


# Production scheduling with stock- and staff-related restrictions

Carlo S. Sartori , Vinicius Gandra, Hatice Çalık, and Pieter Smet

KU Leuven, Department of Computer Science, CODES; Leuven.AI, Belgium  
{carlo.sartori,vinicius.gandramartinssantos,hatice.calik,pieter.smet}@kuleuven.be

**Abstract.** Effective production scheduling allows manufacturing companies to be flexible and well-adjusted to varying customer demand. In practice, production scheduling decisions are subject to several complex constraints which emerge from staff working hours and skills, delivery schedules, stock capacities, machine maintenance and machine setup. This paper introduces a novel production scheduling problem based on the real-world case of a manufacturing company in Belgium. Given a set of customer requests which may only be delivered together on one of the provided potential shipment days, the problem is to select a subset of these requests and schedule the production of the required item quantities subject to the aforementioned restrictions. All decisions must be taken for a time horizon of several days, leading to a complex problem where there may not be enough resources to serve all requests. We provide an integer programming formulation of this novel problem which is capable of solving small-scale instances to proven optimality. In order to efficiently solve large-scale instances, we develop a metaheuristic algorithm. A computational study with instances generated from real-world data indicates that the metaheuristic can quickly produce high-quality solutions, even for cases comprising several days, requests and limited stock capacities. We also conduct a sensitivity analysis concerning characteristics of the schedules and instances, the results of which can be exploited to increase production capacity and revenue.

**Keywords:** Production scheduling · Stock levels · Integer programming · Metaheuristic.

## 1 Introduction

Due to increased global competitiveness and market uncertainty, manufacturing companies have become increasingly flexible to meet varying demand for their products. Typically, production lines have finite capacities and demand cannot be met by simply increasing production rates. Instead, careful lot-sizing decisions must be made to determine how much of each product is produced and how much stock is maintained. In practice, these decisions are subject to a variety of constraints, including staff-related restrictions and delivery schedules.

Our work is motivated by a problem faced by a manufacturing company in Belgium. The production environment is equipped with two non-identical

machines capable of producing multiple item types under the supervision of human operators. Each machine can only produce one item type at a time given that the necessary configuration (setup) for an item type is conducted before production begins. The item types and quantities demanded by a set of customers are fixed, but the date to deliver the entire demand of each customer must be selected from multiple options provided by the customer. If resources are not sufficient to satisfy all customer demands, the lost sales are reflected as a penalty cost. Machines require maintenance at regular intervals. Maintenance and certain setups can only be performed by a skilled operator who is available during a specific time slot each day. In order to increase production capacity, the two machines can operate in parallel. They can also operate with overtime or during night shifts, but each of these options incurs additional costs. Another type of cost is incurred when the daily safety stocks are not maintained, meaning that the stock level of an item type drops below a minimum desired threshold. Moreover, a predetermined maximum stock level may not be exceeded under any circumstances for any item type.

Given all these machine-, operator- and stock-related restrictions, a solution for the problem involves generating a production schedule at minimum cost for a finite period of time, which comprises a number of days subdivided into several time blocks. We refer to this problem as the *Production Scheduling Problem with stock- and staff-related restrictions* (PSP). Before providing a detailed description of the PSP in Section 2, we will briefly review some related problems to position the PSP in the literature.

The literature most closely related to the PSP is that of *Lot-sizing and Scheduling Problems* (LSPs): a class of problems with many variants for the planning of production schedules. A review concerning standard models proposed for several LSPs was provided by [3]. Additionally, due to its numerous side constraints, the PSP can also be positioned within the literature of LSPs with *Secondary Resources* (SRs) [12]. SRs are subcategorized as one of two types: disjunctive or cumulative. A disjunctive SR can only be employed once at a time, whereas a cumulative SR is one for which the total accumulated value restricts the solution in some way. The PSP contains SRs of both types. Similar to the study conducted by [11], certain operations can only be performed in the presence of a skilled operator. This operator is a disjunctive SR who cannot perform multiple duties in parallel. Meanwhile, daily stock capacities and the stocks themselves are cumulative SRs since although they constrain the solutions to the problem, they can be used to satisfy multiple requests.

More specifically, the PSP can be considered a variant of the *Discrete Lot-sizing and Scheduling Problem (DLSP)* [5]. This is one of the classic LSPs as per the classification provided by [3]. There are two characteristics that distinguish the DLSP from other LSPs [4]. First, it considers both macro and micro periods where macro periods are formed by a sequence of micro periods. Other LSPs only consider the macro scale. Second, the DLSP assumes all-or-nothing production: only one item type may be produced by a machine during a micro period while running at full capacity. In the PSP, we have both macro and micro periods:

days and blocks. Furthermore, in each block, a machine is either idle or entirely dedicated to a single type of operation (maintenance, setup or production of one item type). One key difference, however, is that while the DLSP considers a holding cost per item in stock, the PSP assumes no holding costs but instead enforces a maximum stock level for each item type.

Another key challenge in the PSP is the order selection and scheduling, for which we refer interested readers to [10] for a thorough review of the topic. Note that order selection and scheduling coupled with sequence-dependent setup times becomes significantly more challenging to solve since the production order can impact the amount of unproductive time introduced depending on the setups required for each machine. A recent problem variant which combines both characteristics was studied by [9].

The PSP differs from the aforementioned DLSP and other LSP variants in three main ways which in combination with one another makes the problem very challenging. First, demand and due dates are not entirely fixed and should be decided. Second, daily production capacities can be increased via parallel production on machines, overtime and night shift production, however all of these options incur additional costs. Third, setup or maintenance operations can be spread over multiple non-continuous blocks, between which the dedicated operator might be assigned other duties, such as the setup or maintenance of another machine. The demand selection and multiple due date options associated with the PSP resemble the order selection and scheduling with leadtime flexibility considered by [2] for a single-machine system. However, the PSP remains distinct due to the second and third aforementioned characteristics.

The remainder of this paper is organized as follows. Section 2 formally introduces the PSP. Section 3 details a *Late Acceptance Hill-Climbing* (LAHC) heuristic [1] we designed for the PSP. A recent successful application of LAHC in combination with exact methods to solve a variant of the LSP [6] encouraged us to select this method. Section 4 goes on to provide a computational study and introduce new instances, while Section 5 concludes the paper.

## 2 Notation and problem description

The PSP is modeled over an ordered set  $D = \{1, \dots, |D|\}$  of days, each of which begins at 05:00 and has a duration of 24 hours. Each day is decomposed into a set  $B = \{1, \dots, b_n\}$  of time *blocks* of equal length as depicted in Fig. 1. Fig. 1 also highlights specific blocks that define intervals during which certain operations can take place. A task to be carried out on a machine takes an integer number of blocks to be completed and cannot take less than one block. Throughout the remainder of this paper, time is expressed in terms of number of blocks.

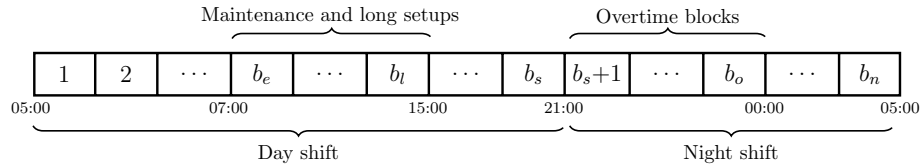


Fig. 1: Representation of one day as a set of time blocks.

A day begins with block 1 and ends with block  $b_n$ , after which the following day begins. The factory is open every day between blocks 1 and  $b_s$ . Blocks  $1, \dots, b_s$  constitute the *day shift*, whereas blocks  $b_s+1, \dots, b_n$  constitute the *night shift*. The night shift may be used at an additional cost in one of two different ways. First, blocks  $b_s+1, \dots, b_o$  may be individually scheduled as *overtime* and incur a cost  $p_o$  per block. Overtime may be scheduled any day and as often as necessary. Alternatively, a *full night shift* may be scheduled at a cost  $p_n$  per day, thereby enabling all night blocks  $b_s+1, \dots, b_n$  to be used. The organization in question requires full night shifts to be scheduled for at least  $d_n$  consecutive days. We refer to this as the *minimum consecutive night shifts* constraint. It is not possible to schedule overtime and a full night shift on the same day.

