Solving the Production Leveling Problem with Order-Splitting and Resource Constraints

Johannes Vass · Nysret Musliu · Felix Winter

Abstract We investigate an extended problem formulation of the Production Leveling Problem (PLP), which was recently introduced in the literature. For the PLP problem the task is to assign orders to production periods such that the load is balanced, capacity limits are not exceeded and the order's priorities are considered. The extended problem (PLP-OSRC) introduced in this paper additionally includes order-splitting, resource constraints and due dates. We provide a mixed integer programming formulation for the PLP-OSRC based on the existing model for the PLP and evaluate it with a state-of-the-art MIP solver. To solve practically sized instances we apply a local search approach based on simulated annealing and propose two innovative neighborhood moves. We compare our approaches on two sets of randomly generated instances and show that the simulated annealing approach provides competitive results to MIP for the smaller instances. Moreover, it provides good solutions for very large instances that could not be solved by our MIP model in a reasonable amount of time.

Keywords Production Leveling, Simulated Annealing, MIP

1 Introduction

As many modern-day factories in the area of industrial manufacturing migrate towards full automation, a strong need for efficient automated production planning systems becomes more and more apparent. Although many known practical planning problems deal with short-term scheduling and long-term planning tasks, there is also an important need for mid-term planning systems

Johannes Vass, Nysret Musliu and Felix Winter
Christian Doppler Laboratory for Artificial Intelligence and Optimization for Planning and Scheduling
DBAI, TU Wien
E-mail: {jvass,musliu,winter}@dbai.tuwien.ac.at

that aim to efficiently distribute orders created by long-term planning systems into smaller short-term scheduling problems.

Recently, we have introduced such a mid-term planning problem called the production leveling problem (PLP) in [11,10]. The PLP takes a number of jobs as its input and aims to evenly distribute the production orders in an optimized plan over a given planning horizon. Finding a balanced distribution of the workload over the set of production periods is mainly motivated by the idea that solutions to the PLP will lead to improved short-term schedules that efficiently utilize the production capacities in each period and to encourage just-in-time manufacturing.

Applications for the PLP arise in different areas of the industry. For example, a practical application of the PLP has been deployed by our industry partners in electronic component manufacturing, where it is desired to assign a well balanced product mix to each production period. As setup costs that arise between the manufacturing of different product families are not too high in this case, a well balanced product mix leads to increased capacity utilization as well as decreased storage- and transport costs as just-in-time production in the sense of heijunka [3] is encouraged. Several practical instances for this application have been introduced and are available[1].

Leveling problems similar to the PLP have been studied in other application domains in the past like for example the balanced academic curriculum problem [2,4], nurse scheduling [7,9,8] or assembly line balancing [1]. However, in contrast to other leveling problems the PLP includes the consideration of order priorities in its objective function, which is of high importance for practical production planning problems.

Previously, we have shown that the PLP is NP-hard and have proposed metaheuristic approaches as well as an exact approach based on mixed integer programming in [11]. Although these existing solution methods can be used to approach practically sized instances of the problem, some real life mid-term planning scenarios cannot be tackled with the standard PLP problem formulation, as it does not consider the availability of resources used during production. Furthermore, the original specification of the PLP does not allow to split given orders into multiple sub orders, which can in some scenarios improve the quality of solutions.

In this paper, we therefore introduce an extended problem formulation of the PLP that supports order-splitting and the consideration of resource constraints called the production levelling problem with order-splitting and resource constraints (PLP-OSRC). In addition to providing a formal specification to the problem, we extend the mixed integer programming formulation for the PLP to model the PLP-OSRC, which allows us to approach instances of the extended problem with state-of-the-art mixed integer programming solvers. Furthermore, we propose two innovative neighborhood moves that we utilize within a metaheuristic approach based on simulated annealing to solve very large realistically sized instances of the PLP-OSRC. Finally, we implement

[1] `https://www.dbai.tuwien.ac.at/staff/jvass/production-leveling/`

all of our proposed methods and perform a large number of experiments to empirically evaluate the performance of the investigated solution approaches.

The remainder of the paper is structured as follows: Section 2 briefly reviews the formal problem specification of the PLP and then gives a detailed formal specification of the PLP-OSRC. Afterwards, we propose the integer programming formulation for the extended problem and describe the details absout the metaheuristic approach and the novel neighborhood moves in Section 3. In Section 4 we give an overview of all conducted experiments and discuss computational results before we give concluding remarks in Section 5.

2 Problem Statement

In this section, we provide a description of the production leveling problem with resource constraints and order-splitting (PLP-OSRC), which is a real-life industrial planning problem that is concerned with evenly distributing a set of orders over a planning horizon.

Recently, we introduced the production leveling problem (PLP) in [11]. The main differences between the PLP and the extended problem, that we describe in this paper, are that given orders are allowed to be split into multiple parts and that additional resource constraints can be defined for the PLP-OSRC.

In the following, we first review the problem description for the PLP in Section 2.1, before we later in Section 2.2 introduce the notion of order splits and additional constraints of the extended problem.

2.1 The Production Leveling Problem

The input to the PLP contains a list of orders to be distributed over the planning horizon, where each order defines a number of demanded items of a particular product type that need to be produced. Furthermore, the importance of each order is defined by a given priority value.

The goal of the problem is to assign each order to a single period in the given production horizon. A feasible solution needs to make sure that given maximum production volumes for each of the periods are not exceeded, where each period defines an overall maximum production volume and product type specific maximum production volumes.

