problem instance and, therefore, it does not necessarily contains its optimal solutions. However, Leung et al. (2005) showed that, for the customer order scheduling problem without setups ($DPm \rightarrow 0 \parallel \Sigma C_j$), there exists at least one optimal solution where the production sequence of the orders is the same on all the machines. This is also the case for other objective functions within the customer order scheduling problem, such as the weighted tardiness, and it is a property that greatly simplifies the design of solution procedures, as there are many methods that use naturally the permutation encoding and therefore can be adapted to the problem in an straightforward manner. However, this property does not hold for the problem under study, as shown in the following remark.

Remark 1. The space of solutions of a $DPm \rightarrow 0|ST_{sd}|C_{max}$ problem instance consisting of sequencing the jobs in the same order in all machines (permutation encoding) does not necessarily contains its optimal solutions.

To prove the remark, we present a counterexample. Taking into consideration the test instance presented in Tables 1, 2, and 3, we enumerate all the permutations with their respective makespan, as presented below:

- $\Pi_A = \{1,2,3\}$, $C_{max} = 22$.
- $\Pi_B = \{1,3,2\}$, $C_{max} = 22$.
- $\Pi_C = \{2,1,3\}$, $C_{max} = 18$.
- $\Pi_D = \{2,3,1\}$, $C_{max} = 23$.
- $\Pi_E = \{3,1,2\}$, $C_{max} = 18$.
- $\Pi_F = \{3,2,1\}$, $C_{max} = 16$.

Since the optimal solution for this instance is obtained by the sequences $\Pi_1 = \{1, 2, 3\}$ in the machine M_1 and $\Pi_2 = \{3, 1, 2\}$ in the machine M_2, with a makespan of 13 time units, as illustrated in Fig. 1, we prove by reduction to the absurd that considering only a permutation encoding does not necessarily yield the optimal solution. Thus, approaches taking into consideration different permutations on all the machines could potentially lead to better results. However, given the higher complexity of non using permutation encoding, this aspect is further analyzed in Section 6.

Finally, one property can be derived to obtain a lower bound for the problem, which can be stated as follows.

Proposition 1. *A lower bound for the problem* $DPm \rightarrow 0|ST_{sd}|C_{max}$ *is given by*

$$LB = \max_{i\in\{1,\ldots,m\}} \left\{ \sum_{j=1}^{n} \left(p_{ij} + \min_{k\in\{1,\ldots,n\}} \{s_{ijk}\} \right) - \max_{j,k\in\{1,\ldots,n\}} \{s_{ijk}\} \right\} \quad (1)$$

Proof. Let C be the (optimal) makespan for the problem $DPm \rightarrow 0||C_{max}$. C is given by the following expression (Wagneur & Sriskandarajah, 1993):

$$C = \max_{i\in\{1,\ldots,m\}} \sum_{j=1}^{n} p_{ij} \quad (2)$$

For our problem with sequence-dependent setup times, for each order k, the lowest contribution to the makespan is given by the smallest setup time for each position j and for each machine i. Since we are assuming that all machines are prepared in the first position, we exclude the maximal setup time for each position and for each machine. Therefore, the estimate expressed by Eq. (1) is the lowest possible value for a solution of the problem under study. □

4. Mixed-integer linear programming models for the problem

In view of the properties presented in Section 3, different solution procedures can be devised. On the one hand, we have proved that the space of solutions consisting on having the same sequence of jobs across all the machines (permutation encoding) does not necessarily yield the optimal solution. However, the permutation encoding is the most-widely employed in approximate solution procedures for related problems (including the customer order scheduling without setups), so it might be that considering only permutations has advantages in terms of finding good solutions with less CPU time requirements. With this idea, we develop two different MILP models to assess the quality of the solutions that the different encoding mechanisms can achieve.

In the first model – denoted in the following as M1 – we extend the formulation by Framinan and Perez-Gonzalez (2018), which uses positional decision variables. Let x_{ijk} be a binary decision variable equal to 1 if order k is scheduled in position j of machine i. By taking into account the sequence-dependent setup times s_{ilk}, the model determines the setup time of position j on each machine i D_{ij}, and consequently the maximal completion time of the orders C_{max}. Hereafter, the notation used for the M1 model is presented.

Indices and sets

k: index for orders: $k \in K := \{1, 2, \ldots, n\}$.

j: index for positions: $j \in J := \{1, 2, \ldots, n\}$.
i: index for machines: $i \in I := \{1, 2, \ldots, m\}$.

Parameters

p_{ik}: processing time of the order k in machine i.
s_{ilk}: setup time of the order k in position l of machine i.

Decision variables

C_{max}: completion time of project (makespan).
D_{ij}: setup time of position j in machine i.

$$x_{ijk} = \begin{cases} 1, & \text{if the order } k \text{ is processed in the position } j \text{ of machine } i. \\ 0, & \text{otherwise} \end{cases}$$

The resulting MILP model is as follows.

minimize

$$C_{max} \tag{3}$$

subject to

$$\sum_{k=1}^{n} x_{ijk} = 1, \qquad \forall i \in I, j \in J \tag{4}$$

$$\sum_{j=1}^{n} x_{ijk} = 1, \qquad \forall i \in I, k \in K \tag{5}$$

$$D_{ij} \geq (x_{i,j-1,k} + x_{ijl} - 1) s_{ijk}, \qquad \forall i \in I, j > 1, \forall k, l \in K \tag{6}$$

$$C_{max} \geq \sum_{r=1}^{j} D_{ir} + \sum_{k=1}^{n} \sum_{r=1}^{j} p_{ik} x_{ikr}, \qquad \forall i \in I, j \in J \tag{7}$$

$$C_{max} \geq 0, \tag{8}$$

$$D_{ij} \geq 0, \qquad \forall i \in I, j \in J \tag{9}$$

$$x_{ijk} \in \{0, 1\}, \qquad \forall i \in I, j \in J, k \in K \tag{10}$$

The objective function (3) is the maximum completion time (makespan) minimization. Constraints (4) ensure that an order is scheduled only in one position of machine i. Set of constraints (5) enforces that a position is occupied only by a job from the order k in machine i. Set of constraints (6) calculates the setup time of the machine i in position j. We assume that all the machines are prepared in the first position (the setups times are only computed if j is greater than 1). Set of constraints (7) compute the makespan, considering processing and setup times. Finally, constraint sets (8), (9), and (10) establish the domain of the decision variables. The proposed model has $m \cdot n^2$ binary decision variables, $m \cdot n + 1$ continuous decision variables and $m \cdot n(4 - m)$ integer linear constraints.