There are two machines available to produce a set  $I$  of different item types. Let  $M = \{m_1, m_2\}$  denote the set of machines and  $e_i^m \geq 0$  represent the total number of type  $i \in I$  items that machine  $m \in M$  produces per block. If machine  $m$  cannot produce items of type  $i$ , then  $e_i^m = 0$ .

Machines can operate simultaneously, which we will henceforth refer to as *parallel operation*. Parallel operation occurs in a day  $d \in D$  and a block  $b \in B$  when no machines are idle during this time slot. Parallel operation during the day shift incurs an additional cost  $p_p$  for that day because the company must hire an additional outsourced external worker. During night shifts, however, machines can operate in parallel without additional costs because night shifts already include the outsourced external worker in their cost.

Machines operate to fulfill a set  $R$  of customer requests, where each  $r \in R$  requires a quantity  $q_r^i$  of item type  $i \in I$  to be produced. Let  $c_i$  be the sales price of each item type  $i \in I$ , then  $c(r) = \sum_{i \in I} q_r^i c_i$  denotes the revenue obtained from request  $r \in R$  when it is fulfilled. Each request  $r$  has a set of allowed shipping days  $D_r \subseteq D$  which are specified by the customer. A request  $r$  can only be shipped on day  $d \in D_r$  if all the necessary quantities of items are available at the end of day  $d$  (partial shipments are not permitted). Items do not necessarily need to be produced during day  $d$  and may instead be taken from the available stock. If a request cannot be shipped by its latest feasible day, it is considered *unserved*.

The allowed shipping days are also crucial with respect to maintaining feasible stock levels. Shipments are scheduled by the end of each day (after block  $b_n$ ). After shipment, but before the start of the next day, stock levels must be updated. The stock on day  $d \in D$  corresponds to the stock on day  $d-1$  plus the total production minus the shipped items on day  $d$ . For each item type  $i \in I$ , its stock at the end of day  $d$  must never exceed the maximum stock level  $S_i^{\max}$  that can be stored in the warehouse. There is also a minimum level  $S_i^{\min}$  that should be kept in stock at the end of each day to ensure sufficient resources are available in case of disruptions or unexpected orders. Having stocks below the minimum stock level is allowed, but incurs a penalty  $p_s$  at the end of the day per unit below the minimum. Naturally, the stock of an item can never be negative.

For each  $(m, d, b)$  tuple such that  $m \in M$ ,  $d \in D$  and  $b \in B$ , we assign one of the following four tasks: (i) production of an item type, (ii) machine setup

from one item type to another, (iii) maintenance and (iv) idle. If a task requires more than one block of time, then multiple blocks must be scheduled. Tasks with multiple blocks are not necessarily continuous. Idle and maintenance/setup blocks may be scheduled in between other setup/maintenance tasks. When night shift blocks are not employed in the schedule, they are all set to idle.

The type of item produced on a machine can be changed with sequence-dependent setup times that require  $s_{ij}^m$  blocks to change the configuration of machine  $m$  from production of item type  $i \in I$  to type  $j \in I$ . These setups are classified as either short ( $U$ ) or long ( $L$ ). A short setup  $(i, j) \in U$ ,  $i, j \in I$  may be carried out at any time. Meanwhile, a long setup  $(i, j) \in L$ ,  $i, j \in I$  is only allowed during the range of blocks  $[b_e, b_l]$  due to staff-related constraints. Additionally, maintenance must be scheduled for each machine  $m \in M$  at most  $g_{\max}^m$  days apart and each maintenance takes  $f^m$  blocks. No production is possible while a machine is undergoing maintenance or setup. Long setups and maintenance can only be scheduled during the range  $[b_e, b_l]$  and are never allowed in parallel with another maintenance or long setup.

Since production is supposed to be a continuous process, we need to take into account certain information concerning schedules from the previous time horizon. This historic information includes: the last item type for which each machine was configured, the number of days since maintenance was performed for each machine, the stocked quantity of each item, and the number of consecutive night shifts scheduled at the end of the preceding scheduling period. The last of these parameters provides us with the number of *mandatory night shifts*  $h_n \geq 0$ : the number of night shifts that must be scheduled at the beginning of the time horizon in order to comply with the minimum consecutive night shifts constraint.

The primary decision is to select when and which requests to produce and ship. This not only involves deciding on which date/time, on which machine and in which order to produce items, but also how many items of each type to produce, and how many items to take from the stock. Additionally, we must decide when to schedule overtime or night shifts to extend production capacity. All decisions must also account for the constraints related to the limited shipping days, stock levels, machine maintenance and limited setup times.

A solution is sought which minimizes the revenue loss from unserved requests, additional personnel costs (overtime, night shift and parallel operations) and the penalties incurred by stock deficits. Let  $X$  be the set of all feasible solutions to the PSP and  $f(s)$  be the cost of solution  $s \in X$ . The aim is to find the minimum cost solution  $s^* = \arg \min_{s \in X} f(s)$ . This cost is defined by five components. First, the cost  $c(r)$  of all unserved requests  $r \in R$  (or the revenue loss). Second, a cost  $p_p$  for each day  $d \in D$  that contains at least one parallel operation. Third, a cost  $p_n$  for each day  $d \in D$  that a night shift has been scheduled. This cost is incurred even if all night shift blocks are completely idle, as long as day  $d$  is part of a minimum consecutive night shifts sequence. Fourth, a cost  $p_o$  for each block  $b \in B$  of overtime scheduled for each day  $d \in D$ . Finally, a penalty  $p_s$  is incurred per unit of item  $i \in I$  below its minimum stock level  $S_i^{\min}$  at the end of every day  $d \in D$ .

In order to formally define the problem, an Integer Linear Programming (ILP) formulation is provided in Appendix A. This ILP formulation is also used to assess the performance of the heuristic approach proposed in Section 3.

### 3 A heuristic approach

Preliminary experiments revealed that the ILP formulation can only provide high-quality solutions for very small instances within a time limit of one hour. Thus, a tailored algorithm is required in order to produce high-quality solutions for large-scale PSP instances within reasonable processing times. In this paper, we employ a heuristic which improves an initial solution through insertion and removal of requests in order to efficiently explore the PSP’s solution space.

The proposed heuristic to solve the PSP adapts LAHC [1], which is a simple metaheuristic framework that requires only a single parameter: the length of the list containing previous solution costs. LAHC has recently been employed to solve a variant of the LSP and achieved high-quality results [6], demonstrating that it is also a good choice for this class of problems. Algorithm 1 outlines the LAHC algorithm employed to solve the PSP.

Algorithm 1 begins by constructing an initial feasible solution using the procedure which will be detailed in Section 3.1, followed by initializing LAHC’s fitness array  $F$  and counter variables  $iter$  and  $idle$  (lines 2–4). The main loop of the algorithm (lines 5–18) is iterated over until either  $M_{iter}$  iterations have been reached or no improvement has been observed for  $\lfloor \alpha M_{iter} \rfloor$  iterations. Both  $M_{iter}$  and  $\alpha \in [0, 1]$  are parameters of the LAHC.

