Computers & Industrial Engineering 198 (2024) 110634

Contents lists available at ScienceDirect

Computers & Industrial Engineering

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

# Changeover minimization in the production of metal parts for car seats ☆

J. Manuel Colmenar [a,*], Manuel Laguna [b], Raúl Martín-Santamaría [a]

[a] Universidad Rey Juan Carlos, Calle Tulipan s/n, Mostoles, 28933, Madrid, Spain
[b] University of Colorado Boulder, Leeds School of Business, 995 Regent Dr, Boulder, 80309, CO, United States of America

ARTICLE INFO

Dataset link: https://doi.org/10.5281/zenodo.13912006

*Keywords:*
Lot sizing
Multi-period production scheduling
Nonidentical parallel machines
Metaheuristic optimization

ABSTRACT

We tackle a capacitated lot-sizing and scheduling problem (CLSP) with the main objective of minimizing changeover time in the production of metal parts for car seats. Changeovers occur when a machine (or production line) is reconfigured to produce a different product or part, leading to production downtime and loss of efficiency. In this study, we first provide a mixed-integer programming (MIP) formulation of the problem. We test the limits of solving the problem with commercial mathematical programming software. We also propose two approaches to tackle instances found in practice for which the mathematical programming model is not a viable solution method. Both approaches are based on partitioning the entire production of a part into production runs (or work slots). In the first approach, the work slots are assigned to machines and sequenced by a metaheuristic that follows the search principles of the GRASP (greedy randomized adaptive procedure) and VNS (variable neighborhood search) methodologies. In the second approach, we develop a Hexaly Optimizer (formerly known as LocalSolver) model to assign and sequence work slots. The study provides insights into how to minimize changeovers and improve production efficiency in metal parts manufacturing for car seats. The findings of this study have practical implications for the auto-part manufacturing industry, where efficient and cost-effective production is critical to meet the demands of the market.

## 1. Introduction

The efficient scheduling of machines is a crucial aspect of manufacturing processes, including the production of metal parts for car seats. Machine scheduling involves determining the order and timing of tasks to be performed on each machine to minimize production time, increase productivity, and reduce costs. We present a study on machine scheduling in the manufacturing of metal parts for car seats. We propose a mathematical model for scheduling the production runs of metal parts on a set of machines, considering machine availability, processing times, and changeover times. The model is solved using a mixed-integer linear programming approach. We also develop heuristic approaches to tackle problem instances of the size found in practical settings. Our computational testing focuses on determining the conditions under which the problem can be tackled by solving the MIP formulation with a commercial mathematical programming software and at what point it is advisable to switch to a heuristic-based approach.

Our approach of first formulating the problem as a mathematical program and test its limits is typical in the production literature. It also mimics the approach taken in industry, where companies first try to solve optimization problems with general solvers before embarking on the development of specialized solutions. This is why in addition to a mathematical programming solver (Gurobi) we tried Hexaly, a heuristic-based general-purpose solver. Once we verified that a problem-specific solution method was necessary and that metaheuristics were the appropriate optimization technology, we chose to pursue two prominent approaches: Greedy Randomized Adaptive Search Procedures (GRASP) (Feo & Resende, 1995; Resende & Ribeiro, 2016) and Variable Neighborhood Search (VNS) (Bamoumen, Elfirdoussi, Ren, & Tchernev, 2023; Kyriakakis, Aronis, Marinaki, & Marinakis, 2023; Yildiz, Ozcan, & Cevik, 2023). Our motivation for this selection is based on the documented success of these methodologies in production scheduling and related combinatorial problems, such as those associated with vehicle routing. Our literature review convinced us to focus on construction and improvement methods, where GRASP and VNS surfaced as the best choices. The pseudo-greedy GRASP constructions produce a sampling of solutions with a controlled diversification that provides effective starting points for VNS. The flexibility embedded in the VNS structure allows for controlling the depth of the search applied to each of the solutions constructed by the first phase of GRASP.

☆ This work has been partially supported by the Spanish Ministerio de Ciencia e Innovación (MCIN/AEI/10.13039/501100011033) under grant refs. RED2022-134480-T, PID2021-126605NB-I00 and by ERDF A way of making Europe.

* Corresponding author.
*E-mail address:* josemanuel.colmenar@urjc.es (J.M. Colmenar).

https://doi.org/10.1016/j.cie.2024.110634
Received 7 February 2024; Received in revised form 30 September 2024; Accepted 6 October 2024
Available online 16 October 2024

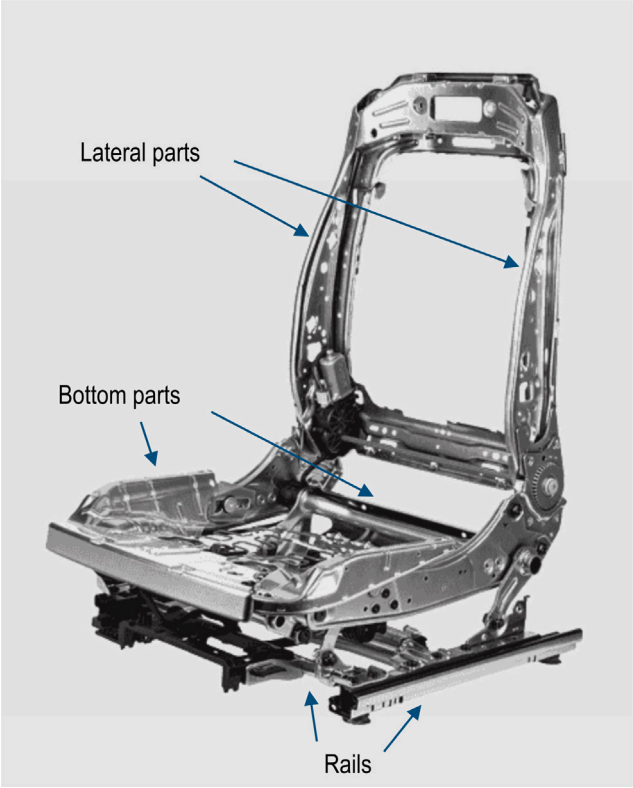



**Fig. 1.** Example of the internal structure of a car seat, formed by metal parts.

### 1.1. Problem description

The problem consists of scheduling the production of metal parts for car seats on a set of production lines. Fig. 1 shows an example of the internal structure of a car seat, formed by several metal parts with various degrees of similarity. For instance, the lateral parts are symmetrical. But the bottom parts are quite different from the rails. Parts that are similar belong to the same profile family.

Each part has a preferred ("home") line and one or two secondary lines. Each part belongs only to one profile family. There are two types of changeovers, one with a long changeover time and another with a short changeover time. A long changeover time occurs when changing a line to produce parts from two different profile families. When changing a line to produce parts within the same profile family, the changeover is short. Demand for each part is estimated for several weeks in a planning horizon. A scheduler must decide on the production sequences for each line and the lot sizes to cover the estimated demand while minimizing the total changeover time.

### 1.2. Illustrative example

The following example illustrates the problem that we are tackling. Suppose that five parts must be produced to meet demand in a planning horizon of 5 weeks. For this simple example, we make three assumptions: (1) the production facility consists of a single machine capable of producing the five parts, (2) there are no tool restrictions, (3) long changeovers are 10 h and a short changeover requires 3 h. A week consists of 10 shifts of 7.5 h for a total of 75 production hours. Table 1 shows the data for this example. The part numbers are in column 1. Column 2 shows the profile family for each part. The production rates, expressed in parts per hour (PPH), are shown in column 3. Finally, columns 4 to 8 show the inventory positions of each part. The inventory positions consider the initial inventory and the weekly demand. The negative numbers indicate the shortages that must be addressed by the production schedule.

**Table 1**
Data for illustrative example.

| Part | Profile | PPH | Week 1 | Week 2 | Week 3 | Week 4 | Week 5 |
|---|---|---|---|---|---|---|---|
| P61045.01 | 61045 | 360 | 1300 | −1800 | −5800 | −5800 | −8200 |
| P61045.02 | 61045 | 240 | 1200 | 400 | −1400 | −3600 | −7800 |
| P61045.03 | 61045 | 120 | −1200 | −2400 | −4800 | −7200 | −18 000 |
| P61048.01 | 61048 | 360 | 6400 | 4410 | −1750 | −2150 | −3500 |
| P61048.02 | 61048 | 300 | 1500 | −900 | −1800 | −3500 | −7000 |

**Table 2**
Feasible production sequence and lot sizing for the illustrative example.

| Run | Part | Quantity | Start | Duration | End | Week |
|---|---|---|---|---|---|---|
| 1 | P61048.01 | 3600 | 0 | 10 | 10 | 1 |
| | Changeover | | 10 | 3 | 13 | 1 |
| 2 | P61048.02 | 7000 | 13 | 23.3 | 36.3 | 1 |
| | Changeover | | 36.3 | 10 | 46.3 | 1 |
| 3 | P61045.03 | 3440 | 46.3 | 28.7 | 75 | 1 |
| | P61045.03 | 7440 | 75 | 62 | 137 | 2 |
| | Changeover | | 137 | 3 | 140 | 2 |
| 4 | P61045.01 | 3600 | 140 | 10 | 150 | 2 |
| | P61045.01 | 4600 | 150 | 12.8 | 162.8 | 3 |
| | Changeover | | 162.8 | 3 | 165.8 | 3 |
| 5 | P61045.02 | 7800 | 165.8 | 32.5 | 198.3 | 3 |
| | Changeover | | 198.3 | 3 | 201.3 | 3 |
| 6 | P61045.03 | 2847 | 201.3 | 23.7 | 225 | 3 |
| | P61045.03 | 4273 | 225 | 35.6 | 260.6 | 4 |

Table 2 shows a feasible production schedule consisting of 6 production runs (of a minimum of 10 h), 1 long changeover, and 4 short changeovers, for a total of 22 h of changeover time. No changeover time is charged for setting up the machine for the first production run.

A lower bound on the minimum number of changeovers of each type and on the total changeover time can be obtained by relaxing the due date constraints. This relaxation allows the completion of the total production of each part in a single run. The optimal sequence for this relaxed problem would be to produce all the parts with the same

profile before changing the machine to produce all the parts with the next profile. For this example, the sequence that orders the production runs by the part numbers (i.e., all parts in the 61045 profile family are produced first followed by all parts in the 6048 profile family) provides a lower bound for the changeover values. The lower bounds are:

- Long changeovers: 1
- Short changeovers: 3
- Changeover time: 19

A long changeover occurs when changing from P61045.03 to P61048.01. The short changeovers are the changes associated with the following pairs of parts (P61045.01, P61045.02), (P61045.02, P61045.03), and (P61048.01, P61048.02). The solution in Table 2 differs from the lower bound by one short changeover. The splitting of the production run is necessary because it is not possible to produce all 18000 units of P61045.03 in a single production run and at the same time meet all the demand obligations in week 2. Some production runs are divided into two rows in Table 2 to account for the week in which the parts are added to the inventory. For instance, production run #3 adds 3440 parts to the inventory of P61045.03 in week one and 7440 parts to the inventory in week 2.