Next, we present an alternative formulation for the problem under consideration in which the number of distinct permutations allowed for the machines is a parameter defined by the user. In this second MILP, the orders are allocated in positions of a given permutation. Considering a given number of specified permutations, the model determines the best position for an order in a given permutation, and, consequently, the allocation of permutations in the available machines. Taking into account the notation for the first mathematical formulation, we present the additional notation for this model (M2 in the following).

Index

w: index for the number of permutations in the machines: $w \in \mathcal{P} := \{p_1, \ldots, p_{max}\}$.

Decision variables

$$x_{jkw} = \begin{cases} 1, & \text{if the order } k \text{ is processed in the position } j \text{ of permutation } w. \\ 0, & \text{otherwise} \end{cases}$$

$$z_{iw} = \begin{cases} 1, & \text{if the order machine } i \text{ uses permutation } w. \\ 0, & \text{otherwise} \end{cases}$$

$$\min C_{max} \tag{11}$$

subject to

$$\sum_{k=1}^{n} x_{jkw} = 1, \quad \forall j \in J, \forall w \in \mathcal{P} \tag{12}$$

$$\sum_{j=1}^{n} x_{jkw} = 1, \quad \forall k \in K, \forall w \in \mathcal{P} \tag{13}$$

$$\sum_{p \in \mathcal{P}} z_{iw} = 1, \quad \forall i \in I, \forall w \in \mathcal{P} \tag{14}$$

$$s_{ijk}(x_{j-1,kw} + x_{jlw} + z_{iw} - 2) \leq D_{ij}, \quad \forall i \in I, \quad \forall j > 1, \quad \forall k \in K, \quad \forall l \in J, \forall w \in \mathcal{P} \tag{15}$$

$$\sum_{r=1}^{j} D_{ir} + \sum_{k=1}^{n} \sum_{r=1}^{j} p_{ik} \cdot (x_{rkw} + z_{iw} - 1) \leq C_{max}, \quad \forall w \in \mathcal{P}, \quad \forall i \in I, \quad \forall j \in J \tag{16}$$

$$C_{max} \geq 0 \tag{17}$$

$$D_{ij} \geq 0, \quad \forall \quad \forall i \in I, \forall j \in J \tag{18}$$

$$x_{jkw} \in \{0, 1\}, \quad \forall \quad \forall j \in J, \forall k \in K, \forall w \in \mathcal{P} \tag{19}$$

$$z_{iw} \in \{0, 1\}, \quad \forall i \in I, \forall w \in \mathcal{P} \tag{20}$$

Eq. (11) is the objective (makespan minimization). Set of constraints (12) ensures that only one order is scheduled in the position j of permutation w. Set of constraints (13) imposes that each order k is scheduled in a single position of the permutation w. Set of constraints (14) enforces each machine to select a permutation. Set of constraints (15) calculates the setup time of machine i of order k before the order l. Set of constraints (16) calculates the completion time of a given order. Finally, constraint sets (17), (18), (19) and (20) determine domain of the decision variables. The proposed model has $p_{max} \cdot n^2 + m$ binary decision variables, $mn + 1$ continuous decision variables, and $p_{max} \cdot (2n + 3mn) + 3m$ integer linear constraints.

In this model, if $p_{max} = 1$, the same permutation is used for all machines. On the other hand, if $p_{max} = m$, each machine can present a different permutation. In this manner, the effect of the number of permutations in the quality of the solutions obtained can be assessed, as well as give some clues for the design of approximate procedures for the problem. The results of the comparison of the models are presented in Section 6.3.1, while the approximate procedures designed are presented in the next section.

5. Approximate algorithms

In this section, we present two solution approaches for the problem under study. These approaches can be classified as matheuristics since they incorporate characteristics of heuristic algorithms and the MILP models presented in the previous section. The first one is based on the concept of the well-known Greedy Randomized Adaptive Search Procedures (GRASP) metaheuristic (Feo & Resende, 1995), building a list of decision variables in the M1 model to be fixed, taking into account specific characteristics of the problem instance. The second matheuristic clusters the machines considering the similarities in setup times distributions, reducing the number of decision variables to be solved by a MILP model. These algorithms are described in the next subsections.

5.1. Fixed variable list algorithm

After the computational experiments with the MILP models proposed in Section 4 –which are presented in detail in Section 6.3.1–,

we have observed that the inclusion of sequence-dependent setup times makes the problem substantially more difficult than its classical counterpart without explicitly considering setup times. More specifically, we have checked that the usage of the permutation encoding greatly reduces the quality of the solutions so, in order to develop efficient procedures, different processing sequences have to be considered for each machine. We also observed that most positions associated with high setups are hardly present in high-quality solutions (i.e. typically orders are placed in positions incurring with small set-ups). Thereby, our idea is to develop a procedure based on the MILP models that sets to zero those decision variables that probably would not be selected in optimal or near-optimal solutions, thus hopefully reducing the solution space for the MILP models without removing good solutions from this solution space. Note however that such procedure obtains approximate solutions for the problem, as it is not possible to establish the optimality of the so-obtained solutions. Therefore, the proposed procedure is based on the GRASP metaheuristic introduced by Feo and Resende (1989), originally for the well-known set covering problem. This semi-greedy algorithm (Hart & Shogan, 1987) is composed of two main phases: a construction phase and an improvement phase. In the construction phase, a feasible solution is built using a constructive procedure that combines greediness and randomness in order to avoid local optima and migrate to other regions of the search space. In the improvement phase, the solution generated in the construction phase is improved using a local search procedure. In the GRASP, the two phases are repeated iteratively until a stopping criterion is met, even if we do not use this feature in our proposal.

The construction phase in GRASP is based on the concept of a Restricted Candidate List (RCL), which can be limited by the number of elements (cardinality-based) or by their quality (value-based) (Resende & Ribeiro, 2016). In the cardinality-based RCL, the number of elements is fixed without considering the quality of elements. In the value-based RCL, the number of elements is variable, and the number of elements inserted in the list is controlled by a parameter α. The solutions in the RCL are randomly selected in each step of the constructive procedure, combining greediness and randomness. As advantages of GRASP we can emphasize that it shows a small dependency on its parameters; it has an extremely simple computational implementation; due to the concept of RCL, GRASP facilitates the construction of feasible solutions; and the use of a randomized constructive heuristic in the first phase, together with the local search in the second phase, allows combining diversification and intensification.

In our proposal, we incorporate the ideas from GRASP into an innovative matheuristic. Initially, we construct a list of decision variables taken from the MILP model that could be set to zero as they probably would not be selected in an optimal or near-optimal solution. We denote this set of variables as Fixed Variable List (FVL), similarly to the RCL concept. By setting these variables to zero, a reduced (i.e. faster to be solved) MILP model can be obtained and solved using a solver. Note that this step is similar to the local search procedure in the second phase of GRASP. More specifically, in our algorithm, for each decision variable x_{ijk} in M1, we calculate an indicator δ_{ijk} that evaluates the impact of setup time, as expressed in Eq. (21):

$$\delta_{ijk} = \frac{s_{ijk} - s^{ij}_{min}}{s^{ij}_{max} - s^{ij}_{min}} \tag{21}$$

where $s^{ij}_{min} = \min_k s_{ijk}$, and $s^{ij}_{max} = \max_k s_{ijk}$.