---

**Algorithm 1:** Late Acceptance Hill-Climbing (LAHC).

---

```

1 Input: Instance of the PSP, parameters  $M_{iter}, \alpha, L_s, \gamma, \beta_{roll}, \beta_{night}, \eta$ ;
2  $s \leftarrow \text{initialSolution}()$ ;
3  $F[k] \leftarrow +\infty, k = 1, \dots, L_s$ ;
4  $iter, idle \leftarrow 0$ ;
5 while  $iter < M_{iter}$  and  $idle < \lfloor \alpha M_{iter} \rfloor$  do
6    $s' \leftarrow \text{buildNewSolution}(s, iter, M_{iter}, \gamma, \beta_{night}, \eta)$ ;
7   if  $f(s') \geq f(s)$  then
8      $idle \leftarrow idle + 1$ ;
9   else
10     $idle \leftarrow 0$ ;
11     $k \leftarrow iter \bmod L_s$ ;
12    if  $f(s') < F[k]$  or  $f(s') \leq f(s)$  then  $s \leftarrow s'$ ;
13    if  $f(s) < F[k]$  then  $F[k] \leftarrow f(s)$ ;
14    if  $idle = \lfloor \alpha \beta_{roll} M_{iter} \rfloor$  then  $s \leftarrow s^*$ ;
15     $iter \leftarrow iter + 1$ ;
16 return  $s^*$ 

```

---

In every iteration of the LAHC, a new solution  $s'$  is generated using the current solution  $s$  and the procedure which will be described in Section 3.2 (line 6). Then, lines 7–15 update the idle iteration counter, the current solution  $s$  and the fitness array  $F$  according to the original strategy introduced by [1]. Note

that  $F$  allows a worsening solution  $s'$  to be accepted whenever its total cost is less than the cost of the solution  $L_s$  iterations earlier, where  $L_s$  is the size of LAHC's list of solution costs. To increase intensification of the search, we included a rollback procedure so that when the number of idle iterations reaches a specified percentage of its maximum value, the current solution  $s$  is replaced with  $s^*$  (lines 16-17). Here,  $\beta_{\text{roll}} \in [0, 1]$  is also a parameter. The first rollback is always allowed. Any additional rollbacks can only occur if  $s^*$  is improved after the preceding rollback. The best solution  $s^*$  generated over all iterations is returned (line 19).

### 3.1 Initial solution

Initial solution  $s$  is constructed by first initializing the blocks for every day and every machine as idle. Next, the required maintenance is scheduled for each machine with as many days in-between as possible while avoiding parallel maintenance blocks. These initial maintenance days are fixed throughout the entirety of LAHC's execution. After these steps, solution  $s$  should contain a feasible maintenance schedule for all machines, otherwise the instance is considered infeasible. Algorithm 2 outlines the steps to build the initial solution.

---

#### Algorithm 2: initialSolution()

---

```

1  $s \leftarrow \text{idleSolution}();$ 
2  $s \leftarrow \text{scheduleMaintenance}(s);$ 
3  $O_r \leftarrow \text{true}, \forall r \in R;$  // Initially all requests are available
4 while  $\exists r \in R : O_r = \text{true}$  do
5    $c^{\max} \leftarrow \max c(r) : O_r = \text{true};$ 
6   select request  $y : O_y = \text{true}$  with probability  $0.8 \cdot c(y)/c^{\max};$ 
7    $s \leftarrow \text{insertRequestBestPosition}(s, y, \text{true});$ 
8    $O_y \leftarrow \text{false};$ 
9 return  $s;$ 

```

---

Once a feasible maintenance schedule is generated, requests are inserted into  $s$  along with the necessary production and setup blocks. These insertions are performed in a greedy-randomized manner (lines 4–8 of Algorithm 2). While there remain available requests to be inserted, one request  $r \in R$  is selected at random with probability  $0.8c(r)/c^{\max}$ , where  $c(r)$  denotes the profit for shipping request  $r$  and  $c^{\max}$  is the value of the most profitable request among all currently available ones (lines 5–6). The selected request  $r$  is inserted into its best shipment day  $d^* \in D_r$  using the procedure described in Section 3.3 (line 7). Once  $r$  is inserted into  $s$  or once no feasible day remains into which  $r$  can be inserted, the request is marked as *processed* so that it is no longer considered for insertion in the construction phase (line 8). Request insertion is repeated until all requests have been marked *processed*.

### 3.2 New solution generation

A new solution  $s'$  is constructed from the current solution  $s$  via a series of modifications. Algorithm 3 outlines the steps to perform these changes. The algorithm removes requests at random from  $s'$ , as well as unnecessary production blocks arising from such removal, employing the procedure described in Section 3.4 (line 7 of Algorithm 3). Note that by removing requests and unnecessary production blocks, more space or *slack* is created to later reinsert requests and therefore more effectively explore the solution space of the PSP. The algorithm reinserts these requests in a random order on their best possible shipping dates using the heuristic outlined in Section 3.3 (line 8). The number of removed requests is selected uniformly from  $[1, \max\{20, \lfloor \gamma |R| \rfloor\}]$ , where  $\gamma \in [0, 1]$  is a parameter of the LAHC. We limit the removal to a maximum of 20 requests to account for very large instances. In the case that not all of the removed requests are successfully reinserted, the remaining unserved requests incur a penalty.

---

**Algorithm 3:** buildNewSolution( $s$ , iter,  $M_{\text{iter}}$ ,  $\gamma$ ,  $\beta_{\text{night}}$ ,  $\eta$ )

---

```

1 Input: Solution  $s$ , num. of iterations iter, parameters  $M_{\text{iter}}$ ,  $\gamma$ ,  $\beta_{\text{night}}$ ,  $\eta$ ;
2  $s' \leftarrow s$ ;
3 if iter >  $M_{\text{iter}}\beta_{\text{night}}$  and iter mod  $\eta = 0$  then
4    $s' \leftarrow \text{removeUnusedOvertimeAndNightShift}(s')$ ;
5    $s' \leftarrow \text{insertRandomOvertimeOrNightShift}(s')$ ;
6  $y \leftarrow \text{rand}(1, \min\{20, \gamma|R|\})$ ;
7  $s' \leftarrow \text{randomRequestRemoval}(s', y)$ ;
8  $s' \leftarrow \text{bestRequestInsertion}(s')$ ;
9  $s' \leftarrow \text{fixStockBelowMinimum}(s')$ ;
10 return  $s'$ ;

```

---

Once the number of iterations in Algorithm 1 reaches  $\lfloor \beta_{\text{night}} M_{\text{iter}} \rfloor$ , new solutions are permitted to employ overtime or additional night shifts. Here, value  $\beta_{\text{night}} \in [0, 1]$  is another parameter of the LAHC. Overtime and night shifts are modified as follows (lines 3–5). First, unused but active overtime and night shifts are removed, followed by the activation of new overtime or night shift blocks. The procedures for (de)activating overtime and night shift blocks are detailed in Section 3.5. These (de)activations are only executed every  $\eta$  iterations of the LAHC so that the algorithm has sufficient time to make best use of the new overtime or night shift blocks. The value of  $\eta$  is parameterized as well. The decision to postpone the activation of night shifts and overtime to later iterations in the LAHC’s execution is not arbitrary. In practice, night shifts and overtime are deemed undesired by both employers and employees. Therefore, avoiding their use forces the LAHC to produce solutions without featuring them.

When all requests have been fulfilled in  $s'$ , a procedure to increase stock levels by scheduling production blocks is employed (line 9). This procedure examines each day  $d \in D$  for the items  $i \in I$  that have stocks below their minimum level. It then attempts to schedule production blocks for these items on days  $d' \leq d$  to reduce stock penalties. Production blocks on days  $d' < d$  are only added if they do not result in a solution exceeding maximum stock levels.



### 3.3 Request insertion heuristic

A shipping day must be determined for each request while respecting stock levels and only considering the allowed shipping days for that request  $D_r$ . Our proposed request insertion heuristic inserts requests into each allowed shipping day. The shipping day which yields the best result is then selected. When attempting to ship a request  $r$  on a given day  $d$ , production of all requested items  $q_r^i$  ( $i \in I$ ) is scheduled in such a way that the maximum possible number of items is produced from day 0 until day  $d$ . When the produced items are not enough to serve the request, the shortfall of production is compensated by preexisting stock. By producing as many items as possible for each inserted request, solutions have less chance of violating minimum stock levels and stocks may be preserved to help serve requests with greater demand.