The following lists the formal parameters and decision variables to an instance of the PLP:

Input Parameters

$K = \{1, \ldots, k\}$	Set of orders, where k is the number of orders
$M = \{1, \ldots, m\}$	Set of product types, where m is the number of product types
$N = \{1, \ldots, n\}$	Set of periods, where n is the number of periods
$c \in \mathbb{R}^+$	the maximum overall production volume per period
$c_t \in \mathbb{R}^+$	for each product type $t \in M$ the maximum production volume per period
$d_j \in \mathbb{Z}^+$	for each order $j \in K$ its associated demand
$p_j \in \mathbb{Z}^+$	for each order $j \in K$ its associated priority
$t_j \in \mathbb{Z}^+$	for each order $j \in K$ the product type
$d^* \in \mathbb{Z}^+$	the target production volume per period, i.e. $\frac{1}{n} \sum_{j \in K} d_j$
$d_t^* \in \mathbb{Z}^+$	the target production volume per period for each product type $t \in M$, i.e. $\frac{1}{n} \sum_{j \in K \mid t_j = t} d_j$

Variables

- A variable y_j for each order $j \in K$ determines in which period the order shall be produced:

$$y_j \in N \quad \forall j \in K$$

- The total production volume for each period is stored in auxiliary variables $w_i \quad \forall i \in N$:

$$w_i = \sum_{\substack{j \in K: \\ y_j = i}} d_j \quad \forall i \in N$$

- The total production volume for each product type and period is stored in auxiliary variables $w_{i,t} \quad \forall i \in N, t \in M$:

$$w_{i,t} = \sum_{\substack{j \in K: \\ y_j = i \wedge t_j = t}} d_j \quad \forall i \in N, t \in M$$

Hard Constraints

The following hard constraints impose restrictions on the maximum production volumes for each period in the planning horizon:

- H1: The limit for the overall production volume is satisfied for each period:

$$\forall i \in N \quad w_i \leq c$$

- H2: The limit for the production volume of each product type is satisfied for each period:

$$\forall i \in N, t \in M \quad w_{i,t} \leq c_p$$

Objective Function

A multi-objective function which includes three objectives determines the quality of solutions to the PLP. Intuitively, an optimized production plan should assign orders to periods such that the production volume is balanced between the periods while trying to adhere to the production sequence which is implied by the order's priorities.

Therefore, the objectives for the PLP are defined as follows:

1. Minimize the sum of deviations of the planned production volume to the average demand (i.e. the target value d^*) for each period, ignoring the product types.

$$f_1 = \sum_{i \in N} |d^* - w_i| \tag{1}$$

$$\tag{2}$$

2. Minimize the sum of deviations of the production volume of each product type to its respective mean target value d_t^*, making sure that the production of each product type is being leveled.

$$f_2 = \sum_{t \in M} \left(\frac{1}{d_t^*} \cdot \sum_{i \in N} |d_t^* - w_{i,t}| \right) \tag{3}$$

$$\tag{4}$$

3. Minimize the number of times a higher prioritized order is planned for a later period than a lower prioritized order, which we call a priority inversion. This objective makes sure that it costs less to plan the production of more important orders for earlier periods.

$$f_3 = \left| \left\{ (i,j) \in K^2 : y_i > y_j \text{ and } p_i > p_j \right\} \right| \tag{5}$$

In order to combine the objectives (f_1, f_2, f_3) into a single objective function each of the individual cost components is normalized as follows:

$$g_1 = \frac{1}{n \cdot d^*} \cdot f_1 \tag{6}$$

$$g_2 = \frac{1}{n \cdot m} \cdot f_2 \tag{7}$$

$$g_3 = \frac{2}{k \cdot (k-1)} \cdot f_3 \tag{8}$$

The normalization ensures that g_1 and g_2 stay between 0 and 1 with a high probability. Only for degenerated instances, where even in good solutions the target is exceeded by factors ≥ 2 higher values are possible for g_1 and g_2. The value of g_3 is guaranteed to be ≤ 1 because the maximum number of inversions in a permutation of length k is $k \cdot (k-1)/2$.

The final objective function of the PLP is a weighted sum of the three normalized objective functions, where user defined weights a_{1--3} determine the relative importance of each objective.

$$\textbf{minimize } g = a_1 \cdot g_1 \; + \; a_2 \cdot g_2 \; + \; a_3 \cdot g_3 \tag{9}$$

Figure 1 shows a small example instance of the PLP with five orders, which are shown as boxes, where the box height corresponds to the order size. The orders should be distributed over three production periods such that the distances of the stacks of orders and the dashed target line is minimized and no stack crosses the red line which represents the overall maximum production volume.

Similar to the first example, Figure 2 shows a solution for a small example instance with five orders. The numbers inside the orders this time determine the order priorities, where a larger number indicates a higher importance. In this case, it is undesirable that the red order is assigned to an earlier period than the yellow or blue order. Whenever a pair of two orders is not planned so that their priority values are descending over time they cause a priority inversion in the production plan. The third objective of the PLP aims to minimize the total number of priority inversions. In the example, a better solution could be obtained by swapping the red order with the yellow one, because it would eliminate both priority inversions.

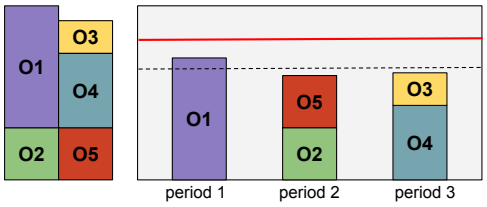

Fig. 1: Example visualizing the effects of the first objective f_1 that aims to minimize the total deviation to the target value (i.e. the dashed line). The solution shown in this example is optimal w.r.t. this objective.