As it can be seen, δ_{ijk} represents the *normalized* setup time if order k is processed in position j in machine i. Therefore, if $\delta_{ijk} = 0$, then j is the position for order k in machine i where the set-up time is minimum. On the other hand, if $\delta_{ijk} = 1$, this setup time is maximum. Intermediate values could present a trade-off between the computational effort and the quality of generated solutions.

The parameter α can be used to control the number of variables to be inserted in the FVL. More specifically, decision variables with a δ_{ijk} value higher than α are set to zero (i.e. the corresponding position of the order is not considered in the reduced MILP model to be solved). If we set an α value close to zero, the reduction is quite aggressive, resulting in a MILP model with a drastically reduced number of decision variables. This reduction may have two effects: the first one is a drastic reduction of the computational effort required to solve a given instance, and the second one is the unfeasibility to solve a given instance within a specified time limit. More specifically, commercial solvers usually apply several heuristics in the pre-solve process to improve the efficiency of the solution process. For example, if several decision variables are fixed as zero, the solver can fail to identify if the problem presents given properties, and the pre-solve procedure is not so efficient, increasing the computational time required to solve the instance. In contrast, if we adopt an α value close to 1, the reduced subproblem is quite similar to the original problem, and the complexity of the model is not substantially reduced. Intermediate values for the parameter α may result in a good trade-off between these aspects.

Thus, we define α as a parameter that controls the greediness of the matheuristic. We present here the concept of Fixed Variable List (FVL), which can be defined as expressed in Eq. (22):

$$FVL \leftarrow \{x_{ijk} | \delta_{ijk} \geq \alpha\} \tag{22}$$

In this way, the decision variables inserted in FVL are fixed as zero. One can easily change this criterion to set decision variables in other values, taking into consideration the problem domain. Taking these aspects into account, the proposed algorithm consists of the following steps. Firstly, we calculate the indicator δ_{ijk} aiming to determine which decision variables will be fixed. One can observe that, for $j = k$, the indicator must not be calculated since the setup times are sequence-dependent. Thereafter, based on the values of δ_{ijk} and α, we construct the list with the decision variables to be fixed. Next, we set as zero the decision variables in the list previously generated. Finally, we run the reduced model, which is a subproblem of the original problem. Algorithm 1 describes the proposed Fixed Variable List Algorithm (FVLA).

Algorithm 1: Fixed Variable List Algorithm

Step 1: For all i, j, and k, with $j \neq k$, calculate δ_{ijk} as in Eq. (21).
Step 2: Construct the list FVL, as in Eq. (22).
Step 3: Set as zero the decision variables in the determined FVL.
Step 4: Taking into account the decision variables fixed in Step 3, solve the reduced problem.

Table 4 presents the setup times for the first machine of the FVLA example. Using Eq. (21), we can calculate the following values: δ_{112} = (2 - 2) / (5 - 2) = 0, δ_{113} = (4 - 2) / (5 - 2) = 2/3, δ_{114} =(5 - 2) / (5 - 2) = 1, δ_{121} = (1 - 1) / (4 - 1) = 0, δ_{123} = (2 - 1) / (4 - 1) = 1/3, δ_{124} = (4 - 4) / (4 - 1) = 0, δ_{131} = (6 - 3) / (9 - 3) = 1/2, δ_{132} = (3 - 3) / (9 - 3) = 0, δ_{134} = (9 - 3) / (9 - 3) = 1, δ_{141} = (2 - 1) / (5 - 1) = 1/4, δ_{142} = (5 - 1) / (5 - 1) = 1, and δ_{143} = (1 - 1) / (2 - 1) = 0. Assuming α = 2/3, we can set as zero the decision variables associated δ values greater than 2/3: $x_{113} = x_{114} = x_{124} = x_{134} = x_{142} = 0$.

5.2. Clustering sequence algorithm

After some preliminary experiments with the MILP models in Section 4, we could observe that, in some test instances, one or more machines presented the same permutation in the optimal solutions. Since M2 allows formulating a problem with a limited number of permutation sequences, the idea is to identify groups of similar machines so they may have the same sequence of orders. Thus, we developed a matheuristic that applies the concept of cluster analysis from multivariate statistics.

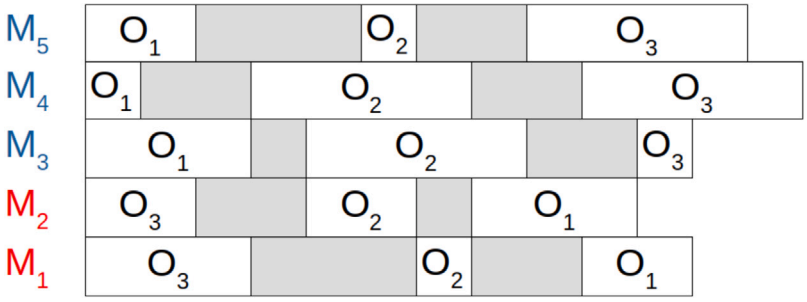

Fig. 2. An illustrative example of the CSA matheuristic.

In the problem under study, we can observe a gradual improvement in the value of the objective function values with the increasing of the number of allowed permutations. An ideal situation is an approach where the number of permutations is free; however, since we have faced an NP-hard problem, we cannot guarantee that the method finds the optimal solution within a given time limit. Furthermore, in the preliminary computational results, we could observe that the setup times distribution played a key role in the capability of solution procedures to find high-quality solutions, since with the increasing of setup times also increases the difficulty to solve a given instance. Thereby, the machines can be clustered, taking into account the similarities in the setup times. Once adopted this criterion, the machines in the same cluster can present the same sequence, and several decision variables can be reduced. Although this approach is not able to determine the bottleneck machine, the proposed matheuristic presents an approximate procedure for its determination.

Let G_i be a vector with the cluster of each machine, which is the output of a given clustering procedure. If two machines i' and i'' are grouped in the same cluster, we can add the constraint set expressed as in Eq. (23):

$$x_{i'jk} = x_{i''jk}, i' = 1, \ldots, m, i'' = i, \ldots, m-1, i' \neq i'', G_{i'} = G_{i''} \forall j \in J; \quad \forall k \in K; \tag{23}$$

In this way, the number of decision variables is drastically reduced, depending on the number of clusters considered. As stated in the previous matheuristic, the number of fixed variables plays a key role in the quality of solutions found, as well as in the computational effort required for the model resolution. Algorithm 2 describes the Clustering Sequence Algorithm (CSA).