Algorithm 4 outlines the overall framework of the request insertion heuristic for a single request  $r$ . The insertion of parallel production is optional and given as a parameter. Production of items is inserted backwards, beginning from the shipping day  $d$  back until the first day of the time horizon (lines 8-11). This mechanism aims to maintain production as close to the shipment date as possible, which decreases the chance of violating maximum stock levels. If a solution producing all  $q_r$  items is not found, the remaining items to complete request  $r$  are removed from stocks (lines 12-15). In this case, minimum stock levels can be violated. After all permitted shipping days have been checked, the best solution is returned (line 19).

The *insertProduction* method receives a set of item types and their respective quantities  $RP$  to be produced on a given day  $d'$ . For every  $i \in I$  where  $RP_i \geq 1$ , the method attempts to insert a total of  $b = \lceil RP_i / e_i^m \rceil$  production blocks on any machine  $m \in M$  where  $e_i^m > 0$ . The method ends as soon as either all production has been scheduled or once it is not possible to insert any more production on that day. The items and machines are iterated over sequentially. The production of  $b$  blocks of item type  $i$  is then scheduled on machine  $m$  on day  $d'$  in accordance with one of two possible methods.

The first method is used when machine  $m$  on day  $d'$  has no production of item  $i$ . In this scenario, production of item  $i$  is inserted along with the necessary setups into the first sequence of idle blocks on day  $d'$  where it can be fit. If the production of  $b$  blocks is not possible, the same method is called to insert  $b - 1$  blocks on the same day and on the same machine. The second method handles the insertion of production of item  $i$  on a day and machine that is already producing at least one block of  $i$ . This method increments the sequence of production blocks of  $i$  by  $b$  blocks. Tasks already scheduled on day  $d'$  are pulled back or pushed forward, replacing idle blocks. The resulting day is feasible if every non-idle task remains scheduled and no conflict is found.

### 3.4 Request removal heuristic

Given a current solution and a set of requests  $\bar{R}$  to be removed from the schedule, the request removal heuristic starts by removing the scheduled shipping days of all requests in  $\bar{R}$ . Stock levels for every item and day are then recalculated. In

**Algorithm 4:** insertRequestBestPosition( $s, r, bp$ )

---

```

1 Input: Solution  $s$ , Request  $r$ , boolean  $bp$  to enable parallel production;
2  $s^* \leftarrow s$ ;
3 foreach  $d \in D_r$  do
4    $s' \leftarrow s$ ;
5    $RP \leftarrow q_r^i \forall i \in I$ ;           // Remaining production of each item
6    $IP \leftarrow \emptyset \forall i \in I$ ;       // Inserted production of each item
7    $d' \leftarrow d$ ;
8   while  $d' \neq 0$  or  $RP \neq \emptyset$  do
9     insertProduction( $s', d', RP, IP, bp$ );
10     $RP \leftarrow RP \setminus IP$ ;
11     $d' \leftarrow d' - 1$ ;
12  if  $RP \neq \emptyset$  then
13     $possible \leftarrow \text{RemoveFromStock}(s', RP, d)$ ;
14    if  $possible = \text{False}$  then go to next  $d$ ;
15   $s' \leftarrow \text{Schedule shipping of request } r \text{ on day } d$ ;
16  if  $f(s') < f(s^*)$  then
17     $s^* \leftarrow s'$ ;
18 return  $s^*$ ;

```

---

this step, solutions often have a large number of items being produced which are not shipped, possibly violating maximum stock levels.

In order to remove unnecessary production and correct stock level violations, a removal slack  $SLK$  is calculated for each day and item type.  $SLK$  expresses how many production tasks of item type  $i$  can be removed from day  $d$  without violating  $S_i^{min}$  and is calculated as  $SLK_{[d][i]} = \min(stock_{[d]} - S_i^{min}, SLK_{[d+1][i]})$ .  $SLK$  is used as an upper bound and the request removal heuristic continues by removing as many production tasks as possible for each day and item type. Removing a large number of production tasks results in a partial solution with more idle blocks, providing additional flexibility for request insertion. When removing production tasks, setups for certain item types become obsolete as those items are no longer being produced, hence these setups are also removed.

### 3.5 Overtime and night shift heuristics

Once the number of iterations performed by Algorithm 1 exceeds the threshold defined in Section 3.2, non-mandatory night shifts and overtime blocks are (de)activated in the schedule. For a particular day  $d \in D$ , (de)activation of night shifts and overtime is performed by the use of *Boolean flags* indicating whether a night shift (alternatively overtime) is active for a day  $d$ . Overtime and night shift flags cannot be simultaneously active during the same day  $d$ .

*Overtime and night shift removal:* Before the insertion of overtime or night shifts, the algorithm first removes all unused blocks. For every day  $d$  with active overtime but for which all blocks in  $[b_{s+1}, b_o]$  contain idle tasks, the overtime flag is deactivated. For night shifts, a sequence of at least  $d_n$  consecutive days with night shifts is extracted from the solution (if such a sequence exists). All empty

night shifts – those with only idle blocks in both machines – in this sequence are deactivated from either the beginning, the end, or the middle of the sequence so long as the minimum consecutive night shift constraint is respected. Note that mandatory night shifts are never removed and are always available for use.

*Overtime and night shift insertion:* After the removal of unused overtime and night shifts, new insertions are performed. The algorithm chooses with uniform probability one of the three following insertion methods:

- (I1) Overtime insertion: a number  $\delta \in [1, |D|]$  is selected with uniform probability. Then, the procedure iterates over all days  $D$  in a random order, activating overtime flags for days without any active flags. This continues until either  $\delta$  overtime blocks have been activated or all days have been checked.
- (I2) Earliest night shift insertion: this can only be executed if the instance contains mandatory night shifts. The algorithm selects with uniform probability a number  $\delta \in [1, d_n - 1]$ . Then, starting from the first day without a mandatory night shift, the algorithm activates night shifts for the next  $\delta$  days in the time horizon. Any of the  $\delta$  days which already contains an activated night shift is counted as an activation. In the case that a day contains overtime, it is deactivated and a night shift is activated in its place.
- (I3) Random night shift insertion: a day  $d_1 \in [h_n, |D|]$  is selected at random. Then, for all days  $d_1, \dots, d_1 + d_n$ , night shifts are activated. Any active overtime is replaced with a night shift.

For both (I2) and (I3), feasibility with respect to the minimum consecutive night shift constraint is maintained at all times.

## 4 Computational study

In order to provide some managerial insights on certain PSP characteristics and analyze the performance of the ILP model as well as the LAHC heuristic, this section presents the results obtained from a computational study on the PSP.

All experiments were conducted on a computer with Intel Xeon E5-2660 at 2.6 GHz and 160 GB of RAM running Ubuntu 20.04 LTS. The LAHC was implemented in C++, compiled using g++ 9.3 and executed in single-thread mode. The ILP was implemented using the C++ API of Gurobi 9 and it was run for up to one hour per instance, using maximum eight threads. Based on the company's requests, we set the maximum execution time of the LAHC to ten minutes.

### 4.1 New instance sets

In order to run experiments using the proposed algorithm and stimulate further research regarding the PSP, instances were generated based on real-world data. The instances, solutions and a validator are publicly available at an online repository [8]. The company that inspired this work provided us with a set of items ( $I$ ) and machines ( $M$ ), minimum and maximum stock levels per item ( $S_i^{\min}$  and  $S_i^{\max}$ ), a set of short/long setups and their duration ( $U$ ,  $L$  and  $s_{ij}^m$ ), maintenance durations and frequencies per machine ( $f^m$  and  $g_{\max}^m$ ) and time restrictions for long setup and maintenance (bounded by the block indexes  $b_{e,l,s,o,n}$ ). The efficiency of machines per item ( $e_i^m$ ) is identical for both machines

with the exception of certain items which cannot be produced by machine  $m_2$ , in which case  $e_i^{m_1} > 0$  and  $e_i^{m_2} = 0$ . This data was considered standard and left unaltered for every instance. The company also provided the cost of items ( $c_i$ ), overtime ( $p_o$ ), night shift ( $p_n$ ) and parallel production ( $p_p$ ). For privacy reasons, these values were converted into proportional values. Finally, a month’s worth of customer requests were provided and used as the basis for generating multiple instances.