2.2 The Production Leveling Problem with Order-Splitting and Resource Constraints

The PLP as we described it in the previous section appears in many real-life industrial applications where production volumes are ought to be leveled evenly over the planning horizon to keep the production process robust and efficient. However, as it assumes that each order is indivisible, it cannot be used in any practical context where single customer orders can actually be distributed over

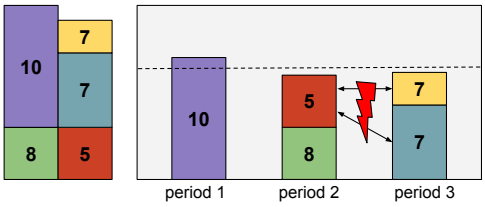

Fig. 2: Example solution that contains two priority inversions. The numbers in each box determine the priority value of the associated order.

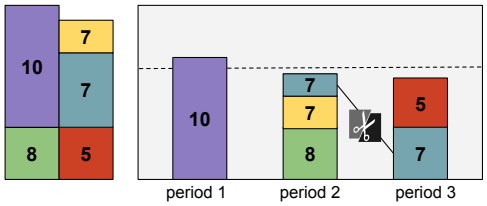

Fig. 3: Example solution, where the blue order (with priority 7) is split between period 2 and 3. This way, we do not introduce any priority inversion and nevertheless get a similarly good leveling as in the example before.

multiple periods in the planning horizon. Figure 3 shows an example why splitting orders can be sometimes useful: The additional flexibility that is obtained by allowing orders to be split can help to create solutions that are good both in terms of levelness and prioritization. Another shortcoming of the basic PLP in practical environments is the absence of resource constraints, which prevents us, for example, to take staffing and the availability of tools or machines into account while distributing orders over production periods.

In this section, we therefore introduce a novel extension of the production leveling problem that we call the production leveling problem with order-splitting and resource constraints (PLP-OSRC). Instances to the PLP-OSRC define the same parameters as specified for the PLP in Section 2.1, but include additional parameters that determine how many splits per order are feasible and provide the parameters about the additional constraints. The following lists the additional input parameters for the PLP-OSRC:

Input Parameters

$R = \{1, \ldots, o\}$	Set of secondary resources where o is the number of secondary resources
$pd_j^{\min} \in N$	the earliest period to which an order $j \in K$ can be assigned without a penalty
$pd_j^{\max} \in N$	the latest period to which an order $j \in K$ can be assigned without a penalty
$ps_j^{\min} \in \{1, \ldots, d_j\}$	the minimum size of any partition of order $j \in K$
$pc_j^{\max} \in \mathbb{Z}^+$	the maximum number of partitions of order $j \in K$
$ru_{j,r} \in \mathbb{R}_0^+$	the amount of usage of secondary resource $r \in R$ by order $j \in K$
$ru_r^{\min} \in \mathbb{R}^+$	the minimum penalty-free usage of resource $r \in R$ in each period
$ru_r^{\max} \in \mathbb{R}^+$	the maximum penalty-free usage of resource $r \in R$ in each period

Since a single order can be split and planned into multiple periods, the decision variables of the PLP-OSRC have to capture additional information than for the PLP. The following list defines the variables for the PLP-OSRC:

Variables

- Variables $x_{i,j}$ determine the amount of order j which is planned to be produced in period i. If a variable $x_{i,j} > 0$, we say that a partition of order j is planned in period i.

$$x_{i,j} \in \mathbb{Z}_0^+ \quad \forall i \in N, j \in K$$

- Auxiliary variables y_j^{start} and y_j^{end} determine the periods where the first and last partition of an order j are planned:

$$y_j^{start} = \min(\{i \in N \mid x_{i,j} > 0\}) \quad \forall j \in K$$

$$y_j^{end} = \max(\{i \in N \mid x_{i,j} > 0\}) \quad \forall j \in K$$

- The production volume for each period is stored in auxiliary variables w_i:

$$w_i = \sum_{j \in K} x_{i,j} \quad \forall i \in N$$

- The production volume for each product type and period is stored in auxiliary variables $w_{i,t}$:

$$w_{i,t} = \sum_{\substack{j \in K: \\ t_j = t}} x_{i,j} \quad \forall i \in N, \forall t \in M$$

- Auxiliary variables $u_{i,r}$ capture the total usage of secondary resources in each of the planning periods, where the resource usage of a single order partition is determined relative to the total order usage.

$$u_{i,r} \in \mathbb{R}_0^+ = \sum_{j \in K} ru_{j,r} \cdot \frac{x_{i,j}}{d_j} \quad \forall i \in N, r \in R$$

Hard Constraints

In addition to the two production volume hard constraints from the PLP, the PLP-OSRC defines another two hard constraints that restrict the minimum partition size and the maximum partition count of each order:

- H3: The minimum partition size is reached for every partition of every order:

$$\forall i \in N, j \in K \quad x_{i,j} = 0 \ \vee \ x_{i,j} \geq ps_j^{\min}$$

- H4: The maximum partition count is not exceeded for any order:

$$j \in K \quad |\{i \in N \mid x_{i,j} > 0\}| \leq pc_j^{\max}$$

Objective Function

The three objectives f_1, f_2, f_3 defined in Section 2.1 are also used in the multi-objective function of the PLP-OSRC. However, f_3 has to be slightly reformulated to be compatible with the novel variable definitions. Furthermore, two new objectives f_4 and f_5 influence the quality of solutions to the PLP-OSRC depending on the earliness/lateness of orders and the over- and under-utilization of resources.