Algorithm 2: Clustering Sequence Algorithm

Step 1: For each machine i and for each order k, with $j \neq k$, calculate $\gamma_k = \sum_{j=1}^{n} s_{ijk}$.

Step 2: Apply a clustering algorithm on the γ_k values to determine $\mathcal{K}$ clusters.

Step 3: Considering the generated clusters, insert the constraint set (23).

Step 4: Taking into account the decision variables fixed in Step 3, solve the reduced problem defined by Eqs. (3)-(10).

Fig. 2 illustrates the behavior of the proposed CSA. In this example, we consider five machines. A clustering algorithm can be executed, taking into account the setup times of these machines. In this example, two clusters were generated: the first one containing the machines 1 and 2, and the second one the remainder machines. Considering the machines 1 and 3 as the centroids of each cluster, the sequence $\Pi_1 = \{3, 2, 1\}$ is allocated in machines 1 and 2, and the sequence $\Pi_3 = \{1, 2, 3\}$ is allocated in machines 3, 4, and 5. In this way, we adopt the same sequence for each cluster, reducing the number of decision variables of the problem.

6. Computational experiments

6.1. Experimental design

Initially, we tried to replicate the test instances generated by Framinan and Perez-Gonzalez (2017), adding setup times. However, the explicit consideration of sequence-dependent setup times makes the problem even more complex. For values of n greater than 30, the model usually failed to solve the instances with the specified time limit of 300 s. Furthermore, we had as an aim to study the effect of the maximum number of permutations allowed, as defined in M2. After some preliminary computational experiments, we could observe that M2 increased the computational effort required to solve a given test instance. Thereby, we generated two sets of instances: the first set with small-sized test instances (Testbed 1), and the second set with large-sized test instances (Testbed 2).

Testbed 1 is composed of instances with $m = \{6, 8\}$, $n = \{5, 6\}$, and $s \in \{[1, 25], [1, 75], [1, 125]\}$. The purpose of this testbed is to evaluate the effect of the allowed number of permutations in the quality of the solutions obtained by the second mathematical formulation proposed, in comparison with the global optimal solution. We adjusted these parameters empirically until we found a testbed where the majority of instances were solved for both models with the specified time limit. Also, as we investigate the effect of the fixed permutations, the number of machines would be greater than or equal to the number of orders. Testbed 2 is composed of instances with $m = \{10, 20\}$, $n = \{20, 30\}$, and $s \in \{[1, 25], [1, 75], [1, 125]\}$. The purpose of this testbed is to evaluate the quality of the approximate methods under comparison. Ten instances of each combination of machines orders and setups have been generated. In both sets, the processing times were generated according to a uniform [1, 99] distribution, as suggest by Framinan and Perez-Gonzalez (2017). The proposed test instances are available here.

6.2. Statistics used in the analysis of the computational experiments

We use as performance measures the relative percentage deviation (RPD) and the success rate (SR). RPD is calculated for each instance as in Eq. (24):

$$\text{RPD} = \frac{v_{\text{METHOD}} - v_{\text{BEST}}}{v_{\text{BEST}}} \times 100\% \tag{24}$$

where v_{METHOD} denotes the best objective function value obtained by a given method while v_{BEST} denotes the best objective function value obtained among all methods. To summarize the computational results, we calculate average RPD (ARPD) for a given method taking several runs of an instance, or even the runs in a given set of instances.

SR is calculated as the number of times that a given method results in the best solution (with or without a draw) divided by the number of test instances in a given instance class, is expressed as in Eq. (25):

$$\text{SR} = \frac{n_{\text{BEST}}}{n_{\text{INST}}} \times 100\% \tag{25}$$

Table 5
Computational results for small-sized test instances.

Set	m	n	s	M1		M2(1)		M2(2)		M2(3)		M2(4)	
				ARPD (%)	t (s)	ARPD (%)	t (s)	ARPD (%)	t (s)	ARPD (%)	t (s)	ARPD (%)	t (s)
1	6	5	[1,25]	0.0	0.2	1.1	0.2	0.0	2.0	0.0	1.8	0.0	4.9
2	6	5	[1,75]	0.0	0.5	9.2	0.2	2.8	1.2	0.8	8.4	0.2	34.5
3	6	5	[1,125]	0.0	0.6	20.0	0.2	5.2	1.3	1.6	5.7	0.6	39.7
4	8	5	[1,25]	0.0	0.4	1.5	0.5	0.1	15.6	0.0	5.4	0.0	8.2
5	8	5	[1,75]	0.0	1.2	11.7	0.5	3.2	15.4	0.0	107.5	0.0	58.0
6	8	5	[1,125]	0.0	10.4	20.1	0.6	7.6	27.3	2.8	149.3	1.4	168.6
7	6	6	[1,25]	0.0	0.3	1.2	0.2	0.0	2.7	0.0	4.4	0.0	3.6
8	6	6	[1,75]	0.0	0.3	6.8	0.3	1.4	1.0	0.0	3.9	0.0	12.1
9	6	6	[1,125]	0.0	0.5	19.5	0.3	5.7	1.3	2.0	5.1	1.3	58.7
10	8	6	[1,25]	0.0	0.8	1.7	0.6	0.0	28.5	0.0	11.5	0.0	48.3
11	8	6	[1,75]	0.0	1.2	10.3	0.8	3.0	59.5	0.2	39.1	0.0	50.7
12	8	6	[1,125]	0.0	4.3	22.1	0.8	8.0	46.8	3.4	106.5	2.5	243.7
			Average	**0.0**	**1.7**	**10.4**	**0.4**	**3.1**	**16.9**	**0.9**	**37.4**	**0.5**	**60.9**

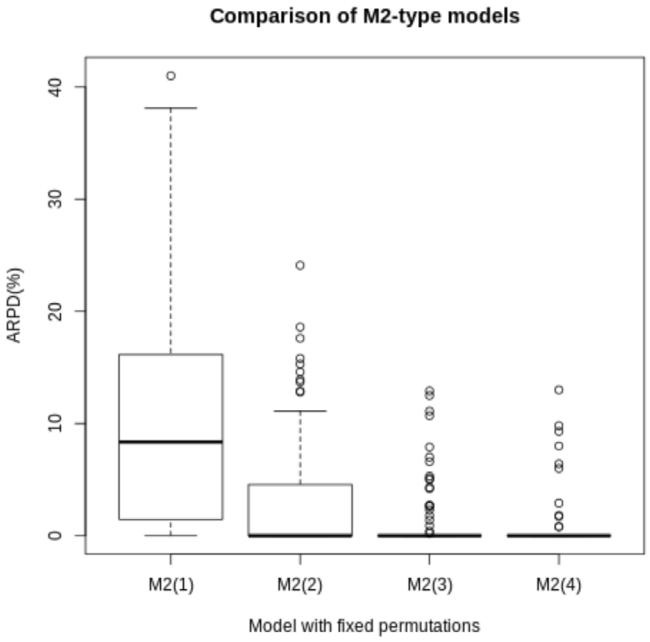

Fig. 3. Boxplot for ARPD values depending on the number of fixed permutations.