### 1.3. Literature review

It is well known that no two production scheduling problems are the same. There are common elements to many production scheduling problems, such as sequence dependency, lot size determination, and assignment of jobs to machines. However, even the smallest of differences can cause major changes in standard models and solution approaches. In a variety of practical settings, changeover times can have a significant impact on production efficiency and equipment utilization. Discrete-time mixed-integer programming formulations have been used to model scheduling problems in production environments with single and multiple parallel machines (Velez, Dong, & Maravelias, 2017). Due to the complexity of mixed-integer programming (MIP) formulations that model sequence dependency, solution approaches such as column-generation (Kang, Malik, & Thomas, 1999) and genetic algorithms (Roychowdhury, Allen, & Allen, 2017) have been proposed. When the problems also include lot sizing, the models become even more complex because in addition to sequencing the production runs on each machine, the length of the runs must be determined, resulting in what is known as capacitated lot-sizing and scheduling problems (CLSP). The CLSP literature is vast and includes mathematical programming models, as well as heuristics, and metaheuristic approaches. A recent detailed literature review of the CLSP literature can be found in De Armas and Laguna (2020).

Resource sharing, which is common in many production environments, adds to the complexity of the scheduling problem. The issue is that a limited number of additional resources (such as pallets, operators, tools, molds, etc.) are necessary to produce different parts or products. Therefore, optimization models must consider the limited availability of these additional resources when scheduling production on parallel machines (Özpeynirci, Gökgür, & Hnich, 2016). Edis, Oguz and Ozkarahan (Edis, Oguz, & Ozkarahan, 2013) present a comprehensive review and discussion of work on parallel machine scheduling problems with additional resources.

In our review of the existing literature, we found that the CLSP tackled by Xiao, Zhang, Zheng, and Gupta (2013) is closest to the problem we faced when we engaged with the car seat manufacturing facility we studied. The article is motivated by the production of semiconductors. The CLSP considers sequence-dependent setup times, time windows, machine eligibility, and preference constraints. They optimize an objective function consisting of a weighted sum of inventory holding cost, backlog cost, setup time cost, and machine preference cost. They propose an MIP formulation, which they solve using Cplex. They also propose a relax-and-fix algorithm to find initial solutions that are then subjected to fix-and-optimize process consisting of a local search with a neighborhood modification phase. The relax-and-fix approach is a rolling-horizon process with overlapping time intervals. That is, they use a shift forward strategy to solve a sequence of smaller MIP problems by fixing the binary variables for previous periods, enforcing the binary restrictions for the current periods, and relaxing the binary variables for future periods. The fix-and-optimize search is based on a neighborhood scheme that selects a machine to optimize all its associated variables while maintaining the variables of all other machines fixed. A hybrid fix-and-optimize approach that alternates between machine-based decomposition and time-based decomposition is also developed and tested. The experimental results show that Cplex can solve problems with a planning horizon of 14 periods, 10 machines, and 10 to 14 items. The size of these instances corresponds to the real-world application that motivated the research project. Larger instances, with the same number of periods with up to 15 machines and 20 items are solved with the proposed heuristics. The average deviations from the lower bounds found by Cplex are about 12%.

The size of the problems that Xiao, et al. tackle is significantly smaller than the ones we found in the manufacturing facility that we studied. Also, production managers in our study were concerned with first finding solutions with no backlog (i.e., zero inventory shortages) and within those solutions they wanted the one with the least changeover time. For these reasons, we did not attempt to either reproduce Xiao, et al.'s approach or request their code for inclusion in our computational testing.

Koch et al. (2022) formulate a mixed-integer programming model for the production of off-the-road tires. Their formulation includes a large number of constraint sets to model setup times, upstream resources saturation, and customer prioritization, among other things. The model is multi-objective in nature, however, the problem is solved as a single objective with weight values provided by a Taguchi design of experiments. The MIP is not able to solve problems of realistic size and therefore a decomposition approach is suggested. This approach consists of solving a lot sizing problem followed by solving a machine assignment problem. Koch, et al. show that their decomposition approach provides solutions that are better than those constructed by a human scheduler.

## 2. Mathematical programming formulation

Our mixed-integer program assigns parts to machines, determines the production sequence within each period and each machine, and establishes the sizes of each lot. The first two decisions are modeled with binary variables and the third is a set of continuous variables. The model that we first formulated attempted to directly capture the goals of the production managers. The main goal was to find a solution with zero shortages. Once that was accomplished, the goal was to minimize the total changeover time. Finally, subject to the two previous minimization efforts, an attempt was made to find a solution where machine preferences were maximized. Therefore, we modeled the problem as a hierarchical program where minimizing shortages had the highest priority, followed by minimizing total changeover time, and finishing with the maximization of machine preferences. Preferences were expressed in the form of priority values, where the highest priority was zero. In this way, maximizing total machine-preference was equivalent to minimizing priority values.

*Parameters*

| | |
|---|---|
| $J$ : | Set of parts |
| $K$ : | Set of machines |
| $T$ : | Set of periods |
| $r_{jk}$ : | Production rate (units per hour) of part $j$ on machine $k$. If machine $k$ is not capable of producing part $j$ then $r_{jk} = 0$. |

$c_{ij}$ : Changeover time from part $i$ to part $j$. If parts $i$ and $j$ belong to the same profile family, the changeover is short. A long changeover ($c^{long}$) is incurred when parts $i$ and $j$ belong to two different profile families.

$d_{jt}$ : Inventory position of part $j$ in period $t$. The inventory position subtracts the demand in each period from the beginning inventory. The position becomes negative when the accumulated demand exceeds the initial inventory.

$q_{kt}$ : Capacity (given in number of hours) of machine $k$ available in period $t$.

$p_{jk}$ : Priority for producing part $j$ on machine $k$, where $p_{jk} = 0$ if $k$ is the preferred machine for producing part $j$. If $k$ is the second preferred machine for producing part j then $p_{jk} = 1$, and so forth.

*Decision variables*

$I_{jt}^{-}$ : Inventory shortage of part $j$ at the end of period $t$.

$s_{jkt}$ : Sequencing variable for part $j$ on machine $k$ in period $t$. The value of this variable indicates the relative position of the part in the production sequence of machine $k$ in period $t$.

$x_{jkt}$ : Number of production hours of part $j$ on machine $k$ in period $t$. The lot size for part $j$ on machine $k$ in period $t$ is given by $r_{jk}\, x_{jkt}$.

$y_{ijkt}$ : Binary variable that equals 1 if part $j$ immediately follows part $i$ on machine $k$ in period $t$.

$z_{jkt}$ : Binary variable that equals 1 if the setup for part $j$ is carried over from period $t$ to $t+1$ on machine $k$.

*Model*

The hierarchical objective function consists of first minimizing the total shortage. This is because in some scenarios it might not be possible to find a solution for which all the demand can be met on time. Shortages are equivalent to backlog quantities, except that these quantities may remain unfulfilled throughout the planning horizon.

$$\text{Minimize} \sum_{j\in J}\sum_{t\in T} I_{jt}^{-} \tag{1}$$

We then minimize the total changeover time.

$$\text{Minimize} \sum_{i\in J}\sum_{j\in J:i\neq j}\sum_{k\in K}\sum_{t\in T} c_{ij}\, y_{ijkt} \tag{2}$$

All other things being equal (i.e., total shortage and changeover time), we would like to assign parts to their preferred machines.

$$\text{Minimize} \sum_{i\in J}\sum_{j\in J:i\neq j}\sum_{k\in K}\sum_{t\in T} p_{jk}\, y_{ijkt} \tag{3}$$

The demand coverage constraints are formulated as follows:

$$\sum_{k\in K}\sum_{t'=1}^{t'=t} r_{jk}\, x_{jkt'} + I_{jt}^{-} + d_{jt} \geq 0 \quad \forall j \text{ and } t \tag{4}$$

The capacity constraint for each machine limits the total production time and the changeover time to the available time in each period.

$$\sum_{j\in J} x_{jkt} + \sum_{i\in J}\sum_{j\in J:i\neq j} c_{ij}\, y_{ijkt} \leq q_{kt} \quad \forall k \text{ and } t \tag{5}$$

A set of constraints is needed to link the production and the changeover variables. Production of part $j$ on machine $k$ in period $t$ can only occur if machine $k$ is set up to produce part $j$ at the beginning of the period or a changeover to part $j$ occurs during the period. This is enforced by constraint set (6). When production is activated, constraint set (7) enforces a minimum amount of production hours equivalent to the time required for a long changeover.

$$x_{jkt} \leq q_{kt}\left(z_{jkt-1} + \sum_{i\in J:i\neq j} y_{ijkt}\right) \quad \forall j,k \text{ and } t \tag{6}$$

$$x_{jkt} \geq c^{long}\left(z_{jkt-1} + \sum_{i\in J:i\neq j} y_{ijkt}\right) \quad \forall j,k \text{ and } t \tag{7}$$

The following set of constraints model the changeover flow. For each part, machine, and period, the machine is either set to produce $j$ or there is a change from another part into part $j$. When the machine is set to produce part $j$, that is, at least one term on the left-hand side of (8) is 1, then the machine must be changed to produce another part or remain set to produce part $j$ in the following period. Constraints (9) indicate that only one setup may be carried out from one period to the next. That is, machines must be set to produce one part at the beginning of each period. For period 1, the model chooses what to produce first. For the following periods, the setup is carried over from the previous period.

$$z_{jkt-1} + \sum_{i\in J:i\neq j} y_{ijkt} = z_{jkt} + \sum_{i\in J:i\neq j} y_{jikt} \quad \forall j,k \text{ and } t \tag{8}$$

$$\sum_{i\in J} z_{jkt} = 1 \quad \forall k \text{ and } t \tag{9}$$

Constraints (8) and (9) are not sufficient to model the production sequence on each machine in each period because they allow "cycles". (These cycles are like "subtours" in a traveling salesperson problem.) For instance, in our illustrative example, constraint (8) would allow the following sequence in period $t$:

($P61045.03$, $P61048.01$, $P61048.02$, $P61048.02$, $P61048.01$, $P61045.03$)

For P61045.03, constraint (8) is satisfied by making $z_{P61045.03,t-1} = z_{P61045.03,t} = 1$ (we do not include machine index because there is only one machine). For P61048.01 to be part of the sequence, constraint (8) is satisfied by the following values of the $y$ variables $y_{P61048.02,P61048.01,t} = y_{P61048.01,P61048.02,t} = 1$. The same values satisfy the constraint (8) for P61048.02. These variable values allow the production of parts with two different profiles without charging for the long changeover from the 61045 to the 61048 profile family.

The following set of constraints break cycles and create sequences where a production run of a part appears no more than once in a sequence of a period. The constraints assign a position value to each part being produced on a machine in a period. These constraints are an adaptation of the so-called Miller-Tucker-Zemlin (MTZ) subtour elimination constraints (Miller, Tucker, & Zemlin, 1960).

$$s_{ikt} - s_{jkt} \leq |J|(1 - y_{ijkt}) - 1 \quad \forall i,j,k \text{ and } t \tag{10}$$

When a changeover from part $i$ to part $j$ occurs on machine $k$ in period $t$ (i.e., $y_{ijkt} = 1$), then $s_{ikt} - s_{jkt} \leq -1$. That is the position of $j$ in the production sequence of machine $k$ in period $t$ must be greater than the position of part $i$ by at least one unit. When $y_{ijkt} = 0$, the constraint is nonbinding. The model includes the typical binary (for the $y$ variables) and non-negativity restrictions for all other decision variables.