Consider  $AvgI = \frac{\sum_{r \in R} \sum_{i \in I} q_r^i}{|D|}$ , which is the average number of items requested per day in a given time horizon of  $|D|$  days. The company generates their production schedule for a time horizon of 10 days considering blocks of 60 minutes and an  $AvgI$  up to one million in high-demand weeks. Based on the provided data, we generated two instance sets corresponding to periods of low and high production demands. For every instance in the low-demand and high-demand sets,  $AvgI = 500,000$  and  $1,000,000$ , respectively. The high-demand set corresponds to a busy scenario for the company and can therefore be considered realistic in terms of size. Each benchmark set contains 18 instances. Each instance is named in accordance with its three primary attributes:  $|D|$ ,  $|R|$  and  $b_{dur}$  (the length of each block in minutes). For example, the high-demand instance H\_10\_15\_30 has a time horizon of 10 days, 15 requests (with an average of 1,000,000 items per day) and blocks of 30 minutes.

Given  $AvgI$  and these three primary attributes, the remaining attributes of each instance were generated as follows. The minimum number of consecutive days with night shifts  $d_n$  was selected with uniform probability from  $[2, \min(10, |D| * 0.5)]$ . For each request, the permitted shipping days  $D_r \subseteq D$  are either every day, every two days or every five days with selection probabilities 0.4, 0.4 and 0.2, respectively. Each request  $r \in R$  comprises of at most three different items selected with uniform probability from  $I$ . The demand for each request  $r$  is selected from  $[0.8, 1.1]$  of the average item demand per request ( $\frac{AvgI * |D|}{|R|}$ ) and divided randomly among the items comprising request  $r$ . Note that two instances with the same  $|D|$  and  $|R|$  are identical in every aspect, except for  $b_{dur}$ . Moreover, instances from the same instance set have the same  $AvgI$  despite the varying number of requests  $|R|$ .

## 4.2 LAHC parameters

Parameters for LAHC were obtained by tuning the algorithm with the irace package [7]. Tuning was performed for the two instance types, resulting in a low-demand and a high-demand parameter set. This is not arbitrary as schedules for low- and high-demand instances differ considerably. In each tuning, irace was given a budget of 5,000 runs and six randomly selected instances as the training set. We fixed LAHC’s maximum number of iterations to  $M_{iter} = 20,000$  and the idle rate  $\alpha = 0.2$ . The best parameter sets reported by irace are provided in the format  $\{L_s, \gamma, \beta_{roll}, \beta_{night}, \eta\}$ . For the low-demand instances parameter values were  $\{2000, 0.60, 0.56, 0.62, 20\}$ , whereas for the high-demand instances they were  $\{2000, 0.64, 0.84, 0.01, 60\}$ .

### 4.3 Results

Table 1 provides the results of the ILP and those obtained by the LAHC. The results concerning instances in the same set with the same  $|D|$  and  $|R|$  but different block sizes are aggregated into a single row. For example, row L\_10\_15 provides the aggregated results for instances L\_10\_15\_15, L\_10\_15\_30 and L\_10\_15\_60. Moreover, since the LAHC is an inherently stochastic algorithm, we ran it ten times per instance with different seeds for the random number generator. Therefore, the cells associated with LAHC correspond to the averages or minimums of 30 runs: 10 runs per instance for 3 different block sizes. Similarly, for the ILP, each cell corresponds to the average or minimum of 3 runs: 1 run per instance for 3 different block sizes. The online repository [8] provides a complete table with detailed results to each individual instance.

In these experiments, the ILP reached the one-hour time limit for all instances except L\_10\_15\_30 and L\_10\_15\_60 (solved in 1800 and 800 seconds, respectively). Therefore, the ILP columns in Table 1 report only the upper bounds (the values of the best solutions found) and lower bounds provided by the solver, but not the computation times. More specifically, columns  $UB_{\min}$ ,  $UB_{\text{avg}}$  and  $LB_{\text{avg}}$  report the minimum upper bound, average upper bound and average lower bound, respectively. For the LAHC, column BKS reports the best-known solution value (cost). The next columns report the average values for the solution cost ( $S_{\text{avg}}$ ), execution time in seconds ( $\text{Time}_{\text{avg}}$ ), standard deviation of the solution costs ( $SD_{\text{avg}}$ ), number of blocks used for overtime ( $OT_{\text{avg}}$ ), number of days with night shifts ( $NS_{\text{avg}}$ ), number of days with parallel tasks ( $PD_{\text{avg}}$ ) and number of unserved requests ( $UR_{\text{avg}}$ ).

Table 1: ILP and LAHC results.

Instance	ILP			LAHC							
	$UB_{\min}$	$UB_{\text{avg}}$	$LB_{\text{avg}}$	BKS	$S_{\text{avg}}$	$\text{Time}_{\text{avg}}$	$SD_{\text{avg}}$	$OT_{\text{avg}}$	$NS_{\text{avg}}$	$PD_{\text{avg}}$	$UR_{\text{avg}}$
L_10_15	900.00	143,438.03	900.00	985.00	1,074.50	341.93	34.17	4.63	0.00	0.53	0.00
L_10_25	120.00	212,243.26	0.00	258.75	348.06	259.74	42.20	7.82	0.00	1.43	0.00
L_20_15	331,921.48	522,131.40	0.00	307.50	431.10	522.36	48.38	16.89	0.00	0.37	0.00
L_20_25	683,697.80	703,534.73	0.00	595.00	801.96	525.58	76.02	8.17	0.00	4.30	0.00
L_40_50	1,538,832.12	1,539,432.12	0.00	1,780.00	2,044.94	601.65	97.98	6.04	0.00	14.87	0.00
L_40_100	1,578,473.24	1,579,463.24	0.00	1,920.00	2,282.71	602.96	76.14	2.72	0.00	18.33	0.00
H_10_15	70,580.86	394,387.34	1,594.74	3,782.50	3,963.75	228.38	49.52	0.40	6.83	8.13	0.00
H_10_25	64,098.45	317,605.53	15,649.56	62,604.85	84,184.75	185.93	10,383.02	2.75	8.80	6.00	2.53
H_20_15	1,497,758.35	1,621,849.64	76,390.43	233,532.05	269,658.30	547.06	37,532.99	4.93	15.23	12.50	2.37
H_20_25	1,281,182.11	1,533,486.47	2,855.43	7,637.50	20,550.81	577.50	8,581.04	0.19	11.83	15.17	0.23
H_40_50	3,481,699.59	3,481,999.59	0.00	282,959.48	493,129.05	601.41	66,928.92	64.87	10.30	35.70	6.47
H_40_100	3,324,599.38	3,331,056.53	1,800.00	13,128.75	59,202.84	603.47	67,203.97	8.81	27.17	34.37	1.33

The first thing to note from Table 1 is that the ILP was able to find the best solution for instances L\_10\_15 and L\_10\_25 when comparing the minimum values  $UB_{\min}$  (ILP) and BKS (LAHC) for these instances. However, when considering the average across all three block sizes, LAHC obtains far lower solution costs in under five minutes. For both low- and high-demand instance types, LAHC finds solutions with costs far lower than the ILP's as the instances become larger. Such differences in solution quality are due to the size of the ILP model, which becomes significantly large and experiences significant difficulty

to solve even for a time horizon of just 20 days. Indeed, the ILP was unable to produce any feasible solution for instances of 40 days and blocks of 15 minutes.