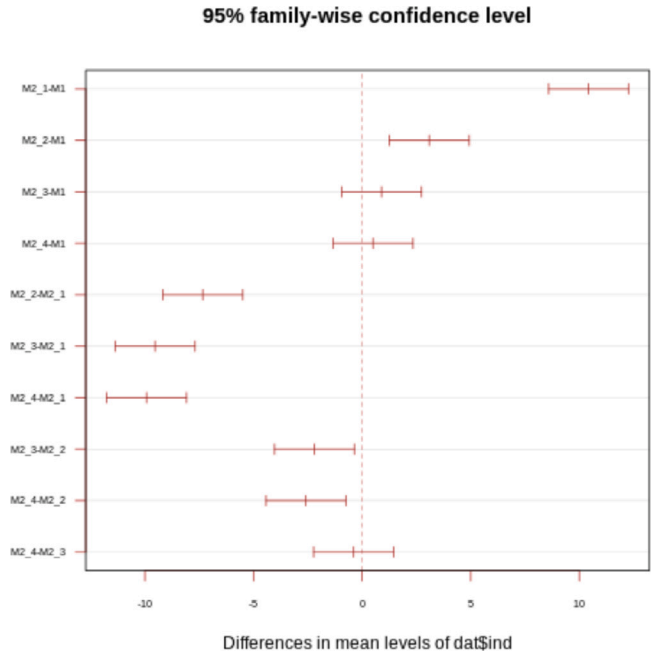

Fig. 4. Tukey confidence intervals for ARPD in small-sized instances.

where n_{BEST} is the number of instances in which a given method achieved the best solution and n_{INST} is the number of instances in the given instance set. Finally, we also used the computational time (t), expressed in seconds, to evaluate the performance of the methods under comparison.

6.3. Results and discussion

We implement the matheuristics presented in Section 5, as well as a number of methods from related scheduling problems (see the list of these methods in Section 6.3.2), using Julia with Atom IDE (https://atom.io/). For the mathematical programming models as well as the proposed matheuristics, we use the commercial solver IBM ILOG CPLEX (https://www.ibm.com/products/ilog-cplex-optimization-studio) version 12.10 with JuMP library (https://www.juliaopt.org/JuMP.jl/stable/) (Lubin & Dunning, 2015). We perform the computational experience on a PC with AMD Ryzen 3 3200U APU 3.5 GHz Dual-Core and 8GB memory, with the Ubuntu 20.04 LTS operating system. For all methods under comparison, we adopt a time limit $t_{limit} = 300$ s.

6.3.1. Results for small-sized test instances

Aiming to evaluate the impact of the maximum allowed number of permutations in the quality of the solutions, as well as in the computational effort required to solve a given instance, we have carefully designed the small-sized test instances. After several preliminary computational experiments, we have found an experimental design that enabled the evaluation of the above-mentioned characteristics. Table 5 illustrates the computational results for small-sized test instances. Recall that M1 is the model defined by Eqs. (3)–(10), and M2(p_{max}) is the model defined by Eqs. (11)–(20), with $\mathcal{P} = \{1, 2, 3, 4\}$.

Based on the results presented in Table 5, we can observe that M1 was the best method in terms of the quality of the solutions found. This model provides the optimal solution for all the test instances in this testbed. Fig. 3 illustrates boxplots for ARPD values, also taking into account the M2-type models.

These results speaks for the need of not restricting the search to permutation encoding schemes, as they do not generally lead to the best solutions. We can emphasize that the increase in the number of allowed permutations improves the quality of the solutions. However, for the smaller distribution of setups, all the models can find optimal or near-optimal solutions. With the increase in setup times, the M2-type models have more problems to find high-quality solutions. Regarding the success rate indicator, M1, M2(1), M2(2), M2(3), and M2(4) present SR values equal to 100.0%, 16.7%, 51.7%, 81.7%, and 90.0% respectively.

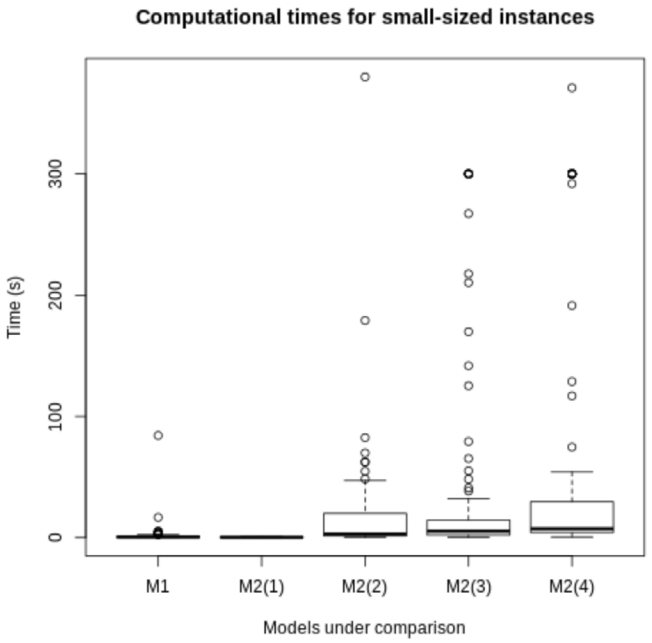

Fig. 5. Boxplot for average computational times considering small-sized instances.

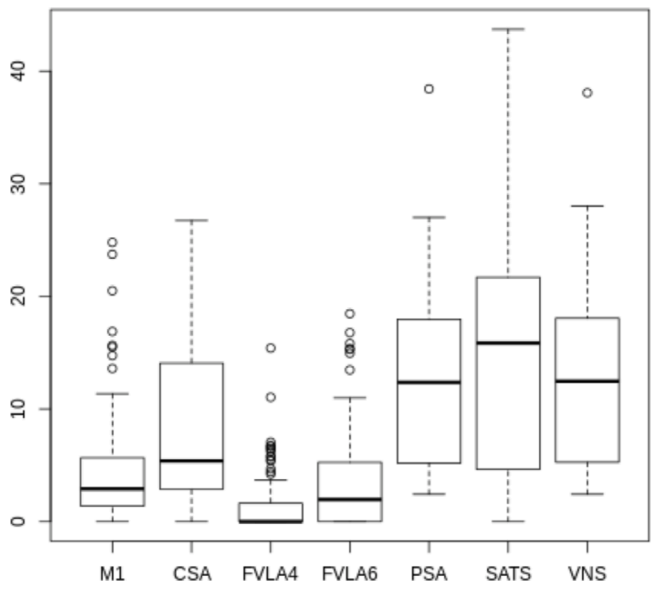

Fig. 6. Boxplot for ARPD values on the large-sized test instances.