The following defines objectives f_3, f_4, f_5 for the PLP-OSRC:

- Function f_3 counts the number of priority inversions in the assignment where they are redefined to handle order splits. That is, f_3 counts the number of order-pairs (i, j) for which i has a higher priority than j but i finishes only after j starts.

$$f_3 = \left| \left\{ (i,j) \in K^2 : p_i > p_j \text{ and } y_i^{end} > y_j^{start} \right\} \right| \tag{10}$$

$$\tag{11}$$

- The objective function f_4 calculates a penalty for every order whose first partition is planned before the order's minimum period or whose last partition is planned after the order's maximum period.

$$f_4 = \sum_{j \in K} \left(\max(pd_j^{\min} - y_j^{start}, 0) \ + \ \max(y_j^{end} - pd_j^{\max}, 0) \right) \tag{12}$$

$$\tag{13}$$

- Objective f_5 calculates a penalty for over-usage and under-usage of secondary resources.

$$f_5 = \sum_{r \in R} \sum_{i \in N} \begin{cases} 1 - \frac{u_{i,r}}{ru_r^{\min}} & \text{if } u_{i,r} < ru_r^{\min} \\ \frac{u_{i,r}}{ru_r^{\max}} - 1 & \text{if } u_{i,r} > ru_r^{\max} \\ 0 & \text{otherwise.} \end{cases} \tag{14}$$

Whereas objectives f_1, f_2, f_3 can be normalized as specified in Section 2.1, objectives f_4 and f_5 are normalized as follows:

$$g_4 = \frac{1}{n \cdot k} \cdot f_4 \tag{15}$$

$$g_5 = \frac{1}{n \cdot o} \cdot f_5 \tag{16}$$

$$\tag{17}$$

Objective g_4 applies normalization through a division by the number of orders times the number of periods. As the penalty for each order is at most k, the normalized objective is also guaranteed to be ≤ 1. The resource objective g_5 is normalized by the number of periods and resources which normally also leads to values between 0 and 1, however the upper bound is not strict. When looking at instances without secondary resources, g_5 must not be considered in the objective function because it would yield a division by zero.

The final objective function for the PLP-OSRC is the following weighted sum (with user defined weights $a_1 - a_5$).

$$\textbf{minimize } g = a_1 \cdot g_1 \ + \ a_2 \cdot g_2 \ + \ a_3 \cdot g_3 + \ a_4 \cdot g_4 + \ a_5 \cdot g_5 \tag{18}$$

3 Solution Approaches

In the previous section we have provided an in depth problem definition of the PLP-OSRC. In this section, we first propose an integer programming formulation of the problem in Section 3.1 before we later describe a metaheuristic local search based solution approach in Section 3.2

3.1 Integer Programming Model

Previously, we proposed an integer programming model for the PLP [11]. In this section we extend that model for the PLP-OSRC, based on the formal problem description that we specified in Section 2. We carry over the input parameters without changes, which is why they are not repeated in this section. In contrast to the integer programming model for the PLP, th e The former binary decision variables x_{ij} are converted to an integer domain in order to model the planned production amount for order j in period i and thus account for order splits. Furthermore, the model for the PLP-OSRC The variables and detailed formulation are as follows:

Variables

$x_{ij} \in \mathbb{Z}^+$	for each $i \in N$, $j \in K$ stating how much demand of order j is planned in period i
$\hat{x}_{ij} \in \{0,1\}$	for each $i \in N$, $j \in K$ stating whether a partition exists in period i. A partition exists for order j in period i iff $x_{ij} > 0$.
$y_j^{start} \in N$	for each order $j \in K$ the period assignment of its first partition
$y_j^{end} \in N$	for each order $j \in K$ the period assignment of its last partition
$z_{ij} \in \{0,1\}$	for orders $i, j \in K$ where $p_i > p_j$, existence of a priority inversion between i and j
$s_i^+ \in \mathbb{R}^+$	for each $i \in N$ the surplus production volume for period i
$s_i^- \in \mathbb{R}^+$	for each $i \in N$ the missing production volume for period i
$s_{it}^+ \in \mathbb{R}^+$	for each $i \in N$, $t \in M$ the surplus production volume for period i and product type t
$s_{it}^- \in \mathbb{R}^+$	for each $i \in N$, $t \in M$ the missing production volume for period i and product type t
$u_{ir}^+ \in \mathbb{R}^+$	for each $i \in N$, $r \in R$ the amount of over-usage of resource r in period i
$u_{ir}^- \in \mathbb{R}^+$	for each $i \in N$, $r \in R$ the amount of under-usage of resource r in period i
$v_j^{start} \in \mathbb{Z}^+$	for each $j \in K$ the amount of violation of the earliest period soft constraint
$v_j^{end} \in \mathbb{Z}^+$	for each $j \in K$ the amount of violation of the latest period ($\widehat{=}$ due date) soft constraint

Formulation

$$\min \quad a_1 g_1 + a_2 g_2 + a_3 g_3 + a_4 g_4 + a_5 g_5 \tag{19}$$

s.t.