In terms of the lower bounds produced by the ILP, the  $LB_{\text{avg}}$  for low-demand instances is always the trivial bound considering only mandatory night shifts and maintenance without any production. Meanwhile, for high-demand instances the ILP improved the lower bound for those with time horizons of less than 40 days, sometimes even by a large margin (for example instance H\_20\_15 for which the trivial LB is 0). This may occur due to the fact that in high-demand instances, machine occupancy rates are high enough for the ILP to prove that lower solution values are impossible, whereas with low occupancy rates this is harder to prove since more blocks are likely idle and could be used to avoid parallel tasks or stock penalties. Because block usage depends on several other factors, the model requires longer execution times to improve lower bounds.

LAHC's standard deviation is low for low-demand instances, whereas for high-demand instances the observed variation increases significantly. This high standard deviation is possible given the different number of unserved requests which incur large penalties. Column  $UR_{\text{avg}}$  shows that while all requests were served for the low-demand instances, in the high-demand instances a number of requests remained unserved, increasing the solution cost. For example, on average 97% of the total cost of solutions for H\_40\_50 instances is due to unserved requests. Meanwhile, solutions for H\_40\_100 are only penalized in the 15-minute block set where unserved requests account for 50% of the total cost on average, but for blocks of 30 and 60 minutes all requests are served and so no penalty is incurred. These results indicate how the difficulty of solving the problem increases as the search space expands.

Out of the total hours available for overtime and total number of days, the low-demand instance set employs on average 15% of overtime hours and 21% of days with parallel tasks. These are the two main components that incur costs in the low-demand instances as no night shifts are employed and the penalty per item under minimum stock levels is on average responsible for only 4% of the total solution cost. Meanwhile, high-demand instances employ on average 64% of the available night shifts, 13% of overtime hours and 76% of days with parallel tasks. For high-demand instances with 40 days the usage of night shifts, overtime and parallel tasks may reach as high as 79%, 65% and 90%, respectively. Items below the minimum stock levels were also successfully avoided in the high-demand instances, representing 0.5% of the total solution cost.

Further analyses are performed concerning the impact of block lengths and the relaxation of different constraints. Table 2(a) provides the solution gaps for each block size and instance set. For a block size  $b_{dur} \in \{15, 30, 60\}$ ,  $gap_{\text{BKS}}$  of  $b_{dur}$  is calculated by  $\frac{BKS(b_{dur}) - \min_{\text{BKS}}}{\min_{\text{BKS}}}$ , where  $BKS(b_{dur})$  is the best solution found for  $b_{dur}$  and  $\min_{\text{BKS}} = \min_{t \in \{15, 30, 60\}} BKS(t)$ . Similarly,  $gap_{\text{avg}}$  is calculated between the average solution of  $b_{dur}$  and the minimum average solution of all block sizes. The average processing time is reported by  $time_{\text{avg}}$ . Solutions with blocks of 30 minutes often perform better than the other two block lengths for both instance sets. Instances with blocks of 60 minutes have the quickest

processing times and obtain the second best gap. In contrast, when scheduling blocks of 15 minutes, the search space is much larger and this results in longer processing times, fewer iterations and worse solution values. To evaluate statistically significant differences, the pairwise T-test was performed with a confidence level of 95%. Although using blocks of 30 minutes resulted in the best solutions, no statistically significant difference was found when comparing the results of the three block sizes.

Table 2(b) provides the gap between the average solutions produced by the LAHC for the original instance sets and those obtained when relaxing one of the following PSP constraints: (i) shipment day, meaning requests may be shipped on any day; (ii) time windows for long setup and maintenance, meaning these two tasks may be performed during any block and; (iii) maximum stock levels, where  $S_i^{max}$  is doubled for every item. These constraints were selected because we consider them to be the most constraining. For the low-demand instance set, statistical tests were performed using the Wilcoxon signed-rank test and demonstrated significant differences between LAHC’s results and those obtained by the shipment day and time window relaxations. While relaxing shipment days improves the solutions, relaxed time windows counter-intuitively resulted in a worsening of solution quality. The reasons behind these results are twofold. First, time window relaxation increases the number of blocks to be considered for maintenance and long setups and increases the size of the search space significantly. Second, results show an increase of days with parallel tasks and total number of used overtime blocks, while a decrease on items under the minimum stock is observed. Therefore, the flexibility given by the relaxed time windows enables more production of items to be scheduled, which is considered a priority of the request insertion heuristic (to insert as many production tasks as possible and take as few items as possible from stock). To remedy this behavior one option would be to calibrate the algorithm and give it more time to insert production while prohibiting parallel tasks and overtime blocks.

Table 2: Sensitivity analysis.

$b_{dur}$	Low			High			Inst. set	Ship any day	Maintenance any time	Double Max stock
	15	30	60	15	30	60				
gap <sub>BKS</sub>	13.39	0.00	13.77	19.51	4.32	7.58	Low	-5.00	13.13	-3.11
gap <sub>avg</sub>	11.29	0.06	13.57	254.71	5.26	6.46	High	-3.48	-16.68	-33.91
time <sub>avg</sub>	572.78	483.93	370.40	521.14	458.01	392.73				

(a) Impact of block size

(b) Gap<sub>avg</sub> with relaxations

When considering high-demand instances, the Wilcoxon signed-rank test suggests statistically significant differences for all relaxations. The results indicate that improvements may be obtained by increasing the shipping frequency, doubling the stock capacity or hiring more skilled workers for maintenance and long setups. Indeed, the results demonstrate that doubling stock capacity would bring the largest profits, although one should also consider the construction or rental costs incurred by doing so. Interestingly, increasing the stock capacity by more than 100% did not present significant gains for the considered instances.

## 5 Conclusion

This paper introduced a real-world production scheduling problem with stock- and staff-related restrictions. To serve a profitable selection of available customer requests within a given time horizon, production, setup and maintenance tasks must be scheduled in blocks of time of predetermined lengths. In addition to an integer programming formulation of the problem, this paper also designed a heuristic algorithm with local search moves based on the insertion and removal of requests. Given the originality of the problem, and thus the lack of benchmark instances in the literature, and with an aim to stimulate future research on the subject, a set of instances was derived from real-world data provided by the company which inspired this research.

A computational study demonstrated the efficacy of the proposed meta-heuristic in producing high-quality schedules in quick processing times, even for the more challenging scenarios. Moreover, experiments were carried out to understand the impact of varying demand and block size, along with a sensitivity analysis concerning constraints regarding stock capacities, request shipment days and task time windows. This analysis suggested that companies confronted with similar situations ought to consider operational changes regarding limited shipping days, maintenance windows and stock limits. All of these changes should be exploited to increase revenue. However, we foresee that a broader analysis concerning the trade-off between the gain from such changes and the costs associated with making them represents a crucial consideration which ought to be explored by future research. Furthermore, additional studies could be conducted considering the following extensions: technician scheduling, which would result in flexible times for long setups maintenance; more than two machines, which should be considered along with the scheduling of multiple operators so as to allow parallel operations; and a dynamic version of the problem where requests are not known *a priori*.

**Acknowledgments:** Research supported by KU Leuven (C2 C24/17/012) and ‘Data-driven logistics’ (FWO-S007318N). Editorial consultation provided by Luke Connolly (KU Leuven).