With the goal of improving upon the lower bounds of the linear programming relaxation of our formulation, we tested an alternative set of subtour elimination constraints. In particular, we implemented the set of constraints proposed by Sarin, Sherali, and Bhootra (SSB) (Sarin, Sherali, & Bhootra, 2005). The SSB constraints are a disaggregation of the MTZ constraints and have been shown to produced tighter bounds in the context of the asymmetric traveling salesmen problem. Unfortunately, the adaptation of the SSB constraints to our problem did not produced the desired effect. Therefore, our computational experiments with the mixed-integer programming formulation were done using the MTZ subtour elimination constraints.

After a few preliminary runs with this model and discussion of the results with key personnel in the company, it was revealed that the optimal solution did not change after optimizing the first two objectives. Furthermore, most parts were "naturally" being assigned to their preferred machines because the production rate on these machines is typically higher. Therefore, a decision was made to drop the third

objective from the model. Additional analysis also showed that there was no advantage in treating the first two objectives hierarchically. Shortages are at least one order of magnitude larger than changeover times, and therefore a straight sum of these two quantities accomplishes the desire to give the minimization of shortages a priority. Therefore, we simplify the objective function to the following expression:

$$\text{Minimize} \sum_{j\in J}\sum_{t\in T} I^{-}_{jt} + \sum_{i\in J}\sum_{j\in J:i\neq j}\sum_{k\in K}\sum_{t\in T} c_{ij}y_{ijkt} \tag{11}$$

Our computational experiments show that the model is most sensitive to an increase in the number of parts and periods. In the environment that we studied, the number of machines is relatively small. However, the number of parts is at least one order of magnitude larger than the number of machines. The number of parts has a quadratic effect on the number of binary variables associated with the production sequencing (i.e., the $y$ variables). This quadratic relationship is exacerbated by any significant increase in the number of periods.

## 3. Metaheuristic approach

Metaheuristics play an important role in practical optimization problems because of their ability to produce high quality solutions in affordable computational times. Given the various constraints in this problem, we have designed a set of algorithmic elements that can be combined to configure an effective search procedure for this problem.

The description of our metaheuristic approach starts with the computational model and is followed by the proposed solution construction methods along with the neighborhoods used to explore the solution space. We also describe an efficient computation of the objective function that we use within the search process.

### 3.1. Computational model

Let $I$ represent the instance that stores the set of problem features described in Section 2. We also define $S = \{S_1, \ldots, S_{|K|}\}$ as a set of $|K|$ lists, where $S_k$ is the workload for machine $k$. The workload is a list of *work slots*, with each work slot being a tuple $w = (j, l, t^i, t^e)$, where $j \in J$ is the index of the part, $t^i$ is the time when the work slot begins, and $t^e$ is the time when the work slot ends. Since $r_{jk}$ is the production ratio of part $j$ on machine $k$, a work slot produces $lr_{jk}$ units of part $j$, where $l = t^e - t^i$ represents the slot duration (i.e., load).

Therefore, following the simplification described in Eq. (11), the computation of the objective function $\mathcal{F}$ for a given solution $S$ is given by the sum of the total *changeover* time and the total *shortage*:

$$\mathcal{F}(S) = changeover(S) + shortage(S) \tag{12}$$

Let $(v, w)$ be a pair of consecutive work slots in machine $k$ such that work slot $v$ produces part $i$ and work slot $w$ produces part $j$, then the total changeover time for solution $S$ is given by:

$$changeover(S) = \sum_{k\in K}\sum_{(v,w)\in S_k} c_{v(i)w(j)} \tag{13}$$

Note that in Eq. (13) we are omitting the $l$ and $t$ elements of the work slot tuples because they do not participate in the calculation of the changeover time. That is, only the indices of the jobs in slots $v$ and $w$ are required to calculate the changeover time. For the sake of clarity, we follow the same logic below, that is, we only include the elements of the work slot tuple that are needed for a given calculation.

The shortage of part $j$ in period $t$ is the difference between the inventory position $d_{jt}$ and the production of part $j$ up to period $t$ in solution $S$, denoted by $production_{jt}(S)$. Recall that the inventory position for each part starts with the number of units in the initial inventory and the position becomes negative when the cumulative demand exceeds the initial inventory. Therefore the shortage for solution $S$ is calculated as follows:

$$shortage(S) = -\sum_{j\in J}\sum_{t\in T} min(0, d_{jt} + production_{jt}(S)) \tag{14}$$

The cumulative production of part $j$ up to period $t$ depends on the work slots scheduled to start prior to period $t$. A work slot could be scheduled on machine $k$ in such a way that $t^i < t$ and $t^e > t$. In this case only the units produced until period $t$, i.e., $(t - t^i)r_{jk}$, count towards the inventory position of part $j$ at period $t$. The cumulative production of part $j$ up to period $t$ is calculated as follows:

$$production_{jt}(S) = \sum_{k\in K}\sum_{\substack{w\in S_k \wedge \\ w(j,t^i)<t}} (min(t, w(j,t^e)) - w(j,t^i))r_{jk} \tag{15}$$

Our procedure does not consider any downtime in addition to the changeover time. That is, if work slot $v$ producing part $i$ immediately precedes work slot $w$ that produces part $j$, then the time $w(j, t^i)$ when work slot $w$ begins depends on the time $v(i, t^e)$ when work slot $v$ ends and the changeover time between part $i$ and part $j$. That is, $w(j, t^i) = v(i, t^e) + c_{ij}$.

### 3.2. Construction methods

The first construction approach consists of generating random solutions. The process generates several work slots of random duration, where the number of slots and their duration is enough to meet the demand. Algorithm 1 shows the pseudo-code for this approach. The procedure starts with an empty solution $S$ and sets the uncover demand $U_j$ for part $j$ as the inventory position in that last period of the planning horizon, i.e., $d_{j|T|}$. The procedure uses $J'$ as the set of parts with uncovered demand and the main loop (line 4) terminates when this set is empty. Within the main loop, a part $j$ is selected at random from $J'$ (line 5). A machine $k$ is randomly selected from $K_j$, which is the set of machines that are able to produce part $j$ (line 6). After a machine is selected, the work slot duration $l$ is randomly chosen between $c^{long}$ and $max(c^{long}, U_j/r_{jk})$. This means that when $U_j/r_{jk} < c^{long}$ then $l = c^{long}$ (line 7). A new work slot $w$ is created in line 8 with the selected part and duration. Note that the $t^i$ and $t^e$ values are determined when the slot is added to the selected machine in line 9. This operation is denoted with the $\oplus$ symbol to indicate that the work slot is added at the end of the corresponding list. The number of uncovered units of demand is updated in line 10, and $j$ is removed from the set of parts with uncovered demand if the uncovered demand is less than or equal to zero (lines 11 and 12). Finally, the solution is returned in line 13.

**Algorithm 1:** *RandomConstruction* ($I$)

```
1  S ← ∅
2  U_j ← −d_{j|T|} , ∀j ∈ J
3  J' ← J
4  while |J'| > 0 do
5      j ← SelectRandom(J')
6      k ← SelectRandom(K_j)
7      l ← SelectRandom(c^long, max(c^long, U_j/r_jk))
8      w ← {j, l}
9      S_k ← S_k ⊕ w
10     U_j ← U_j − l r_jk
11     if U_j ≤ 0 then
12         J' ← J' \ {j}
13 return S
```

Algorithm 1 generates a diverse set of work slots with enough units to cover the demand as long as the search finds a sequence for which the production is completed on time to avoid shortages. We refer to this construction method as $RND$.

In addition to the simple random construction, we have designed a construction method based on the Greedy Randomized Adaptive Search Procedure (GRASP) (Feo & Resende, 1995; Resende & Ribeiro, 2016). GRASP uses a pseudo-greedy strategy to build diverse solutions. Our greedy function measures the contribution to the objective function that

results from assigning a work slot to a machine. Therefore, we need to build a set of work slots with enough capacity to meet demand. The maximum machine time needed to produce all the units demanded of part $j$ is given by:

$$l_j^{max} = -d_{j|T|} / \arg\min_{k \in K_j} r_{jk} \tag{16}$$

This maximum time is obtained by dividing the number of units required in the last period, $-d_{j|T|}$, by the minimum production rate among the machines able to produce $j$. By choosing the lowest rate we guarantee that the set of work slots will be enough to produce the units needed to satisfy the demand. For each part $j$ we create $\lfloor l_j^{max}/c^{long} \rfloor - 1$ work slots with a load $l = c^{long}$ and one work slot with a load $l = c^{long} + l_j^{max} \mod c^{long}$. This is what Algorithm 2 does.

**Algorithm 2:** *GenerateWorkSlots* $(I)$

```
1  W ← ∅
2  for j ∈ J do
3      l_j^max ← −d_{j|T|} / arg min_{k∈K_j} r_jk
4      for l = 1, ..., ⌊l_j^max / c^long⌋ − 1 do
5          l ← c^long
6          w ← {j, l}
7          W ← W ∪ {w}
8      l ← c^long + l_j^max  mod c^long
9      w ← {j, l}
10     W ← W ∪ {w}
11 return W
```

Our construction method uses $W$ as the set of elements that remain to be selected at each step of the process. The pseudocode of the construction method is shown in Algorithm 3. The procedure starts with a call to $GenerateWorkSlots(I)$ that returns $W$ (line 1). The first work slot to be assigned is chosen at random in line 2. The part $j$ associated with the work slot is identified in line 3 and a machine $k$ is randomly selected from the set of machines capable of producing part $j$ in line 4. The work slot $w$ is added to machine $k$ in line 5. The chosen work slot $w$ is removed from the set of work slots in line 6 and the main loop of the method begins, iterating while $W$ is not empty. The main loop starts by creating the so-called candidate list $CL$ in line 8. This list is formed by pairs $(w, k)$ corresponding to the feasible assignments of work slots. That is, each work slot $w$ is associated with all the machines $k$ able to produce the part assigned to the work slot ($k \in K_{w(j)}$). Once the $CL$ is built, a pair $(w, k)$ is selected by either a `GreedyRandom` or a `RandomGreedy` strategy, which we explain below, attending to the value of the $\alpha$ parameter. The selected work slot $w$ is included in the selected machine $k$ of the partial solution $S$ in line 10, and $W$ is updated in line 11. Finally, the solution is returned in line 12.

**Algorithm 3:** *GRASPConstruction* $(I, \alpha)$

```
1  W ← GenerateWorkSlots(I)
2  w ← SelectRandom(W)
3  j ← w(j)
4  k ← SelectRandom(K_j)
5  S_k ← S_k ⊕ w
6  W ← W \ {w}
7  while |W| > 0 do
8      CL ← {(w, k) : w ∈ W ∧ k ∈ K_j}
9      (w, k) ← { GreedyRandom(S, CL, α)
                { RandomGreedy(S, CL, α)
10     S_k ← S_k ⊕ w
11     W ← W \ {w}
12 return S
```

The selection of a $(w, k)$ pair at each step of the construction procedure is done based on two strategies associated with the GRASP methodology, namely, *greedy random* and *random greedy*. Both strategies build a so-called restricted candidate list $RCL$ that is a subset of the elements in $CL$. In *greedy random*, the restricted candidate list $RCL$ is the best (according to the greedy function) $\alpha\%$ elements in $CL$. Then, an element (that is, a $(w, k)$ pair) is chosen at random from $RCL$. In the *random greedy* strategy, the $RCL$ consists of $\alpha\%$ randomly selected elements from $CL$. Then, the best (according to the greedy function) element from $RCL$ is selected.