Aiming to achieve a pairwise comparison between the models evaluated in the small-sized instances, we perform an ANOVA procedure, followed by a Tukey's test (Montgomery, 2017). We obtained a F-value equal to 81.91 in ANOVA. Since the critical f-value is 4.39, we could observe a statistically significant difference between the evaluated methods.

Fig. 4 illustrates the Tukey multiple comparisons of means with 95% family-wise confidence level. Based on the results obtained, we can observe that M1 outperforms models M2(1) and M2(2), as well as M2(2), M2(3), while M2(4) outperforms M2(1). Also, all other differences among the models are not statistically significant.

Fig. 5 illustrates the boxplots for the computation times, considering the five models under comparison. We can observe that M2(1) presents the lowest computational times, followed by model M1. Besides, models M2(2), M2(3), and M2(4) present much larger computational times as compared to the above-mentioned models.

6.3.2. Results for large-sized test instances

After the preliminary computational experiments, we could observe that M2-type models have failed to find integer solutions in several test instances within the time limit specified. Thus, we have not considered these methods in the computational tests for large-sized test instances.

With respect to the FVLA matheuristic, initially we have tried following values of α: $\alpha = \{20, 40, 60, 80\}$. For $\alpha = 20$ and $\alpha = 80$, FLVA failed to find integer solutions within the time limit in several test instances. Consequently, we have considered only $\alpha = 40$ and $\alpha = 60$ in the definitive computational experiments.

The computational experiments with small-sized test instances provided an understanding of the impact of a given number of allowed permutations in the quality of generated solutions. The number of $\mathcal{K}$ allowed permutations is the input of the CSA matheuristic. Based on the above-mentioned computational experiments, initially we have tried the following values for the clusters: $\mathcal{K} = \{\lceil 0.6\ m \rceil, \lceil 0.8\ m \rceil\}$.

Since the value of $\lceil 0.6\ m \rceil$ clusters failed to find integer solution within the specified time limit in several test instances, we have considered only $\lceil 0.8\ m \rceil$ clusters in the definitive computational experiments. For the clustering procedure, we used the well-known $\mathcal{K}$-means algorithm, with Clustering library (https://juliastats.org/Clustering.jl/dev/kmeans.html).

Furthermore, since the problem under study has not been addressed before in the revised literature, we compare our matheuristics with different approximate algorithms from related scheduling problems. More specifically, we compare the proposed methods with others from related problems using Non-Permutation Encoding (NPE) in addition to the comparison with solution procedures using Permutation Encoding (PE). It is to note that, even if the natural encoding of a given scheduling problem is the NPE, this does not necessarily mean that a solution procedure using NPE would yield better results than others using PE. This is known e.g. for the parallel machine setting and for the hybrid flowshop layout, where the differences in the performance between NPE and PE procedures is so small that PE procedures are overwhelmingly used as they are much faster and therefore can explore more solutions in less CPU time (see in this regard Fernandez-Viagas, Perez-Gonzalez, & Framinan, 2019 regarding the hybrid flow shop layout). Therefore, the tested procedures are:

- Perturbative Search Algorithm (PSA) metaheuristic, proposed by Riahi et al. (2019) for the customer order scheduling with total completion time minimization. This heuristic is considered to be the most efficient for this problem, and we adapt it to our problem as a proxy of the best available methods from the classical customer order scheduling problem (recall that the classical order scheduling problem with makespan objective is trivial).
- The hybrid Simulated Annealing and Tabu Search (SATS) proposed by Lin and Ying (2009) for the non-permutation flow shop with makespan minimization. Although different from our problem, the non-permutation flow shop problem allows for a similar solution encoding (i.e. one sequence per machine, which results in a NPE), therefore we adapt this state-of-the-art procedure to our problem in order to include one feature – that of solution encoding using a sequence per machine – which is not included in the PSA.
- Variable Neighborhood Search (VNS) proposed by Kuo, Chen, and Yeh (2020) for the single-machine scheduling problem with sequence-dependent setup times and delayed precedence constraints to minimize the makespan. By considering this approximate procedure, we compare our proposals with a state-of-the-art algorithm for a similar scheduling problem with setup considerations, which is a feature not included either in PSA or in SATS.

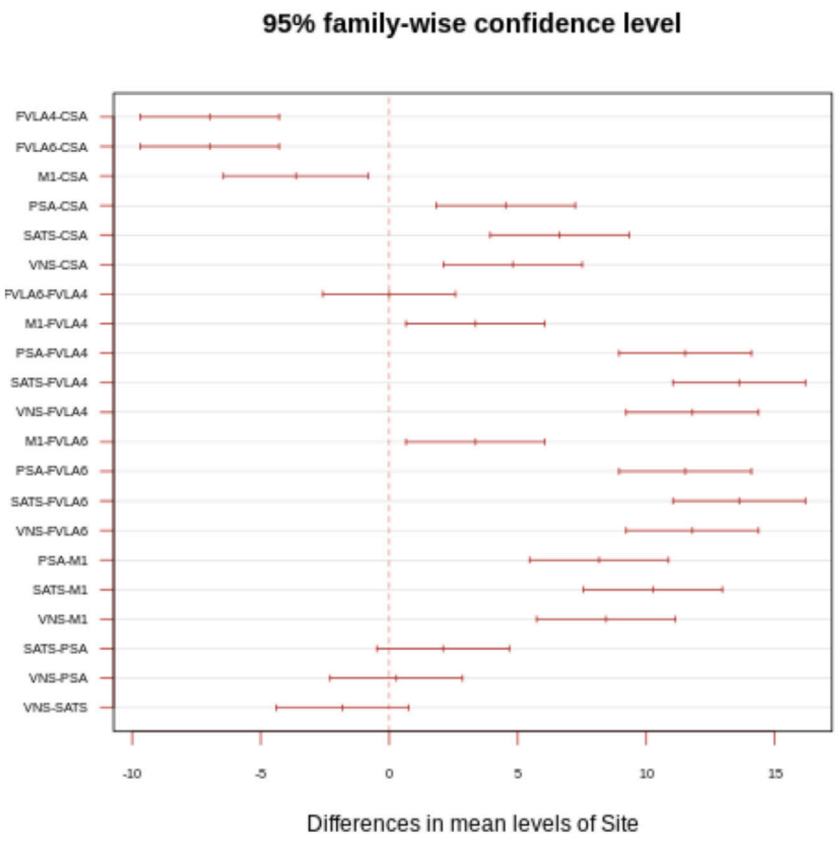

Fig. 7. Tukey confidence intervals for ARPD in large-sized instances.