## References

1. Burke, E.K., Bykov, Y.: The late acceptance hill-climbing heuristic. *European Journal of Operational Research* **258**(1), 70 – 78 (2017)
2. Charnsirisakskul, K., Griffin, P.M., Keskinocak, P.: Order selection and scheduling with leadtime flexibility. *IIE transactions* **36**(7), 697–707 (2004)
3. Copil, K., Wörbelauer, M., Meyr, H., Tempelmeier, H.: Simultaneous lotsizing and scheduling problems: a classification and review of models. *OR spectrum* **39**(1), 1–64 (2017)
4. Drexl, A., Kimms, A.: Lot sizing and scheduling — survey and extensions. *European Journal of Operational Research* **99**(2), 221–235 (1997)
5. Fleischmann, B.: The discrete lot-sizing and scheduling problem. *European Journal of Operational Research* **44**(3), 337–348 (1990)



6. Goerler, A., Lalla-Ruiz, E., Voß, S.: Late acceptance hill-climbing matheuristic for the general lot sizing and scheduling problem with rich constraints. *Algorithms* **13**(6), 138 (2020)
7. López-Ibáñez, M., Dubois-Lacoste, J., Pérez Cáceres, L., Birattari, M., Stützle, T.: The irace package: Iterated racing for automatic algorithm configuration. *Operations Research Perspectives* **3**, 43 – 58 (2016)
8. Sartori, C.S., Gandra, V., Çalik, H., Smet, P.: Instances for production scheduling with stock- and staff-related restrictions. Mendeley Data, V2 at <http://dx.doi.org/10.17632/rpbv622wyd.2> (2021), last access: 26 July 2021.
9. Silva, Y.L.T., Subramanian, A., Pessoa, A.A.: Exact and heuristic algorithms for order acceptance and scheduling with sequence-dependent setup times. *Computers & Operations Research* **90**, 142–160 (2018)
10. Slotnick, S.A.: Order acceptance and scheduling: A taxonomy and review. *European Journal of Operational Research* **212**(1), 1–11 (2011)
11. Tempelmeier, H., Copil, K.: Capacitated lot sizing with parallel machines, sequence-dependent setups, and a common setup operator. *OR spectrum* **38**(4), 819–847 (2016)
12. Wörbelauer, M., Meyr, H., Almada-Lobo, B.: Simultaneous lotsizing and scheduling considering secondary resources: a general model, literature review and classification. *Or Spectrum* **41**(1), 1–43 (2019)

## Appendix A Integer Linear Programming formulation

To model the PSP as an ILP we introduce some additional notation.

- $T$ : set of all blocks in the scheduling horizon:  $T = \{1, \dots, |B|, |B|+1, \dots, 2|B|, 2|B|+1, \dots, |D||B|\}$
- $T^M \subset T$ : set of all maintenance blocks.
- $T^N \subset T$ : set of all night-shift blocks.
- $T^O \subset T^N$ : set of all overtime blocks.
- $T_r \subset T$ : set of all shipping blocks for request  $r \in R$ .
- $d(t) \in D$ : the day index of block  $t \in T$ :  $d(t) = 1$  for  $t = 1, \dots, |B|$ ;  $d(t) = 2$  for  $t = |B| + 1, \dots, 2|B| \dots$
- $h_n$ : the index of the last day with a night shift pushed from the previous scheduling horizon.
- $g_0^m$ : at the beginning of current scheduling horizon, the number of days passed without a maintenance for machine  $m \in M$  since the last maintenance in the previous scheduling horizon.
- $b_n^d$ : the last block of day  $d \in D$ .

Additionally, a set of decision variables is used.

- $\eta_d = 1$  if there is a night shift on day  $d \in D$ , 0 otherwise.
- $\pi_d = 1$  if there is parallel processing of machines on day  $d \in D$ , 0 otherwise.
- $\theta_t = 1$  if block  $t \in T^O$  is used as overtime, 0 otherwise.
- $y_t^m = 1$  if machine  $m \in M$  is idle during block  $t \in T$ , 0 otherwise.
- $\mu_t^m = 1$  if  $m \in M$  is under maintenance during block  $t \in T$ , 0 otherwise.

- $\gamma_r^t = 1$  if request  $r \in R$  is fulfilled by shipping at  $t \in T_r$ , 0 otherwise.
- $z_{ti}^m = 1$  if  $m \in M$  is producing item  $i \in I$  during block  $t \in T$ , 0 otherwise.
- $\Delta_{id}$  is the stock level of item  $i \in I$  at the end of day  $d \in D$ .  $\Delta_{i0}$  is the initial stock of  $i$ .
- $\phi_{id}$  is the stock deficit of  $i \in I$  at the end of day  $d \in D$ .
- $\tau_d^m = 1$  if  $m \in M$  undergoes a maintenance on day  $d \in D$ , 0 otherwise.
- $v_{tt'}^m = 1$  if during block  $t \in T^M$ , machine  $m \in M$  is occupied by a long setup to be finished during block  $t' \in T^M$ , 0 otherwise.
- $w_{ij}^{mt'} = 1$  if during block  $t' \in T^M$ , machine  $m \in M$  is occupied and finished a long setup from item  $i \in I$  to item  $j \in I$ , 0 otherwise.
- $\psi_{tt'}^m = 1$  if during block  $t \in T$ , machine  $m \in M$  is occupied by a short setup to be finished during block  $t' \in T$ , 0 otherwise.
- $u_{ij}^{mt} = 1$  if during block  $t \in T$ , machine  $m \in M$  is occupied and finished a short setup from item  $i \in I$  to item  $j \in I$ , 0 otherwise.
- $\rho_{ti}^m = 1$  if machine  $m \in M$  is set-up to produce item  $i \in I$  during block  $t \in T$ , 0 otherwise ( $\rho_{0i}^m = 1$  if the initial configuration of machine  $m$  is for item  $i$ ).

The following is an integer programming formulation for the PSP.

$$\min \sum_{r \in R} \sum_{t \in T_r} c(r)(1 - \gamma_r^t) + \sum_{t \in T^O} p_o \theta_t + \sum_{d \in D} (p_p \pi_d + p_n \eta_d + \sum_{i \in I} p_s \phi_{id}) \quad (1)$$

$$\text{s.t. } \theta_t + \eta_{d(t)} \leq 1, \quad \forall t \in T^O, \quad (2)$$

$$\theta_{t+1} \leq \theta_t, \quad \forall t \in T^O, \quad (3)$$

$$\theta_t + \eta_{d(t)} + y_t^m \geq 1, \quad \forall t \in T^O, m \in M \quad (4)$$

$$\eta_{d(t)} + y_t^m \geq 1, \quad \forall t \in T^N \setminus T^O, m \in M \quad (5)$$

$$(d_n - 1)\eta_d \leq \sum_{d'=d+1}^{d+d_n-1} \eta_{d'} + (d_n - 1)\eta_{d-1}, \quad \forall d : |D| - d_n \geq d > h_n \quad (6)$$

$$(d_n - 1)\eta_d \leq \sum_{d'=d-d_n+1}^{d-1} \eta_{d'} + (d_n - 1)\eta_{d+1}, \quad \forall d \geq \max\{h_n + 1, d_n\} \quad (7)$$

$$y_t^m + \sum_{i \in I} z_{ti}^m + \sum_{t' \in T: t' \geq t} \psi_{tt'}^m + \sum_{t' \in T^M: t' \geq t} v_{tt'}^m + \mu_t^m = 1, \quad \forall t \in T^M, m \in M \quad (8)$$

$$y_t^m + \sum_{i \in I} z_{ti}^m + \sum_{t' \in T: t' \geq t} \psi_{tt'}^m = 1, \quad \forall t \in T \setminus T^M, m \in M \quad (9)$$

$$\sum_{m \in M} (1 - y_t^m) \leq (|M| - 1)\pi_d + 1, \quad \forall d \in D, t \in T \setminus T^N : d(t) = d \quad (10)$$

$$\Delta_{id} = \Delta_{id-1} + \sum_{t \in T: d(t)=d} \sum_{m \in M} e_i^m z_{ti}^m - \sum_{r \in R} q_r^i \gamma_r^t, \quad \forall d \in D, i \in I, t' = b_n^d \quad (11)$$

$$\phi_{id} \geq S_i^{min} - \Delta_{id}, \quad \forall d \in D, i \in I \quad (12)$$

$$\sum_{t \in T_r} \gamma_r^t \leq 1, \quad \forall r \in R, i \in I \quad (13)$$