The greedy function calculates the change in the objective function value of a solution $S$ due to adding work slot $w$ to machine $k$. We define this change as $\Delta(w, k, S)$, as shown in Eq. (17), where $S'_k \leftarrow S_k \oplus w$.

$$\Delta(w, k, S) = \mathcal{F}(S) - \mathcal{F}(S') \tag{17}$$

Since the objective is to minimize $\mathcal{F}(S)$, the larger the value of the greedy function the better. The construction process starts with an empty solution that has a large objective function value equal to the maximum shortage and zero changeovers. Shortage is reduced and changeovers are potentially increased with the sequential addition of work slots. The greedy function, however, maintains a positive value during the construction process because shortage cannot be negative and no additional workslots are added if shortage becomes zero. Algorithm 4 shows our implementation of the *greedy random*. The minimum and maximum values of the greedy function, $\Delta_{min}$ and $\Delta_{max}$, associated with the elements in $CL$ are calculated in lines 1 and 2, respectively. Then, the greedy value threshold for membership in the $RCL$ is found in line 3. This threshold, denoted as $\theta$, depends on the value of the $\alpha \in [0, 1]$ parameter. The restricted candidate list $RCL$ is built in line 4 with those elements whose greedy value is larger or equal to $\theta$. Finally, a pair $(w, k)$ is randomly selected from the $RCL$ in line 5 and returned in line 6. The smaller the value of $\alpha$, the greedier the selection, while the larger the value, the more random. We refer to this construction method as $GR$.

**Algorithm 4:** *GreedyRandom* $(S, CL, \alpha)$

```
1  Δ_min = min_{(w,k)∈CL} Δ(w, k, S)
2  Δ_max = max_{(w,k)∈CL} Δ(w, k, S)
3  θ = Δ_max − α · (Δ_max − Δ_min)
4  RCL ← {(w, k) ∈ CL : Δ(w, k, S) ≥ θ}
5  (w, k) ← SelectRandom(RCL)
6  return (w, k)
```

Algorithm 5 shows the pseudo-code of the *random greedy* strategy. In this case, $\alpha$ determines the size of the *RCL* (line 1), whose elements are randomly selected in line 2. Then, the pair $(w, k)$ with the largest greedy function value is selected in line 3. The interpretation of $\alpha$ is analogous to the one in the *greedy random* strategy in that smaller $\alpha$ values produce a larger $RCL$ and a greedier selection. We refer to this construction method as $RG$.

**Algorithm 5:** *RandomGreedy* $(S, CL, \alpha)$

```
1  size ← max(⌊(1 − α) · |CL|⌋, 1)
2  RCL ← SelectRandomSet(ECL, size)
3  (w, k) ← arg max_{(w',k')∈RCL} Δ(w', k', S)
4  return (w, k)
```

In sum, we propose three methods to construct solutions: the simple random construction ($RND$), the *greedy random* GRASP construction ($GR$), and the *random greedy* GRASP construction ($RG$).

### 3.3. Neighborhood structures

A solution $S$ to the problem is defined as a list of work slot lists, one for each machine (see Section 3.1). A typical search neighborhood

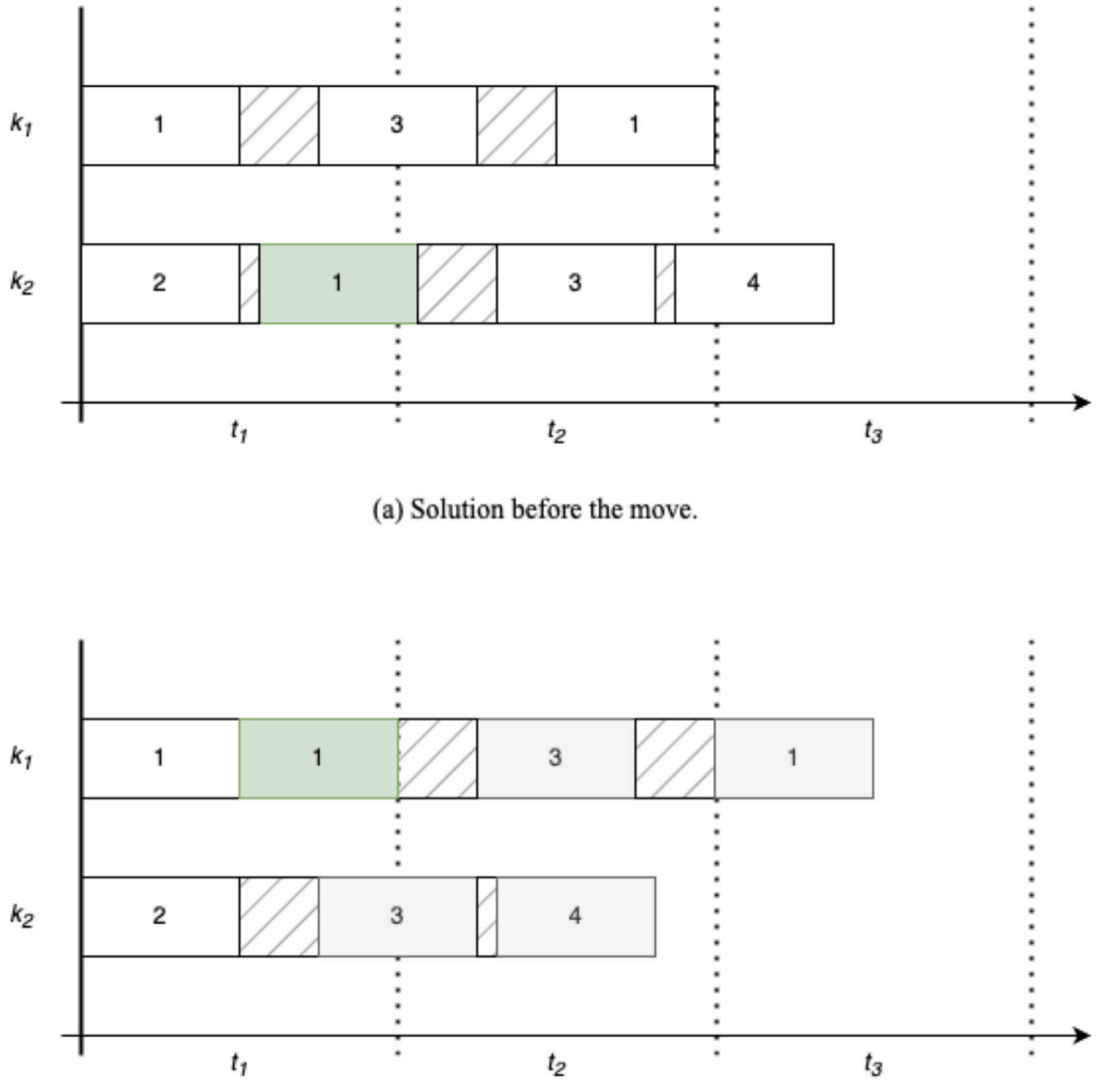


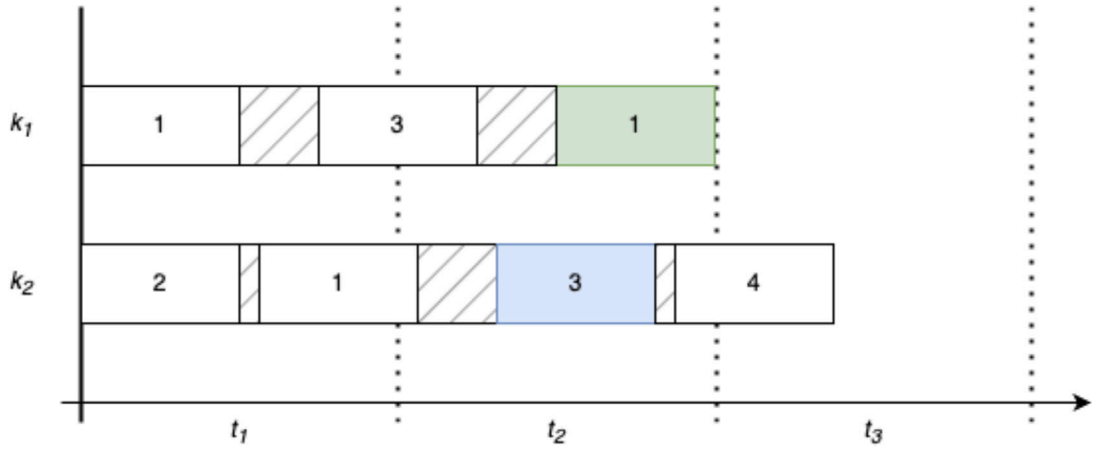

**Fig. 2.** Example of an insert move.

for such a solution representation consists of *insert* and *exchange* moves. These moves can occur both within or across machines. Fig. 2 shows an example with two machines ($k_1$ and $k_2$), four parts (labeled 1 to 4), and three periods ($t_1$ to $t_3$), where the time is represented by the horizontal axis. We assume that parts 1 and 2 belong to one profile family, and parts 3 and 4 belong to another profile family (see Section 1.2). In Fig. 2(a), the list of work slots for machine $k_1$ includes parts 1, 3 and 1. The dashed blocks between each pair of work slots represent the changeover times, which in this case is long because parts 1 and 3 belong to different profile families. The list for $k_2$ includes work slots for parts 2, 1, 3 and 4. In this case the changeover time between 2 and 1 and between 3 and 4 is short because these parts belong to the same profile family.

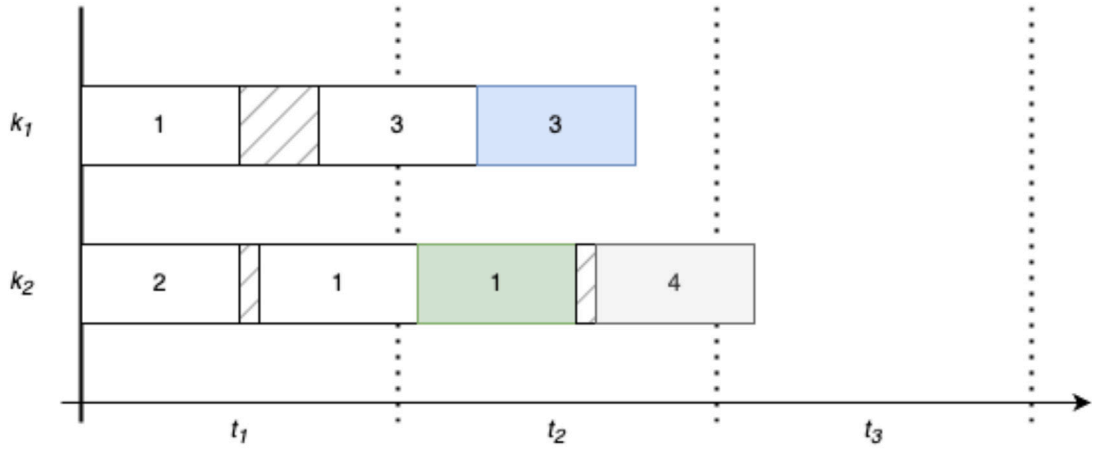
Fig. 2(b) shows the result of inserting the second work slot from machine $k_2$ after the first work slot on machine $k_1$, highlighted with green background. The effect of the move is two-fold. On the one hand, the total changeover time is reduced, since the short changeover between work slots for parts 2 and 1 on machine $k_2$ disappeared, and no changeover has been added on machine $k_1$. On the other hand, the work slots with gray background have changed their starting time. Therefore, the number of units produced in each period changed. In the case of the green work slot of part 1, its entire duration now fits within period $t_1$, however the production rate could be different due to the change of machines. Therefore, the number of units of part 1 completed by the end of period $t_1$ may have changed. Production for part 2 does not change after the move. The entire production of part 3 is still completed by the end of $t_2$ but the number of units produced by the end of $t_1$ might have changed. The production for part 4 is now completed before the end of period $t_2$.