$$\sum_{i \in N} x_{ij} = d_j \qquad j \in K \tag{20}$$

$$x_{ij} \leq d_j \cdot \hat{x}_{ij} \qquad j \in K \tag{21}$$

$$y_j^{start} \leq i + (n-1) \cdot (1 - \hat{x}_{ij}) \qquad j \in K, i \in N \tag{22}$$

$$y_j^{end} \geq i \cdot \hat{x}_{ij} \qquad j \in K, i \in N \tag{23}$$

$$\sum_{j \in K} x_{ij} + s_i^+ - s_i^- = d^* \qquad i \in N \tag{24}$$

$$\sum_{j \in K | t_j = t} x_{ij} + s_{it}^+ - s_{it}^- = d_t^* \qquad i \in N,\ t \in M \tag{25}$$

$$d^* + s_i^+ \leq c \qquad i \in N \tag{26}$$

$$d_t^* + s_{it}^+ \leq c_t \qquad i \in N,\ t \in M \tag{27}$$

$$x_{ij} \geq ps_j^{\min} \cdot \hat{x}_{ij} \qquad j \in K \tag{28}$$

$$\sum_{i \in N} \hat{x}_{ij} \leq pc_j^{\max} \qquad j \in K \tag{29}$$

$$y_i^{end} - y_j^{start} \leq (n-1) z_{ij} \qquad i, j \in K \mid p_i > p_j \tag{30}$$

$$v_j^{start} \geq pd_j^{\min} - y_j^{start} \qquad j \in K \tag{31}$$

$$v_j^{end} \geq y_j^{end} - pd_j^{\max} \qquad j \in K \tag{32}$$

$$\sum_{j \in K} ru_{j,r} \cdot \frac{x_{ij}}{d_j} - u_{ir}^+ \leq ru_r^{\max} \qquad i \in N, r \in R \tag{33}$$

$$\sum_{j \in K} ru_{j,r} \cdot \frac{x_{ij}}{d_j} + u_{ir}^- \geq ru_r^{\min} \qquad i \in N, r \in R \tag{34}$$

$$y_i^{end} \leq y_j^{start} \qquad i, j \in S, S \subseteq K \mid p_i \geq p_j, d_i = d_j, t_i = t_j \tag{35}$$

$$\sum_{t \in M} (s_{it}^- - s_{it}^+) = s_i^- - s_i^+ \qquad i \in N \tag{36}$$

$$g_1 = \frac{1}{n \cdot d^*} \cdot \sum_{i \in N} (s_i^+ + s_i^-) \tag{37}$$

$$g_2 = \frac{1}{n \cdot m} \cdot \sum_{t \in M} \left(\frac{1}{d_t^*} \cdot \sum_{i \in N} (s_{it}^+ + s_{it}^-) \right) \tag{38}$$

$$g_3 = \frac{2}{k \cdot (k-1)} \cdot \sum_{i,j \in K} z_{i,j} \tag{39}$$

$$g_4 = \frac{1}{2k} \cdot \sum_{j \in K} \left(v_j^{start} + v_j^{end} \right) \tag{40}$$

$$g_5 = \frac{1}{n \cdot o} \cdot \sum_{r \in R} \sum_{i \in N} \left(\frac{u_{ir}^-}{ru_r^{\min}} + \frac{u_{ir}^+}{ru_r^{\max}} \right) \tag{41}$$

Constraints (20) to (25) link auxiliary variables to the decision variables. Constraint (20) ensures that the total demand of all partitions of j equals the order's demand d_j. Constraint (21) links the binary variables $\hat{x}_{ij}$ to the decision variables x_{ij} such that $x_{ij} > 0 \rightarrow \hat{x}_{ij} = 1$. Constraints (22) and (23) link the x_{ij} to the y_i^{start} and y_i^{end} variables so that they always hold the period assignment of the first and last period of any partition of an order, respectively. Constraint (24) states for each period that the total planned production volume plus the surplus minus the slack equals the target d^*. As both variables have positive domains and they are subject to minimization, at most one of them will be non-zero in any optimal solution. Constraint (25) repeats this relationship over the variables s_{it}^+ and s_{it}^- for each product type t.

The block of constraints between (26) and (29) models the problem's hard constraints. Constraint (26) ensures that the capacity bound per period is satisfied by enforcing that the sum of target demand d^* and the surplus variable s^+ does not exceed the threshold. Analogously, Constraint (27) enforces the capacity limit per period and product type. Constraint (28) enforces the minimum partition size and (29) the maximum number of partitions into which an order may be split.

Constraints (30) to (34) populate penalty variables for the objective function. Constraint (30) links the y_i^{start} and y_i^{end} to the $z_{i,j}$ variables which track the number of priority inversions. It makes sure that for every pair of orders i, j where i has a higher priority than j, z_{ij} is 1 (representing a priority inversion) if order i finishes after order j starts. The constraints (31) and (32) force the variables v_j^{start} and v_j^{end} to keep track of the violations of the allowed assignment range. Constraints (33) and (34) force u_{ir}^+ and u_{ir}^- to contain the amount of over-usage and under-usage of resource r in period i. That is achieved by comparing with the amount of planned resource usage which is given by the first summand.

Finally, there are two redundant constraints for strengthening the formulation: Constraint (35) enforces a dominance relation for all pairs of orders which have the same product type and demand value. The constraint requires that the higher prioritized order ends not later than the lower prioritized one starts, which is sensible because otherwise we could swap the two orders to obtain a better solution. This cuts off parts of the search space where the optimal solution cannot reside. Constraint (36) links the $s_i^{\{+,-\}}$ and $s_{it}^{\{+,-\}}$ variables together, which also leads to improvements in the average run-time.