Regarding the manner in which the above algorithms are adapted, we note the following. We adapt the PSA for the objective of makespan minimization, taking into consideration sequence-dependent setup times. In the diversification procedure of the original PSA, the first half of the list D is taken from the list E, and the second half is taken randomly. Since the criterion of sorting customer orders in non-increasing order of extra time is not applicable in our variant (due to the sequence-dependent setup times), we have constructed the entire list randomly, such as in an standard iterated greedy algorithm. To construct the list $\mathcal{L}$, we adopt the Total Weighted Processing Times (TWPT) dispatching rule. This list is used for the Permutation Construction and Explorations (PCE) heuristic to build the initial solution, following Riahi et al. (2019). We consider the same original parameters of PSA, as reported by Riahi et al. (2019): $D = 6$, $R = 0.005$, where D controls the exclusion of customer orders from the input solution, and R controls the acceptance of solutions with worse objective function values.

Regarding SATS, we mimic the same steps from the original algorithm, adapting the objective function and constraints to the problem under study. The initial solutions is randomly generated and the algorithm is repeated until the specified time limit. We consider the same parameters of SATS, as reported by Lin and Ying (2009), i.e.: $T_0 = 100$, $T_F = 0.5$, $\alpha_{TEMP} = 0.95$, $I_{ITER} = 200$, $min_T = 3$, and $max_T = 5$.

Regarding VNS, we adapt the objective function and the problem constraints to our variant. The initial solution is also generated randomly, and we reproduce the parameters considered by Kuo et al. (2020), i.e.: $T_s = -(Z_{max} - Z_{min})/\log_e 0.5$, $T_e = -(Z_{max} - Z_{min}) \times 0.0001/\log_e 0.05$, $\alpha_{TEMP} = 0.99$, $K = 1{,}000{,}000$, and $M = 50$. We adapt the values of Z_{max} and Z_{min} to the problem under study: $Z_{max} = n \times p_{max} + (n-1) \times s_{max}$ and $Z_{min} = n \times p_{min} + (n-1) \times s_{min}$.

Finally, note that all the algorithms are allowed a time limit of 300 s. Table 6 presents the computational results for the large-sized test instances. Computational times are not reported since all the methods under comparison used the entire time limit of 300 s. We can observe that M1 and CSA methods fail to find integer solutions in the sets of instances 11 and 12. On the other hand, all other methods under comparison returns feasible integer solutions in the evaluated testbed. The $FLVA_{0.4}$ matheuristic obtain the smallest ARPD value among the evaluated methods. Also, PSA returns the largest ARPD value.

Table 7 illustrates the success rates for all the considered methods. Based on the achieved results, we can observe that FVLA presents the highest SR values. Although SATS returned the worse ARPD value, this algorithm yields the third-highest SR value.

Fig. 6 illustrates the boxplot for ARPD values on large-sized test instances. Once again, we perform an ANOVA procedure aiming to evaluate the difference between the ARPD values found. ANOVA returns a statistics $F = 85.9$, a value much higher than the critical value $f = 4.39$. Since the difference among the methods is statistically significant, we perform Tukey tests, for which the results are illustrated in Fig. 7. On one hand, we can emphasize that $FVLA_{0.4}$ and $FVLA_{0.6}$ matheuristics outperform all other methods. On the other hand, the PSA metaheuristic presents the greatest ARPD values for all sets of instances.

7. Concluding remarks

In this paper, a new variant of the customer order scheduling problem has been investigated, taking into consideration explicitly sequence-dependent setup times. The objective function is the makespan minimization. Two mixed-integer linear programming (MILP) formulations have been developed for the problem under study. Furthermore, two matheuristics based on setting some of the decision variables in the MILP models have also been proposed.

The computational experience carried out illustrates that the proposed Fixed Variable List Algorithm outperforms the PSA metaheuristic – which is the most efficient so far for the classical customer order scheduling problem – as well as the SATS, and VNS metaheuristics from

Table 6
Average relative percentage deviation for large-sized test instances.

Set	m	n	s	M1	CSA	$FVLA_{0.4}$	$FVLA_{0.6}$	PSA	SATS	VNS
1	10	20	[1,25]	2.7	2.6	3.2	2.6	5.9	1.6	6.0
2	10	20	[1,75]	1.7	4.3	4.5	1.3	15.7	18.7	16.0
3	10	20	[1,125]	1.4	10.8	3.0	4.5	26.1	31.7	26.5
4	10	20	[1,25]	3.1	3.7	0.6	0.8	4.3	1.0	4.5
5	10	20	[1,75]	8.8	14.3	0.8	4.1	13.2	16.0	13.4
6	10	20	[1,125]	14.6	18.2	0.4	5.9	18.8	23.7	19.4
7	20	30	[1,25]	1.6	1.3	1.1	0.9	4.5	4.0	4.6
8	20	30	[1,75]	2.7	7.6	1.5	0.5	15.8	19.8	16.1
9	20	30	[1,125]	6.3	16.5	0.6	4.5	25.1	30.4	25.5
10	20	30	[1,25]	3.7	3.7	0.0	1.8	3.9	4.3	4.1
11	20	30	[1,75]	–	–	0.2	6.0	9.6	13.0	9.9
12	20	30	[1,125]	–	–	0.0	7.7	11.1	15.0	11.4
			Average	**4.7**	**8.3**	**1.3**	**3.4**	**12.8**	**14.9**	**13.1**

Table 7
Comparison of success rates for each method.

Method	SR (%)
M1	4.2
CSA	2.5
FVLA_0.4	58.3
FVLA_0.6	27.5
PSA	0.0
SATS	7.5
VNS	0.0

related problems with the same computational effort. The proposed Clustering Sequence Algorithm presents better ARPD values than the M1 MILP as well as than the metaheuristics adapted from the related problems, i.e. PSA, SATS and VNS. However, this method fails to find feasible solutions in the test instances with 20 machines, 30 orders, and setup distributions $s \in \{[1,75],[1,125]\}$.

As extensions of this work, other possibilities for the clustering algorithm required in the CSA matheuristic can be explored. In the proposed FVLA, we did not check if the construction of the list leads to the infeasibility of a given model, so perhaps more sophisticated, non-trivial, modifications of the FVLA could take this aspect into account. Another possibility is the consideration of different objective functions for the proposed variant, such as the total completion time or total tardiness.

CRediT authorship contribution statement

Bruno de Athayde Prata: Conceptualization, Investigation, Software, Formal analysis, Writing - original draft, Visualization, Funding acquisition. **Carlos Diego Rodrigues:** Formal analysis, Methodology, Writing. **Jose Manuel Framinan:** Conceptualization, Writing - review & editing, Supervision, Project administration, Funding acquisition.