Fig. 3 shows an example of the *exchange* move between the last work slot in machine $k_1$ and the third work slot in machine $k_2$, highlighted in green and blue background, respectively. Again, both the total changeover time and the production units are affected.

For a solution $S$, we define the neighborhood associated with an *insert* move, $\mathcal{N}_{insert}(S)$, as the set of solutions generated after all feasible insertions of work slots. An insertion is feasible if the receiving machine is capable of producing the part associated with the work slot being inserted. Similarly, we define the neighborhood associated with an *exchange* move, $\mathcal{N}_{exchange}(S)$, as the set of solutions generated after all the feasible exchanges of work slots. Here again the machines must be capable of the producing the jobs associated with the work slots that they are receiving.




(a) Solution before the move.




(b) Solution after the move.

**Fig. 3.** Example of an exchange move.

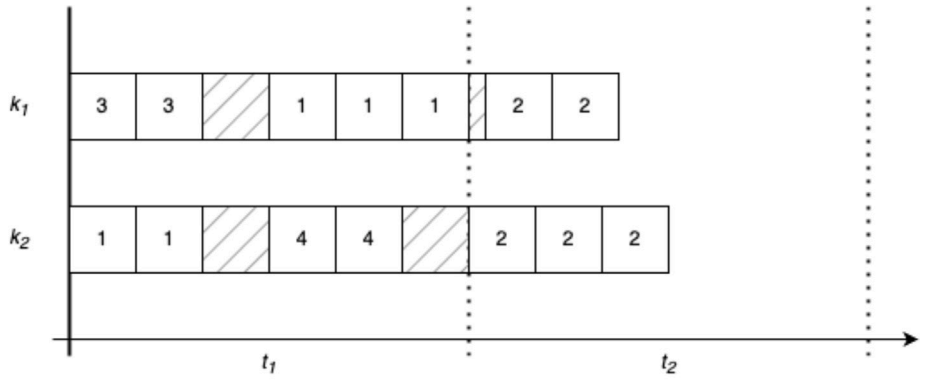
Preliminary experiments designed to test the performance of insertions and exchanges within a solution improvement method showed that insertions alone were sufficient for an efficient exploration of the search space. We could not detect significant benefits of employing both neighborhoods at the same time, given that attractive exchanges were identified by a combination of insertions. In other words, the size of $\mathcal{N}_{exchange}$ is smaller than the size of $\mathcal{N}_{insert}$. Therefore, we chose $\mathcal{N}_{insert}$ as the neighborhood for our *best improvement* local search. We refer to this local search as $LS$.

We also identified situations where the search stagnated due to the way the set of work slots $W$ is created. Consider the situation in Fig. 4(a), in which we assume that there are no shortages. In this situation, there is no insert of a single work slot that can reduce the changeover time. However, by merging individual work slots into larger blocks, as shown in Fig. 4(b), we create opportunities for changeover time reduction. For instance, moving the merged work slot for part 1 on machine $k_2$, shown with a green background, to the second position on machine $k_1$ results in a solution where the total changeover time is reduced, as shown in Fig. 4(c).

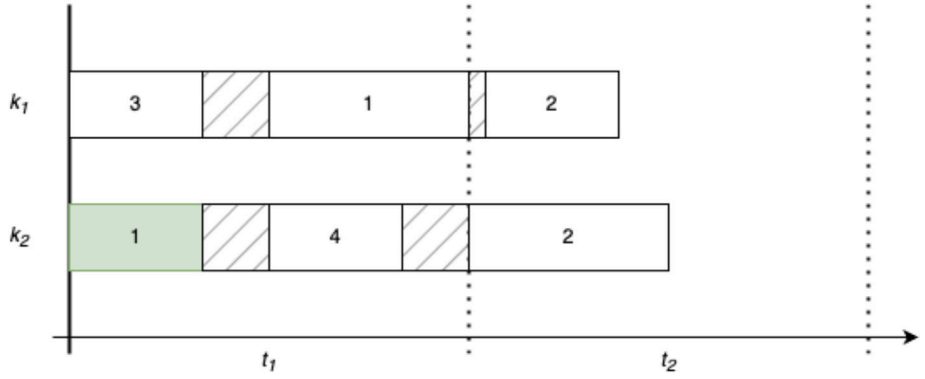
This observation lead us to create a $\mathcal{N}_{insert}$ neighborhood search with merged work slots. We use this neighborhood to perform a merged local search. Once a local optimum is reached, the merged blocks are once again separated into individual work slots to continue the exploration. We refer to this local search as $MLS$.

### 3.4. Search procedures

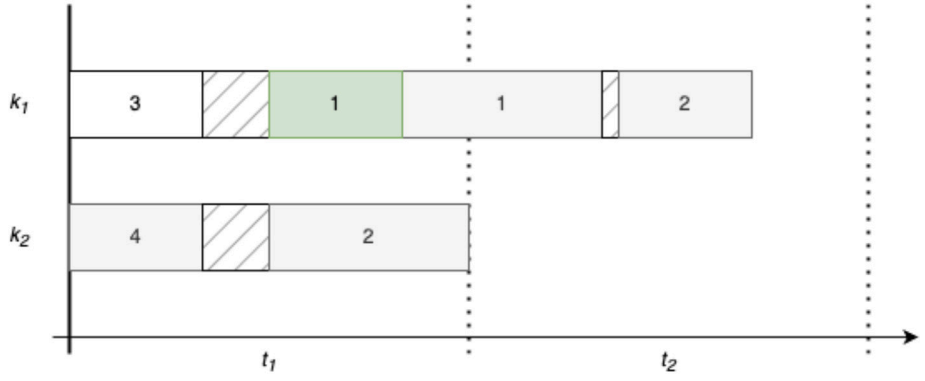
We used the components introduced in the previous sections to configure two full search methods, a Greedy Randomized Adaptive Search Procedure (GRASP) and a Variable Neighborhood Search (VNS). GRASP (Feo & Resende, 1995; Resende & Ribeiro, 2016) is a well-known metaheuristic that has been applied to many different industrial problems (Bamoumen et al., 2023; Kyriakakis et al., 2023). Our GRASP proposal is outlined in Algorithm 6. The input parameters of our GRASP are the problem instance $I$, the value of $\alpha$, and the processing time limit $timeLimit$. The initial number of iterations $i$ is set to zero and the best solution $S^\star$ is set to empty. Then, the main loop consists of constructing (line 4) and improving (line 5) solutions. The best solution is updated every time a new one is found (lines 6 and 7). The procedure returns the best solution found (line 9). Four versions of GRASP are possible using construction methods $GR$ or $RG$ and improvement methods $LS$ or $MLS$. Note that within the GRASP philosophy of constructing solutions




(a) Stagnated solution.




(b) Collapsed solution.




(c) Solution after the move, changeover time reduced.

**Fig. 4.** Collapsing insertion.

in pseudo-greedy fashion, using the $RND$ construction method would not be appropriate.

VNS (Hansen & Mladenović, 2001; Mladenović & Hansen, 1997) is also a well-known metaheuristic that has been applied to many different optimization problems, including the CLSP (Yildiz et al., 2023) and several other industrial problems (Mogale, De, Ghadge, & Tiwari, 2023; Wagner & Mönch, 2023). Our proposal is a multi-start form of VNS, as shown in Algorithm 7. The input parameters are the problem instance $I$, the $\alpha$ value, the processing time limit $timeLimit$, and a VNS-specific parameter $maxShakePercentage$ that controls the maximum level of randomization associated with the `Shake` method. The procedure starts with the initialization of the number of iterations and the best solution. Then, the maximum number of random moves in the `Shake` method is calculated as a fraction of the total number of parts (line 3). According to this calculation, $maxShakePercentage$ must be within $[0, 1]$. A solution is constructed in line 5 and then, the inner VNS loop begins with $shakeMoves = 1$. The `Shake` method is executed to modify the incumbent solution in line 8, obtaining a new solution $S'$.

**Algorithm 6:** *GRASP* $(I, \alpha, timeLimit)$

```
1  i ← 0
2  S* ← ∅
3  while time < timeLimit do
4      S ← Construction(I, α)
5      S' ← Improvement(S)
6      if F(S') < F(S*) then
7          S* ← S'
8      time ← ProcessTime()
9  return S*
```

As stated in Section 3.3, our improvement methods are based on *insert* moves. Therefore, we use the *exchange* move for the `Shake` method. In particular, the method executes $shakeMoves$ random *exchanges*. The resulting solution is improved in line 9 obtaining a new solution $S''$, which is compared with the current best, updating both $S^\star$ and $shakeMoves$ accordingly in lines 10 to 14. The best solution $S^\star$ is returned in line 16 after the multi-start loop ends.

**Algorithm 7:** *VNS* $(I, \alpha, timeLimit, maxShakePercentage)$

```
1  i ← 0
2  S* ← ∅
3  maxShakeMoves ← maxShakePercentage × |J|
4  while time < timeLimit do
5      S ← Construction(I, α)
6      k ← 1
7      while shakeMoves ≤ maxShakeMoves do
8          S' ← Shake(S, shakeMoves)
9          S'' ← Improvement(S')
10         if F(S'') > F(S*) then
11             S* ← S''
12             shakeMoves ← 1
13         else
14             shakeMoves ← shakeMoves + 1
15     time ← ProcessTime()
16 return S*
```

The construction method in VNS could be any of those defined in Section 3.2, namely, $RND$, $GR$, or $RG$. The improving methods could be the local search procedures described in Section 3.3, that is, $LS$ or $MLS$. We configure an additional version by using a variable neighborhood descent (VND) as the `Improvement` method for our VNS. VND combines more than one local search in the same way that VNS loops around various "shake" levels (Hansen, Mladenović, Brimberg, & Pérez, 2019). In our case, we configure two VND processes with $LS$ and $MLS$. In the first configuration ($LS \rightarrow MLS$), the solution to be improved is subjected to $LS$ as long as the solution improves. When the solution stops improving the local search is switched to $MLS$. If an improved solution is found with $MLS$, then the local search switches back to $LS$. This is repeated until no improvement is found while applying $MLS$. The second configuration ($MLS \rightarrow LS$) operates in the same way as the first one with the order of the local searches switched. The three construction methods ($RND$, $GR$, and $RG$) and the two VNDs ($LS \rightarrow MLS$ and $MLS \rightarrow LS$) result in six possible VNS configurations.