The objective function is equivalent to the one presented in Section 2.2 but here it is stated on the variable set of the MIP formulation. In g_1, the sum of the slack and surplus variable $(s_i^+ + s_i^-)$ is equivalent to the absolute difference between planned demand and target demand $|d^* - w_i|$, because at least one of s_i^+ and s_i^- will be 0 in any optimal solution and the other one holds the absolute difference. The same holds true for the analogous variables in g_2. Similarly, instead of using one variable per period to track the resource usage, the two separate variables for over-usage and under-usage u_{ir}^+ and u_{ir}^- are used to compute the cost component g_5.

3.2 Metaheuristic Approach

The IP formulation we proposed in the previous section can be used to solve instances of the PLP-OSRC to optimality. However, solving realistically sized instances with a large number of variables may often not be possible for an IP solver within reasonable run-time and memory limitations. Therefore, we additionally propose a local search based metaheuristic solution approach to the PLP-OSRC in this section.

Previously, we have investigated a construction heuristic and a local search approach using simulated annealing for the PLP in [11] and proposed basic neighborhood moves that can reposition single orders and swap pairs of orders.

In this section, we extend the two neighborhoods previously proposed for the PLP and furthermore propose two innovative search neighborhoods for the PLP-OSRC

3.2.1 Neighborhood Relations

We propose four neighborhood relations when approaching the PLP-OSRC with local search: The first two are extensions of the search neighborhoods to the PLP that have been previously proposed in [11]. Basically, instead of moving and swapping complete orders, the new versions move and swap single order partitions. The third neighborhood splits and merges order partitions and is able to exchange demands between partitions of an order. Finally, the fourth neighborhood shifts all partitions of a single order at once.

In the following, we describe the neighborhood operators in detail:

Move-Partition Neighborhood The move-partition neighborhood of a solution s consists of all solutions s' whose only difference to s is that one partition of some order has been moved to a different period. When splits are disallowed and thus every order has exactly one partition, this is equivalent to the move-order neighborhood of the basic PLP version. When generating random moves for this neighborhood, we uniformly sample the order and the partition as well as the target period.

Swap-Partitions Neighborhood The swap-partitions neighborhood of a solution s consists of all solutions s' whose only difference to s is that two order partitions of different orders, which are not assigned to the same period in s appear with swapped period assignments in s'. In random neighborhood generation, the partitions to be swapped are chosen uniformly at random among all pairs of partitions of different orders, which are not already assigned to the same period.

Split Neighborhood The split neighborhood of a solution s consists of all solutions s' which differ from s only with respect to one order, where the possible changes are:

- One partition of that order is split into two parts and the new part is moved to some other period.
- Two partitions of that order are merged.
- The sizes of two partitions of that order are changed such that the total size of the two partitions together stays the same and both partitions remain non-empty. In other words, demands are moved from one partition to another, which is potentially in a different period.

When sampling split moves randomly, first any single order is chosen randomly. Then, the move is generated such that the three options described above are equally likely.

Shift-order neighborhood The shift-order neighborhood of a solution s consists of all solutions s' which differ from s only with respect to a single order, whose partitions all have been shifted i periods to the left or to the right. If a partition cannot be shifted any more because it is already in the first or last period, it remains in that period. When sampling random shift-order moves, an order is chosen uniformly at random and i is chosen from $\{-1, 1\}$.

3.2.2 Simulated Annealing

Similar as previously described for the PLP in [11], we use a simulated annealing metaheuristic to approach the PLP-OSRC in this paper. Simulated annealing was first introduced in [5] and resembles the physical process of annealing in metallurgy. The basic idea is to iteratively apply randomly generated neighborhood moves to an initial solution to the problem. Whether a neighborhood move is accepted, depends on the resulting solution quality and a temperature parameter which is cooled down over the course of the search process.

Algorithm 1 shows the detailed simulated annealing procedure:

The acceptance function uses the metropolis criterion [5], where the probability $P(i \Rightarrow j)$ to accept a move from solution i to solution j is defined as follows ($f(x)$ is the objective function):

$$P(i \Rightarrow j) = \begin{cases} 1, & \text{if } f(j) \leq f(i). \\ \exp\left(\frac{f(i)-f(j)}{t}\right), & \text{otherwise.} \end{cases} \tag{42}$$

We further use a geometric cool down scheme:

$$t_i = \alpha \cdot t_{i-1} \tag{43}$$

The simulated annealing procedure shown in 1 relies on a number of parameters and an initial solution. We use the construction heuristic that was previously proposed in [11] to produce an initial solution, and set $\mathcal{N}_{1-4}$ to the search neighborhoods we proposed in this section. The remaining parameters have to be carefully selected depending on the computational environment. We describe the tuning of these parameters in Section 4.