Acknowledgments

This study was financed in part by the Coordination for the Improvement of Higher Education Personnel (CAPES), the National Council for Scientific and Technological Development (CNPq), through Grant no. 303594/2018-7, the Spanish Ministry of Science and Education (ASSORT project, Grant no. PID2019-108756RB-I00), and the Andalusian Government (projects DEMAND Grant no. P18-FR-1149 and EFECTOS, Grant no. US-1264511).

References

Abreu, L. R., Cunha, J. O., Prata, B. A., & Framinan, J. M. (2020). A genetic algorithm for scheduling open shops with sequence-dependent setup times. *Computers & Operations Research*, *113*, Article 104793. http://dx.doi.org/10.1016/j.cor.2019.104793, URL http://www.sciencedirect.com/science/article/pii/S0305054819302357.

Allahverdi, A., Ng, C., Cheng, T., & Kovalyov, M. Y. (2008). A survey of scheduling problems with setup times or costs. *European Journal of Operational Research*, *187*(3), 985–1032.

Feo, T. A., & Resende, M. G. (1989). A probabilistic heuristic for a computationally difficult set covering problem. *Operations Research Letters*, *8*(2), 67–71.

Feo, T. A., & Resende, M. G. (1995). Greedy randomized adaptive search procedures. *Journal of Global Optimization*, *6*(2), 109–133.

Fernandez-Viagas, V., Perez-Gonzalez, P., & Framinan, J. M. (2019). Efficiency of the solution representations for the hybrid flow shop scheduling problem with makespan objective. *Computers & Operations Research*, *109*, 77–88.

Framinan, J. M., & Perez-Gonzalez, P. (2017). New approximate algorithms for the customer order scheduling problem with total completion time objective. *Computers & Operations Research*, *78*, 181–192.

Framinan, J. M., & Perez-Gonzalez, P. (2018). Order scheduling with tardiness objective: Improved approximate solutions. *European Journal of Operational Research*, *266*(3), 840–850.

Framinan, J. M., Perez-Gonzalez, P., & Fernandez-Viagas, V. (2019). Deterministic assembly scheduling problems: A review and classification of concurrent-type scheduling models and solution procedures. *European Journal of Operational Research*, *273*(2), 401–417. http://dx.doi.org/10.1016/j.ejor.2018.04.033, URL http://www.sciencedirect.com/science/article/pii/S0377221718303369.

Hart, J. P., & Shogan, A. W. (1987). Semi-greedy heuristics: An empirical study. *Operations Research Letters*, *6*(3), 107–114.

Julien, F., & Magazine, M. (1990). Scheduling customer orders: An alternative production scheduling approach. *Journal of Manufacturing and Operations Management*, *3*(3), 177–199.

Kung, J.-Y., Duan, J., Xu, J., Chung, I., Cheng, S.-R., Wu, C.-C., et al. (2018). Metaheuristics for order scheduling problem with unequal ready times. *Discrete Dynamics in Nature and Society*, *2018*.

Kuo, Y., Chen, S.-I., & Yeh, Y.-H. (2020). Single machine scheduling with sequence-dependent setup times and delayed precedence constraints. *Operational Research*, *20*(2), 927–942.

Lee, I. S. (2013). Minimizing total tardiness for the order scheduling problem. *International Journal of Production Economics*, *144*(1), 128–134.

Leung, J. Y.-T., Li, H., & Pinedo, M. (2005). Order scheduling in an environment with dedicated resources in parallel. *Journal of Scheduling*, *8*(5), 355–386.

Lin, W.-C., Xu, J., Bai, D., Chung, I.-H., Liu, S.-C., & Wu, C.-C. (2019). Artificial bee colony algorithms for the order scheduling with release dates. *Soft Computing*, *23*(18), 8677–8688.

Lin, W.-C., Yin, Y., Cheng, S.-R., Cheng, T. E., Wu, C.-H., & Wu, C.-C. (2017). Particle swarm optimization and opposite-based particle swarm optimization for two-agent multi-facility customer order scheduling with ready times. *Applied Soft Computing*, *52*, 877–884.

Lin, S.-W., & Ying, K.-C. (2009). Applying a hybrid simulated annealing and tabu search approach to non-permutation flowshop scheduling problems. *International Journal of Productions Research*, *47*(5), 1411–1424.

Lubin, M., & Dunning, I. (2015). Computing in operations research using julia. *INFORMS Journal on Computing*, *27*(2), 238–248.

Moccellin, J. V., Nagano, M. S., Pitombeira Neto, A. R., & Prata, B. A. (2018). Heuristic algorithms for scheduling hybrid flow shops with machine blocking and setup times. *Journal of the Brazilian Society of Mechanical Sciences and Engineering*, *40*(2), 40. http://dx.doi.org/10.1007/s40430-018-0980-4.

Montgomery, D. C. (2017). *Design and analysis of experiments*. John Wiley & Sons.

Pinedo, M. (2018). *Scheduling: Theory, algorithms, and systems*. Springer.

Resende, M. G., & Ribeiro, C. C. (2016). *Optimization by GRASP*. Springer.

Riahi, V., Newton, M. H., Polash, M., & Sattar, A. (2019). Tailoring customer order scheduling search algorithms. *Computers & Operations Research*, *108*, 155–165. http://dx.doi.org/10.1016/j.cor.2019.04.015, URL http://www.sciencedirect.com/science/article/pii/S030505481930098X.

Shi, Z., Wang, L., Liu, P., & Shi, L. (2017). Minimizing completion time for order scheduling: Formulation and heuristic algorithm. *IEEE Transactions on Automation Science and Engineering*, *14*(4), 1558–1569.

Wagneur, E., & Sriskandarajah, C. (1993). Openshops with jobs overlap. *European Journal of Operational Research*, *71*(3), 366–378.

Wu, C.-C., Yang, T.-H., Zhang, X., Kang, C.-C., Chung, I.-H., & Lin, W.-C. (2019). Using heuristic and iterative greedy algorithms for the total weighted completion time order scheduling with release times. *Swarm and Evolutionary Computation*, *44*, 913–926. http://dx.doi.org/10.1016/j.swevo.2018.10.003, URL http://www.sciencedirect.com/science/article/pii/S2210650218301317.

Xu, X., Ma, Y., Zhou, Z., & Zhao, Y. (2015). Customer order scheduling on unrelated parallel machines to minimize total completion time. *IEEE Transactions on Automation Science and Engineering*, *12*(1), 244–257.

Xu, J., Wu, C.-C., Yin, Y., Zhao, C., Chiou, Y.-T., & Lin, W.-C. (2016). An order scheduling problem with position-based learning effect. *Computers & Operations Research*, *74*, 175–186.