## 4. Efficient move value calculations

One of the main drawbacks of the trajectory-based search methods is the potentially large number of evaluations that may be required to explore a neighborhood because search directions are mainly determined by the change in the objective function associated with a move (i.e., the so-called move value). The move value is the change of the objective function associate with a transition from the current solution $S$ to a neighbor solution $S'$. In this section, we describe efficient ways for calculating *insert* and *exchange* move values. In our context, these values are the combined effect of changes in the changeover time and the total shortage.

$$\begin{aligned} moveValue &= \mathcal{F}(S) - \mathcal{F}(S') \\ &= \Delta changeover(S, S') + \Delta shortage(S, S') \end{aligned} \tag{18}$$

In order to simplify the equations associated with the calculation of changeover time, we assume that the index of the parts in the work slots corresponding to two partial workload lists in $S$ are $(..., i-1, i, i+1, \ldots)$ and $(..., j-1, j, j+1, \ldots)$. These workloads may be on either the same machine or two different machines. The difference in changeover time after an insert of the work slot with part $j$ to the position immediately following the work slot with part $i$ is given by:

$$\begin{aligned} \Delta changeover(S, S') =& c_{ij} + c_{j,i+1} + c_{j-1,j+1} \\ & - c_{i,i+1} - c_{j-1,j} - c_{j,j+1} \end{aligned} \tag{19}$$

The partial workload lists corresponding to $S'$ after the insert are $(..., i-1, i, j, i+1, \ldots)$ and $(..., j-1, j+1, \ldots)$. The difference in changeover time after an exchange of the work slot with part $i$ and the work slot with part $j$ is given by:

$$\begin{aligned} \Delta changeover(S, S') =& c_{i-1,j} + c_{j,i+1} + c_{j-1,i} + c_{i,j+1} \\ & - c_{i-1,i} - c_{i,i+1} - c_{j-1,j} - c_{j,j+1} \end{aligned} \tag{20}$$

The partial workload lists corresponding to $S'$ after the insert are $(..., i-1, j, i+1, \ldots)$ and $(..., j-1, i, j+1, \ldots)$. The computational complexity is reduced from quadratic, if we were to use Eq. (13), to linear.

The computation of the change in total shortage is not as straightforward as in the case of the changeover time. As seen in the figures in Section 3.3, the starting time of the work slots with gray background, positioned after those that move, could change. Therefore the move could alter the number of parts that are completed within each period after the starting time of the affected work slots. Our strategy to reduce the time to calculate the change in total shortage consists of keeping track of the production of each part $j$ up to period $t$ on each machine $k$. We denote this value as $production^k_{jt}(S)$. The total production of part $j$ up to period $t$ is given by:

$$production_{jt}(S) = \sum_{k \in K} production^k_{jt}(S) \tag{21}$$

Let us assume that machines $k$ and $k'$ participate in a move that transitions the solution from $S$ to $S'$. Then, the production of part $j$ up to period $t$ in solution $S'$ is equal to the production in solution $S$ plus the production on machines $k$ and $k'$ in solution $S'$ minus the production on those machines in solution $S$:

$$\begin{aligned} production_{jt}(S') =& production_{jt}(S) \\ & + production^k_{jt}(S') + production^{k'}_{jt}(S') \\ & - production^k_{jt}(S) - production^{k'}_{jt}(S) \end{aligned} \tag{22}$$

The shortage is calculated for the $(j, t)$ pairs for which $production_{jt}(S') \neq production_{jt}(S)$.

The GRASP construction methods described in Section 3.2 calculate the contribution of a candidate work slot by evaluating the objective function after the work slot is added to the candidate position. Considering that these operations are equivalent to making *insert* moves, we use the strategies described here to expedite the calculations while constructing solutions.

## 5. Hexaly optimizer model

To the best of our knowledge, the identical form of the CLSP proposed in this paper cannot be found in the literature. In addition to the mathematical model proposed in Section 2 and the metaheuristic

proposals described in Section 3, we developed a *Hexaly* model with the goal of establishing credible benchmarks. *Hexaly*[1] (formerly *LocalSolver*) has a established reputation in industry and has been used as a way of assessing performance of proposed procedures when exact approaches such as mathematical programming fail (Benoist, Estellon, Gardi, Megel, & Nouioua, 2011; Wang, Liu, Peng, Wang, & Punnen, 2023; Yarkoni, Raponi, Bäck, & Schmitt, 2022). To take advantage of the general-purpose heuristic nature of *Hexaly*, we created the model by using the same structure developed in Section 3.1. Therefore, the main decision variables in our model are ordered lists of work slots, one for each machine. The shortage and the changeover time are evaluated using the equations that we developed for the heuristic procedure (i.e., Eqs. (12), (13), and (14)).

The model does not impose an explicit limit on the number of work slots assigned to a machine. Given a list of work slots, the capacity of the machine determines the number of units produced of each part in each period. If the end of a given period happens in the middle of a work slot, our model calculates the number of units that belong to the current period and those that belong to the next period. If the list of work slots is such that a fraction of a work slot or even entire work slots fall outside the last period in the planning horizon, then those units do not contribute to the total production of parts. To reduce the search space and avoid solutions that are clearly inferior, we add a constraint to the *Hexaly* model to limit the total load assigned to each machine.

$$\sum_{w \in S_k} w(l) \leq \sum_{t \in T} q_{kt} + \max_{w \in W}(w(l)), \forall k \in K \tag{23}$$

Constraint (23) restricts the total load assigned to each machine to the available amount of production time plus the maximum load across all work slots. The second term in the right-hand-side of the constraint enables the assignment of a work slot that can partially contribute to the total production. That is a work slot that starts within the last period of the planning horizon but finished past the end of the period.

## 6. Experimental results

For the experiments reported in this section, the mathematical model was programmed in Python 3.9 and executed using Gurobi 10.0.3. The *Hexaly* model has been executed using the latest version available at the time of the writing, Hexaly Optimizer 12.0 (build 20231108), and the metaheuristic methods were coded in Java 20 using the MORK framework[2] (Martín-Santamaría, López-Ibáñez, Stützle, & Colmenar, 2024). All the experiments were executed in a Ubuntu virtual machine with 32 cores on a cluster using AMD EPYC 7643 CPUs.

### 6.1. Benchmark instances

Our instances have been generated from real data provided by a company with which we collaborated. We omit company details for contractual reasons. Basing our instances on real data provides a better testing platform than randomly generated instances because the data that the company shared with us captures actual demand patterns and technological differences in production equipment.

The data consists of 7 machines, 103 products, and 12 periods. From these data, we randomly generated a set of 20 instances with $|K| = \{2, 4, 6\}$ and for $|T| = \{6, 8, 12\}$. Table 3 shows the instance id, number of machines ($|K|$), number of parts ($|J|$), number of periods ($|T|$), total shortage (*Shortage*), and average production rate (*Rate*). The total shortage is the sum of the inventory positions for all parts in the last period of the planning horizon, i.e., $Shortage = \sum_{j \in J} d_{j|T|}$. The average rate is calculate for all parts and machines, i.e., $Rate = (\sum_{j \in J} \sum_{k \in K} r_{jk})/(|J||K|)$.

[1] https://www.hexaly.com/.

[2] https://github.com/mork-optimization/mork.

**Table 3**
Test set of problem instances.

| Instance | $\|K\|$ | $\|J\|$ | $\|T\|$ | Shortage | Rate |
|---|---|---|---|---|---|
| CLM-01 | 2 | 25 | 6 | −250 110 | 368 |
| CLM-02 | 2 | 30 | 6 | −401 058 | 465 |
| **CLM-03** | 2 | 41 | 8 | −597 080 | 390 |
| **CLM-04** | 2 | 43 | 6 | −533 195 | 297 |
| CLM-05 | 2 | 49 | 8 | −831 972 | 490 |
| CLM-06 | 2 | 50 | 12 | −1 363 816 | 399 |
| **CLM-07** | 2 | 58 | 12 | −1 749 742 | 493 |
| CLM-08 | 2 | 62 | 8 | −1 054 148 | 316 |
| CLM-09 | 2 | 71 | 12 | −2 140 254 | 313 |
| CLM-10 | 4 | 41 | 6 | −560 228 | 139 |
| **CLM-11** | 4 | 60 | 8 | −1 092 179 | 139 |
| **CLM-12** | 4 | 69 | 8 | −1 188 313 | 180 |
| CLM-13 | 4 | 77 | 12 | −2 212 387 | 136 |
| CLM-14 | 4 | 85 | 12 | −2 472 100 | 179 |
| CLM-15 | 6 | 52 | 6 | −599 961 | 200 |
| CLM-16 | 6 | 53 | 6 | −627 337 | 193 |
| CLM-17 | 6 | 77 | 8 | −1 228 679 | 206 |
| CLM-18 | 6 | 81 | 8 | −1 298 756 | 197 |
| **CLM-19** | 6 | 91 | 12 | −2 614 122 | 209 |
| CLM-20 | 6 | 99 | 12 | −2 764 574 | 193 |

The heuristic strategies described in Section 3 include parameters that required proper adjustment. To this end, we selected a representative subset of the instances to engage in parameter tuning and algorithmic configuration. We selected the subset using the method described in Martín-Santamaría, Cavero, Herrán, Duarte, and Colmenar (2023) with the problem features in Table 3. The process resulted in the six instances highlighted in bold in Table 3 as the representative subset. We refer to these instances as our training set.

### 6.2. Efficiency of move value calculations

Our first experiment assesses the benefits of the move value calculations described in Section 4. We ran three algorithm components 10 times using the training set while evaluating moves with full objective function calculations and with the equations in Section 4. We wanted to measure the effect on the construction of solutions, the improvement method, as well as the full GRASP iterations (i.e., construction plus improvement). For the construction, we used $GR$ (i.e., the *greedy random* construction) with a random $\alpha$ value. For the improvement method we coupled $RND$ (i.e., a random construction) and $LS$ (i.e., the local search based on a first improvement strategy and insert moves). For the full GRASP iterations, we combined $GR$ and $LS$.

Table 4 shows the total execution time in seconds of the runs with the full objective function calculation ($Full$), the partial calculation based on the equations in Section 4 ($Partial$), and the improvement given as a percentage reduction in computational time ($Imp$).

As indicated in Table 4, the average improvement in computational time is of at least 95% across the three search elements. There is a clear advantage in using the equations developed in Section 4 for the calculation of the objective function when adding elements during the construction of a solution or while searching neighborhoods in an improvement method.