Algorithm 1: Simulated Annealing

Data: *initialSolution*, neighborhoods $\mathcal{N}_i$ with probabilities p_i, $t_{\max}$, $t_{\min}$, iterations per temperature w, *timeLimit*, *iterationLimit*
Result: a solution at least as good as *initialSolution*

```
currentSolution ← initialSolution;
bestSolution ← currentSolution;
t ← t_max;
while t ≥ t_min and ¬ time limit reached and ¬ iteration limit reached do
    foreach j ∈ 1, ..., w do
        N ← choose one of neighborhoods N_i according to probabilities p_i;
        m ← select a random move out of N(currentSolution);
        if Accept(m, t) then
            currentSolution ← Apply(m, currentSolution);
            if currentSolution is better than bestSolution then
                bestSolution ← currentSolution;
            end
        end
    end
    t ← Cool-Down(t);
end
return bestSolution;
```

4 Experimental Evaluation

In this section, we provide a detailed description of our experimental environment and give an overview of our conducted experiments. We first describe an instance generator for the PLP-OSRC in 4.1, that we use to generate a large number of benchmark instances for our experiments. Later in Section 4.2, we provide the details on how parameters for the simulated annealing algorithm have been selected. Finally, we describe our computational environment and discuss the final experimental results in sections 4.3 and 4.4.

4.1 Instance Generation

To generate instances for the PLP-OSRC, we extend the random instance generator we previously proposed in [11] for the PLP with additional input parameters regarding partitioning, resources and due dates.

Given a number of orders k, periods n, product types m and resources o, the heuristic constructs an initial solution with the following steps:

1. Partition the number of orders k into m parts $o_1 \dots o_m$, one for each product type.
2. Randomly choose the maximum priority of all orders $p_{\max} \in \{1, \dots, 10\}$
3. For each product $t \in M$, create a set of demands D_t and randomly select the size of the set between 1 to 50. Then, insert the corresponding number of items into the set, where each item is a randomly selected value $d \in \{1, \dots, random(1000-5000)\}$ (the upper bound is a random value between 1000–5000 which is recalculated for each product type).

4. To determine resource usages for each product type, first randomly select a probability between 0.0 and 1.0. Then, based on this probability decide for each resource whether or not it is used for the product type. If it turns out a resource is used for the product type, randomly select a usage value between 0.0 and 1.0. This usage value then determines the resource usage per unit of demand for the particular product type.
5. For each product $t \in M$, generate o_t orders, where each order is generated as follows:
   (a) Randomly choose the demand d_j from the set D_t.
   (b) Randomly choose the priority from $\{1, \ldots, p_{\max}\}$.
   (c) The usage of each resource is calculated by multiplying the order demand d_j with the previously chosen usage factor. The resulting value is rounded up.
   (d) Randomly choose the earliest start $pd_j^{\min}$ from $\{1, \ldots, \lfloor 3/4 \cdot n \rfloor\}$.
   (e) Randomly choose the latest end $pd_j^{\max}$ from $\{pd_j^{\min} + 1, \ldots, n\}$.
   (f) Randomly choose the maximum partition count $pc_j^{\max}$ from $\{1, \ldots, 10\}$.
   (g) Randomly choose the minimum partition size $ps_j^{\min}$ from $\{1, \ldots, \lceil 1/2 \cdot d_j \rceil\}$[2]
6. In order to set the capacity limit c, we first calculate the target demand d^* as $\sum_{j \in K} {}^{d_j}/_n$. The capacity limit c is then derived from the d^* by multiplying with a random value from the normal distribution $\sigma(1.1, 0.02)$. Hence, c is in the expected case 10% larger than d^*. The capacity limits c_t for $t \in M$ are chosen analogously.
7. The minimum and maximum resource usages per period are calculated similarly: First, for each resource r, the average usage per period $\bar{u}_r$ is calculated. Then, the maximum deviation percentage $d_{\max}$ is drawn from the normal distribution $\sigma(0.1, 0.02)$, i.e. 10% on average. Finally, the minimum resource usage $ru_r^{\min}$ is set to $(1 - d_{\max}) \cdot \bar{u}_r$ and the maximum resource usage $ru_r^{\max}$ is set to $(1 + d_{\max}) \cdot \bar{u}_r$.

Using the instance generation procedure, we generated a total of 986 realistically sized large instances for our experiments. The following parameters were sampled uniformly at random: The number of orders k is chosen from $100 \ldots 4000$, the number of periods n from $2 \ldots 80$, the number of products m from $1 \ldots 20$ and the number of resources o from $1 \ldots 5$.

Additionally, we generated another set of 20 smaller instances, using the following random parameters: The number of orders k is chosen from $10 \ldots 100$, the number of periods n from $5 \ldots 10$, the number of products m from $1 \ldots 3$ and the number of resources o from $0 \ldots 3$.

[2] Note that an order can only be split in two parts if the demand is at least twice as large as the minimum partition size. Therefore, $ps_j^{\min}$ is chosen so that splitting is possible.

4.2 Parameter Tuning

As previously mentioned in Section 3.2, the simulated annealing algorithm we use in this paper depends on a number of parameters whose setting has an influence on the algorithm's efficiency and effectiveness. We configure the parameters for our experiments using SMAC, an automatic algorithm configuration tool that relies on Bayesian Optimization in combination with an aggressive racing mechanism in order to efficiently search through parameter configuration spaces [6].

We applied SMAC in parallel mode using 24 cores and a total time limit of 96 hours on the large set including 986 instances. For each problem instance, we specified a time limit of five minutes per run and no iteration limit. The cooling rate was not tuned but set to a value of 0.95, which however does not restrict the parameterization of the simulated annealing procedure as the number of iterations per temperature is still being tuned.

We tuned the initial temperature $t_{\max}$, the minimum temperature $t_{\min}$, the number of iterations per temperature w and a weight for each of the four neighborhood relations (p_{1-4}) which determines how often it is selected for the next move. The detailed configuration space with minimum and maximum values as well as the defaults and the tuning result is shown in Table 1.