$$u_{ij}^{mt} \leq \psi_{tt}^m, \quad \forall t \in T, m \in M, (i, j) \in U \quad (14)$$

$$(s_{ij}^m - 1)u_{ij}^{mt'} \leq \sum_{t \in T: d(t)=d(t'), t < t'} \psi_{tt'}^m, \quad \forall t' \in T, m \in M, (i, j) \in U \quad (15)$$

$$\sum_{t^3 \in T: t^3 \geq t, t^3 \neq t^2} \psi_{tt^3}^m + \sum_{t^3 \in T^M: t^3 \geq t} v_{tt^3}^m + \sum_{i \in I} z_{ti}^m + \psi_{t^1 t^2}^m \leq 1, \quad \forall t, t^1, t^2 \in T: t^1 \leq t \leq t^2, m \in M \quad (16)$$

$$\sum_{t \in T: d(t)=d} u_{ij}^{mt} \leq 1, \quad \forall d \in D, m \in M, (i, j) \in U \quad (17)$$

$$w_{ij}^{mt} \leq v_{tt}^m, \quad \forall t \in T^M, m \in M, (i, j) \in L \quad (18)$$

$$(s_{ij}^m - 1)w_{ij}^{mt'} \leq \sum_{t \in T^M: d(t)=d(t')} v_{tt'}^m, \quad \forall t' \in T^M, m \in M, (i, j) \in L \quad (19)$$

$$\sum_{t^3 \in T^M: t^3 \geq t, t^3 \neq t^2} v_{tt^3}^m + \sum_{t^3 \in T: t^3 \geq t} \psi_{tt^3}^m + \sum_{i \in I} z_{ti}^m + v_{t^1 t^2}^m \leq 1, \quad \forall t, t^1, t^2 \in T^M: t^1 \leq t \leq t^2, m \in M \quad (20)$$

$$\sum_{t \in T: d(t)=d} w_{ij}^{mt} \leq 1, \quad \forall d \in D, m \in M, (i, j) \in L \quad (21)$$

$$\sum_{m \in M} \sum_{t' \in T^M: t' \geq t} v_{tt'}^m + \sum_{m \in M} \mu_t^m \leq 1, \quad \forall t \in T^M \quad (22)$$

$$\sum_{t: d(t)=d} \mu_t^m = f^m \tau_d^m, \quad \forall d \in D, m \in M \quad (23)$$

$$\sum_{d=0}^{g_{max}^m - d_0^m} \tau_d^m \geq 1, \quad \forall m \in M \quad (24)$$

$$\sum_{d'=d-g_{max}^m}^d \tau_d^m \geq 1, \quad \forall d \in D: d \geq g_{max}^m, m \in M \quad (25)$$

$$\mu_{t^1}^m + \mu_{t^2}^m + \sum_{i \in I} z_{ti}^m \leq 2, \quad \forall t, t^1, t^2 \in T^M, m \in M$$

$$t^1 \leq t \leq t^2, d(t^1) = d(t^2) \quad (26)$$

$$z_{ti}^m \leq \rho_{ti}^m, \quad \forall t \in T, m \in M, i \in I \quad (27)$$

$$\sum_{i \in I} \rho_{ti}^m \leq 1, \quad \forall t \in T, m \in M \quad (28)$$

$$\rho_{ti}^m \leq \rho_{(t-1)i}^m + \sum_{j: (j,i) \in U} u_{ji}^{mt-1} + \sum_{j: (j,i) \in L} w_{ji}^{mt-1}, \quad \forall t \in T^M, m \in M, i \in I \quad (29)$$

$$\rho_{ti}^m \leq \rho_{(t-1)i}^m + \sum_{j: (j,i) \in U} u_{ji}^{mt-1}, \quad \forall t \in T \setminus T^M, m \in M, i \in I \quad (30)$$

$$\sum_{j: (i,j) \in U} u_{ij}^{mt} + \sum_{j: (i,j) \in L} w_{ij}^{mt} \leq \rho_{(t-1)i}^m, \quad \forall t \in T^M, m \in M, i \in I \quad (31)$$

$$\sum_{j: (i,j) \in U} u_{ij}^{mt} \leq \rho_{(t-1)i}^m, \quad \forall t \in T \setminus T^M, m \in M, i \in I \quad (32)$$

$$\Delta_{id} \leq S_i^{max}, \quad \forall d \in D, i \in I \quad (33)$$

$$\phi_{id}, \Delta_{id} \geq 0, \quad \forall d \in D, i \in I \quad (34)$$

$$\eta_d, \pi_d \in \{0, 1\}, \quad \forall d \in D \quad (35)$$

$$\tau_d^m \in \{0, 1\}, \quad \forall d \in D, m \in M \quad (36)$$

$$\theta_t \in \{0, 1\}, \quad \forall t \in T^O \quad (37)$$

$$y_t^m \in \{0, 1\}, \quad \forall t \in T \quad (38)$$

$$\mu_t^m \in \{0, 1\}, \quad \forall t \in T^M \quad (39)$$

$$z_{ti}^m, \rho_{ti}^m \in \{0, 1\}, \quad \forall t \in T, i \in I, m \in M \quad (40)$$

$$u_{ij}^{mt} \in \{0, 1\}, \quad \forall t \in T, (i, j) \in U, m \in M \quad (41)$$

$$w_{ij}^{mt} \in \{0, 1\}, \quad \forall t \in T^M, (i, j) \in L, m \in M \quad (42)$$

$$v_{tt'}^m \in \{0, 1\}, \quad \forall t, t' \in T^M : t' \geq t, m \in M \quad (43)$$

$$\gamma_r^t \in \{0, 1\}, \quad \forall r \in R, t \in T_r \quad (44)$$

Objective function (1) minimizes the sum of revenue loss from unserved requests, additional personnel costs (overtime, night shift and parallel operations) and penalties incurred by stock deficits. Constraints (2) ensure that if a block is used for overtime then there is no night shift that day and vice versa (meaning no overtime is possible if there is a night shift on a certain day). Constraints (3) forbid using isolated overtime blocks, in other words: overtime in a block is possible only if the previous overtime block is also used, except for the first overtime block, which can be used without preceding overtime blocks. Constraints (4) and (5) enforce the machines to be idle during night-shift blocks if there is no overtime or night shift used that day. Constraints (6) ensure that for a certain day with no night shift on the preceding day, night shifts are only allowed if there is a night shift on every single one of the following  $d_n - 1$  consecutive days. Similarly, Constraints (7) ensure that for a certain day with no night shift on the following day that night shifts are only allowed if there is a night shift on every single one of the preceding  $d_n - 1$  consecutive days. Constraints (8) and (9) ensure that during any block, a machine is either idle or occupied by a single operation, namely: setup (short or long), maintenance or production. Constraints (10) enforce a parallel processing penalty if more than one machine is not idle. Constraints (11) are the inventory (stock) balance constraints. Constraints (12) retrieve the daily stock deficit per item, if there is any. If the inventory level is greater than the minimum stock requirement, the deficit variable assumes a value of zero thanks to the objective function (1) and binary restrictions (34). By Constraints (13), at most one shipping is conducted per request. Constraints (14)-(16) ensure that no production is performed in between the blocks of the same short setup. Constraints (18)-(20) ensure that no production is performed in between the blocks of the same long setup. Constraints (17) and (21) ensure that only one type of setup is scheduled per day. Constraints (22) ensure that at most one long setup or maintenance takes place during any single block. Constraints (23)-(26) ensure that maintenance blocks are assigned with the required frequency while ensuring that no production is conducted in between the blocks of a maintenance. Constraints (27)-(30) guarantee that production of an item is only possible if the machine has the right configuration, which is validated by a previous block either with the identical configuration or via a completed setup to that item. Maximum stock levels are respected thanks to Constraints (33). Finally, (34)-(44) are nonnegativity and binary restrictions.