### 6.3. Algorithmic configuration

The algorithmic components described in Section 3 can be combined to generate several configurations. In particular, our construction and improvement methods can be combined to configure GRASP and VNS methods. Instead of using trial-and-error or full-factorial with a limited number of parameter values we use `irace`, an automated algorithm configurator based on iterated racing (López-Ibáñez, Dubois-Lacoste, Cáceres, Birattari, & Stützle, 2016). The input to `irace` consists of a set of categorical or numerical parameters (or design components) and

**Table 4**
Comparison of execution time, in seconds, between the full and partial computation of the objective function.

| Instance | *GR* | | | *RND + LS* | | | *GR + LS* | | |
|---|---|---|---|---|---|---|---|---|---|
| | Full | Partial | Imp | Full | Partial | Imp | Full | Partial | Imp |
| CLM-03 | 26.20 | 3.89 | 85.15% | 210.41 | 7.23 | 96.57% | 208.68 | 7.13 | 96.58% |
| CLM-04 | 4.46 | 0.16 | 96.45% | 149.56 | 4.43 | 97.04% | 199.30 | 5.66 | 97.16% |
| CLM-07 | 101.35 | 2.79 | 97.25% | 4 368.88 | 87.00 | 98.01% | 15 277.71 | 314.35 | 97.94% |
| CLM-11 | 72.70 | 1.63 | 97.76% | 1 792.80 | 26.89 | 98.50% | 3 760.13 | 50.57 | 98.66% |
| CLM-12 | 72.95 | 1.90 | 97.40% | 3 290.24 | 42.20 | 98.72% | 5 452.84 | 84.40 | 98.45% |
| CLM-19 | 2525.42 | 38.53 | 98.47% | 51 620.31 | 538.88 | 98.96% | 185 637.22 | 2705.69 | 98.54% |
| **Avg.** | 467.18 | 8.15 | 95.41% | 10 238.70 | 117.77 | 97.96% | 35 089.31 | 527.97 | 97.89% |

**Table 5**
Overview of the parameter configuration for `irace`.

| Parameter | Range/Values | Dependency |
|---|---|---|
| *algorithm* | [*GRASP, VNS*] | *none* |
| *construction* | [*GR, RG*] | If *algorithm* is *GRASP* |
| | [*RND, GR, RG*] | If *algorithm* is *VNS* |
| $\alpha$ | [*random, value*] | For *GR* and *RG* |
| *value* | [0.00, 1.00] | If $\alpha$ is *value* |
| *maxShakePercentage* | [0.00, 0.50] | If *algorithm* is *VNS* |
| *improvement* | [*LS, MLS, VND*] | If *algorithm* is *GRASP* |
| *VND* | [*LS* → *MLS, MLS* → *LS*] | If *improvement* is *VND* |

**Table 6**
Performance comparison in the competitive testing.

| Instance | *Gurobi* | | *Hexaly* | | *VNS* | |
|---|---|---|---|---|---|---|
| | Obj. Fun. | Dev | Obj. Fun. | Dev | Obj. Fun. | Dev |
| CLM-01 | **132** | 0.00% | **132** | 0.00% | **132** | 0.00% |
| CLM-02 | **199** | 0.00% | **199** | 0.00% | **199** | 0.00% |
| CLM-03 | **194** | 0.00% | **194** | 0.00% | **194** | 0.00% |
| CLM-04 | 94 692 | 0.64% | **94 090** | 0.00% | 94 176 | 0.09% |
| CLM-05 | 27 631 | 20.94% | **21 846** | 0.00% | 22 535 | 3.06% |
| CLM-06 | 47 358 | 23.59% | 38 551 | 6.13% | **36 188** | 0.00% |
| CLM-07 | 1 070 902 | 2.64% | 1 077 488 | 3.23% | **1 042 675** | 0.00% |
| CLM-08 | 574 992 | 0.45% | **572 390** | 0.00% | 578 255 | 1.01% |
| CLM-09 | 3 305 131 | 1.50% | **3 255 513** | 0.00% | 3 274 621 | 0.58% |
| CLM-10 | **202** | 0.00% | **202** | 0.00% | **202** | 0.00% |
| CLM-11 | 347 | 8.36% | **318** | 0.00% | **318** | 0.00% |
| CLM-12 | 8 631 | 14.33% | 7 550 | 2.07% | **7 394** | 0.00% |
| CLM-13 | **335 634** | 0.00% | 353 290 | 5.00% | 359 791 | 6.71% |
| CLM-14 | 445 667 | 5.35% | 444 795 | 5.16% | **421 822** | 0.00% |
| CLM-15 | **257** | 0.00% | 264 | 2.65% | **257** | 0.00% |
| CLM-16 | **267** | 0.00% | 274 | 2.55% | **267** | 0.00% |
| CLM-17 | 404 | 1.49% | 422 | 5.69% | **398** | 0.00% |
| CLM-18 | 427 | 3.98% | 445 | 7.87% | **410** | 0.00% |
| CLM-19 | 660 | 9.24% | 641 | 6.55% | **599** | 0.00% |
| CLM-20 | 780 | 20.26% | 623 | 0.16% | **622** | 0.00% |
| Average | 295 725.35 | 5.64% | 293 461.35 | 2.35% | 292 052.75 | 0.57% |
| # Best | 7 | | 9 | | 15 | |

their relationships and dependencies. The system searches for the best configuration according to specified performance measure (e.g., the average objective function value of a set of problem instances). We used the previously defined training set instances for our algorithmic configuration with `irace`.

Table 5 shows the parameters that `irace` explored. For categorical parameters, the table shows the list of values. For numerical parameters, the table shows the range of values. The *Dependency* column shows any connections with other decisions. For example, the set of construction methods depends on the choice algorithm. The improvement method for VNS is *VND*, which can be executed in two different ways.

The maximum number of experiments for `irace` was set to 10,000 and all other parameters were set to their default values. The three best configurations obtained are very consistent. The three suggest *VNS* as the main algorithm, *RG* was chosen as the construction method, and *MLS* → *LS* is suggested as the better *VND* configuration. The values for $\alpha$ and for $maxShakePercentage$ are within a very narrow range around 0.4.

The results of running `irace` confirmed the robustness of the proposal. The outcome gave us a high degree of confidence in selecting the best configuration for our competitive testing. The configuration is the *VNS* procedure with *random greedy* constructions, and $\alpha$ value of 0.4, $maxShakePercentage = 0.44$, and a *MLS* → *LS* strategy for *VND*.

### 6.4. Computational results

Our competitive testing consists of comparing our *VNS* with the mathematical programming approach (referred to as *Gurobi*) and the *Hexaly* solutions. We established an execution limit of 3600 s for all procedures. For *VNS* the time was divided in 30 runs with a time limit of 120 s each. Table 6 shows the results of this experiment. For each algorithm we show the value of the objective function (*Obj. Fun.*) and the deviation calculated against the best known solution (shown in bold in the objective function columns). The last two rows of the table show the average objective function values, the average deviation, and the number of times each procedure produced the best known solution.

The mathematical model reaches the best objective function value in 7 out of the 20 instances. The *Hexaly* model produces 9 out of the 20 and our *VNS* 15 out of the 20. Of the three approaches, the solution of the MIP with *Gurobi* seems to be the least reliable. While in most of the instances it is able to produce solutions within 10% of the best known, it can also produce results that are more than 20% away from the best. We could not find a pattern to determine the reasons for *Gurobi* to struggle with some problem instances. Our *Hexaly* model is a very good option for tackling the instances in our test set. All the solutions that the *Hexaly* optimizer found are within 8% of the best. The best option according to these results is the proposed VNS. Only in one instance (CLM-13) it failed to produce a solution within 5% of the best known. We point out that none of the best known solutions in Table 6 are confirmed optima. We do not show optimality gaps because the MIP formulation does not have a strong relaxation and therefore *Gurobi's* branch-and-bound process produces weak lower bounds.

Throughout our experimentation, we have verified that *VNS* tends to find its final solution (i.e., the solution that is reported as best for a given run) earlier in the search than either *Gurobi* or *Hexaly*. Since the execution of the *VNS* proposal consists of 30 independent runs, we show statistics associated with these runs in Table 7. For each instance, the average objective function value is shown in column *Avg. OF*. The number of times that the best value is reached is shown in column *# Best* and the average deviation to the best value is shown in column *% Dev*. The last column (*TTB (s)*) shows the average time in seconds to reach the best solution.

On average, individual runs produce solutions close to the best known values, with the exception of CLM-04 and CLM-05. The results in Table 7 show the importance of running the VNS procedure multiple times. The sampling process that results from multiple runs produce the desire results, as shown in Table 6. The average time to reaching the best solution in each run seems to indicate that, given a fixed computational budget, it might be worthwhile to experiment with an

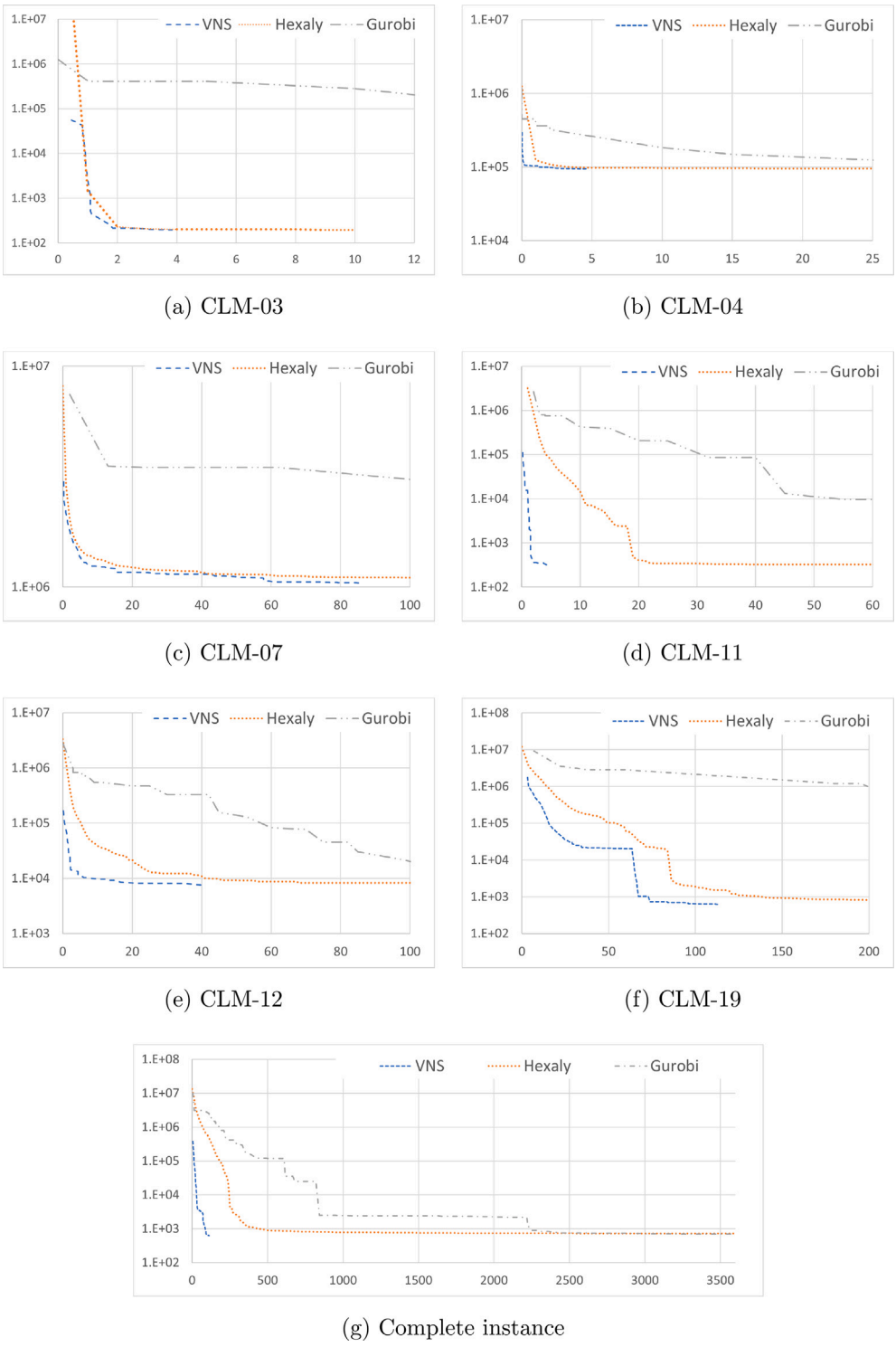



**Fig. 5.** Comparison of convergence speed. *X*-axis represents elapsed time in seconds. *Y*-axis contains the corresponding objective function value at any given time, using a logarithmic scale.

increased number of runs that are shorter (e.g., 60 runs of 60 s instead of 30 runs of 120 s).

In addition to the 20 instances in Table 3, we executed the three solution methods on the complete instance provided by the company. As mentioned above, this instance consists of 7 machines, 103 products and 12 periods. The runs with a limit of one hour resulted in objective function values of 720, 697, and 628 for *Hexaly*, *Gurobi*, and *VNS*, respectively. The three methods found a solution without shortages. VNS produced changeover-time savings of 12.8% and 9.9% over the solutions found with the *Hexaly* and *Gurobi* models, respectively.

A final view of the output data from our experiments confirms the notion that the proposed VNS performs a more aggressive search than the alternatives that we used for comparison. Fig. 5 shows the evolution of the objective function value associated with the incumbent solution for each procedure. For this figure, we use the 6 representative instances and runs with time limits ranging from 12 s (for CLM-03) to 200 (for CLM-19). We also show the evolution of the best objective

**Table 7**
Details of VNS runs.

| Instance | Avg. of | # Best | Dev | TTB (s) |
|---|---|---|---|---|
| CLM-01 | 132.23 | 29 | 0.18% | 0.52 |
| CLM-02 | 202.90 | 5 | 1.96% | 0.85 |
| CLM-03 | 197.50 | 18 | 1.80% | 1.06 |
| CLM-04 | 97 032.33 | 0 | 3.03% | 3.48 |
| CLM-05 | 26 157.03 | 0 | 16.07% | 8.90 |
| CLM-06 | 43 899.00 | 1 | 21.31% | 28.61 |
| CLM-07 | 1 074 679.00 | 1 | 3.07% | 80.93 |
| CLM-08 | 586 627.13 | 0 | 1.45% | 16.89 |
| CLM-09 | 3 311 037.47 | 0 | 1.11% | 115.70 |
| CLM-10 | 204.57 | 19 | 1.27% | 0.49 |
| CLM-11 | 333.53 | 1 | 4.88% | 9.18 |
| CLM-12 | 7 950.77 | 1 | 7.53% | 15.26 |
| CLM-13 | 383 049.90 | 0 | 6.46% | 106.84 |
| CLM-14 | 463 070.53 | 1 | 9.78% | 116.79 |
| CLM-15 | 258.63 | 23 | 0.64% | 1.18 |
| CLM-16 | 270.27 | 16 | 1.22% | 1.08 |
| CLM-17 | 403.73 | 11 | 1.44% | 21.92 |
| CLM-18 | 417.33 | 9 | 1.79% | 21.86 |
| CLM-19 | 637.80 | 1 | 6.48% | 114.65 |
| CLM-20 | 660.57 | 1 | 6.20% | 114.47 |
| Avg. | 299 861.11 | | 4.88% | 39.03 |

function value of the full real-world instance on 3600-s run. We observe that *VNS* reaches its best value faster than *Hexaly* and *Gurobi*. Not surprisingly, *Gurobi's* evolution is the slowest, although this could be slightly improved by manipulating some of the software search parameters (i.e., MIPFocus). The *Hexaly* reaches a good compromise between the other two alternatives and emerges as a viable way of solving the CLSP in practice. Note that we have used logarithmic scale for the objective function value in some of the instances to be able to show the differences among the three approaches.

## 7. Conclusions and future work

Solving real-world problems is an important activity in the operations research community. When the optimization problems are associated with efficiency goals, such a minimizing waste or maximizing production time, companies look for quantifiable benefits while considering the cost of the solution. Commercial optimization software is appealing from the point of view of a potentially short development time, which focuses on creating a suitable model. However, optimization software for mathematical programming models has some limitations if the model becomes too complex or large. Software such as the *Hexaly Optimizer* is an attractive alternative because of its modeling flexibility (e.g., models are not restricted to the framework of mathematical programming) and the combination of techniques (e.g., exact and heuristic) embedded in its solution method.

The development of specialized solution methods typically requires a greater level of commitment and effort. The advantage is that, for all practical purposes, there are no restrictions on the problem features that could be included in a custom procedure. In this paper we described the process that we followed to arrive to solutions that were satisfactory to the company in which we conducted this project. We believe that additional enhancements are possible to improve the performance of the heuristic search that we proposed. There is also the possibility of approaching the problem in an entirely different way within the framework of a matheuristic. For instance, it might be interesting to explore a decomposition approach based on separating the work assignment and lot-sizing decisions from the sequencing decisions. A reduced mixed-integer linear programming could be solved to optimize the size and assignment of production lots to machines. This could be followed by the solution of a set of restricted quadratic assignment problems to optimize the sequencing of production runs on each machine in order to reduced changeover times.

## CRediT authorship contribution statement

**J. Manuel Colmenar:** Writing – review & editing, Writing – original draft, Validation, Supervision, Software, Methodology, Investigation, Funding acquisition, Data curation. **Manuel Laguna:** Writing – review & editing, Writing – original draft, Validation, Supervision, Methodology, Formal analysis, Data curation, Conceptualization. **Raúl Martín-Santamaría:** Writing – original draft, Software, Investigation.

## Acknowledgments

This work has been partially supported by the Spanish Ministerio de Ciencia e Innovación (MCIN/AEI/10.13039/501100011033/FEDER, UE) under grant refs. PID2021-126605NB-I00 and RED2022-134480-T, and by ERDF A way of making Europe.

## Data availability

Instances and source code have been published in Zenodo: https://doi.org/10.5281/zenodo.13912006.

## References

Bamoumen, M., Elfirdoussi, S., Ren, L., & Tchernev, N. (2023). An efficient GRASP-like algorithm for the multi-product straight pipeline scheduling problem. *Computers & Operations Research*, *150*, Article 106082.

Benoist, T., Estellon, B., Gardi, F., Megel, R., & Nouioua, K. (2011). Localsolver 1. x: a black-box local-search solver for 0-1 programming. *4or*, *9*(3), 299–316.

De Armas, J., & Laguna, M. (2020). Parallel machine, capacitated lot-sizing and scheduling for the pipe-insulation industry. *International Journal of Production Research*, *58*(3), 800–817.

Edis, E. B., Oguz, C., & Ozkarahan, I. (2013). Parallel machine scheduling with additional resources: Notation, classification, models and solution methods. *European Journal of Operational Research*, *230*(3), 449–463.

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

Hansen, P., & Mladenović, N. (2001). Variable neighborhood search: Principles and applications. *European Journal of Operational Research*, *130*(3), 449–467.

Hansen, P., Mladenović, N., Brimberg, J., & Pérez, J. A. M. (2019). Variable neighborhood search. In *Handbook of metaheuristics* (pp. 57–97). Springer.

Kang, S., Malik, K., & Thomas, L. J. (1999). Lotsizing and scheduling on parallel machines with sequence-dependent setup costs. *Management Science*, *45*(2), 273–289.

Koch, C., Arbaoui, T., Ouazene, Y., Yalaoui, F., Brunier, H. D., Jaunet, N., et al. (2022). A matheuristic approach for solving a simultaneous lot sizing and scheduling problem with client prioritization in tire industry. *Computers & Industrial Engineering*, *165*, Article 107932.

Kyriakakis, N. A., Aronis, S., Marinaki, M., & Marinakis, Y. (2023). A GRASP/VND algorithm for the energy minimizing drone routing problem with pickups and deliveries. *Computers & Industrial Engineering*, *182*, Article 109340.

López-Ibáñez, M., Dubois-Lacoste, J., Cáceres, L. P., Birattari, M., & Stützle, T. (2016). The irace package: Iterated racing for automatic algorithm configuration. *Operations Research Perspectives*, *3*, 43–58.

Martín-Santamaría, R., Cavero, S., Herrán, A., Duarte, A., & Colmenar, J. M. (2023). A practical methodology for reproducible experimentation: An application to the double-row facility layout problem. *Evolutionary Computation*, 1–36.

Martín-Santamaría, R., López-Ibáñez, M., Stützle, T., & Colmenar, J. M. (2024). On the automatic generation of metaheuristic algorithms for combinatorial optimization problems. *European Journal of Operational Research*, *318*(3), 740–751.

Miller, C., Tucker, A., & Zemlin, R. (1960). Integer programming formulations and travelling salesman problems. *Journal of the ACM*, *7*, 326–329.

Mladenović, N., & Hansen, P. (1997). Variable neighborhood search. *Computers & Operations Research*, *24*(11), 1097–1100.

Mogale, D., De, A., Ghadge, A., & Tiwari, M. K. (2023). Designing a sustainable freight transportation network with cross-docks. *International Journal of Production Research*, *61*(5), 1455–1478.

Özpeynirci, S., Gökgür, B., & Hnich, B. (2016). Parallel machine scheduling with tool loading. *Applied Mathematical Modelling*, *40*(9–10), 5660–5671.

Resende, M., & Ribeiro, C. (2016). *Optimization by GRASP: Greedy randomized adaptive search procedures*. New York: Springer.

Roychowdhury, S., Allen, T. T., & Allen, N. B. (2017). A genetic algorithm with an earliest due date encoding for scheduling automotive stamping operations. *Computers & Industrial Engineering*, *105*, 201–209.

Sarin, S. C., Sherali, H. D., & Bhootra, A. (2005). New tighter polynomial length formulations for the asymmetric traveling salesman problem with and without precedence constraints. *Operations Research Letters*, *33*(1), 62–70.

Velez, S., Dong, Y., & Maravelias, C. T. (2017). Changeover formulations for discrete-time mixed-integer programming scheduling models. *European Journal of Operational Research*, *260*(3), 949–963.

Wagner, S., & Mönch, L. (2023). A variable neighborhood search approach to solve the order batching problem with heterogeneous pick devices. *European Journal of Operational Research*, *304*(2), 461–475.

Wang, Y., Liu, H., Peng, B., Wang, H., & Punnen, A. P. (2023). A three-phase matheuristic algorithm for the multi-day task assignment problem. *Computers & Operations Research*, *159*, Article 106313.

Xiao, J., Zhang, C., Zheng, L., & Gupta, J. N. (2013). MIP-based fix-and-optimise algorithms for the parallel machine capacitated lot-sizing and scheduling problem. *International Journal of Production Research*, *51*(16), 5011–5028.

Yarkoni, S., Raponi, E., Bäck, T., & Schmitt, S. (2022). Quantum annealing for industry applications: Introduction and review. *Reports on Progress in Physics*.

Yildiz, S. T., Ozcan, S., & Cevik, N. (2023). Variable neighborhood search-based algorithms for the parallel machine capacitated lot-sizing and scheduling problem. *Journal of Engineering Research*, Article 100145.