Table 1: Configuration space of Simulated Annealing for parameter tuning

Parameter	**Type**	**Minimum**	**Maximum**	**Default**	**Tuned**
Initial Temperature	real	0.01	10.0	1	6.2
Minimum Temperature	real	10^{-9}	10^{-3}	10^{-6}	$7.6 \cdot 10^{-4}$
Iterations Per Temperature	integer	10^3	10^6	10^3	$1.54788 \cdot 10^5$
Move Partition Neighborhood Weight	integer	0	10	1	0
Swap Partitions Neighborhood Weight	integer	0	10	1	7
Split Order Neighborhood Weight	integer	0	10	1	3
Shift Order Neighborhood Weight	integer	0	10	1	0
Cooling Rate	real (fixed)	0.95	0.95	0.95	0.95

Note that the automatically tuned parameters set the weights for the move partition and shift order neighborhoods to 0, which disables these two neighborhood operators. We therefore evaluated in addition to the automatically tuned algorithm parameters (C_1) also two manually selected parameters configuration that include all neighborhood operators. One that we selected based on manual tuning with a number of conducted benchmarks (C_2) and another one that uses an equal weight for all four neighborhood operators (C_3). Table 2

shows the details about the three parameter configurations evaluated in our experiments.

Table 2: Overview of the algorithm parameter configurations used for experimental evaluation

Parameter	C_1**: auto tuned**	C_2**: manually tuned**	C_3**: equal weights**
Initial temperature	6.2	0.01	0.1
Minimum temperature	0.00076	10^{-9}	10^{-9}
Iterations per temperature	154788	300000	300000
Move Partition weight	0	4	2.5
Swap Partitions weight	7	2	2.5
Split Order weight	3	3	2.5
Shift Order weight	0	1	2.5
Cooling Rate	real (fixed)	0.95	0.95

4.3 Computational Environment

We conducted all experiments for this paper (including parameter tuning) on a computing cluster with 10 identical nodes, each having 24 cores, an Intel(R) Xeon(R) CPU E5–2650 v4 @ 2.20GHz and 252 GB of memory, running Ubuntu 16.04.1 LTS. Experiments with the proposed MIP model have been conducted using Gurobi 8.1.

4.4 Computational Results

In a first series of experiments we evaluated the performance of all investigated methods on the instance set which contains 20 small randomly generated instances. To give the MIP approach sufficient time to prove optimal solutions, we set the time limit for all experiments to 1 hour. The simulated annealing algorithm was run under the same time limit with each of the three parameter configurations (C_1, C_2, C_3) on the instances. We performed 10 repeated runs with every configuration on each instance, and used the median objective value from the 10 runs to compare the final results between the different methods.

Table 3 gives an overview of the experimental results with the small instance set. The first row of the table shows the number of instances where the evaluated methods could produce feasible solutions within the time limit, whereas the second row counts the number of overall best upper bounds achieved by each method. Finally, the third row displays the number of optimal solutions found. We can see that all methods were able to produce feasible solutions for every instance. The exact approach using the Gurobi solver produced the best results for the majority of the instances, followed by simulated

annealing with the manually tuned and automatically tuned parameter configurations. Gurobi was able to prove optimal solutions for four of the instances, while simulated annealing was able to reach one optimal solution.

	Gurobi	SA C_1	SA C_2	SA C_3
# solved	20	20	20	20
# best	13	3	4	0
# optimal	4	0	1	1

Table 3: Summarized results for the experiments with the set of smaller instances. The rows of the table display, from top to bottom, the number of feasible solutions found, the number of best upper bounds produced, and the number of optimal solutions achieved by each method.

Detailed results for experiments with the small instances are visualized in Figure 4. In addition to the achieved results by Gurobi and the simulated annealing approach the figure also displays the best lower bounds found by the mixed integer programming approach. All objective values are shown relative to the overall best found objective value, and therefore costs of 1 denote the overall best found solution costs (results with a lower bound value of 1 denote proven optimal solutions).

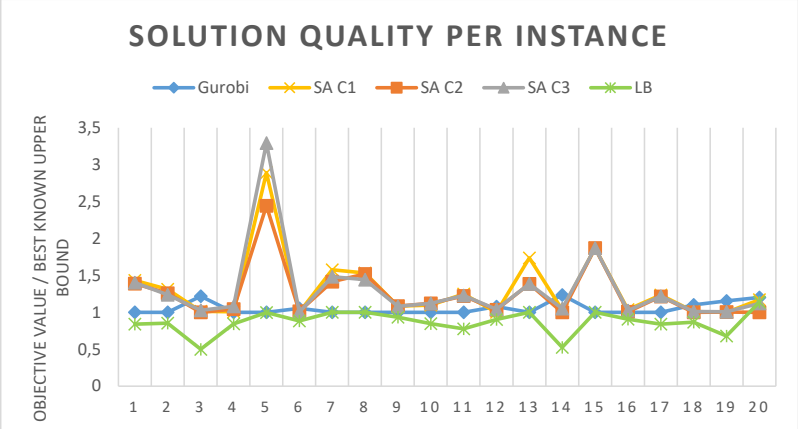

Fig. 4: A visualization of the experimental results for the 20 small instances. The horizontal axis represents the 20 evaluated instances, whereas the vertical axis measures the achieved relative objective values (solution cost produced by each method divided by the overall best found solution cost).

We can see that for the majority of the instances, the exact approach produces the best results. The simulated annealing approach produces similar

outcomes with all three evaluated parameter configurations, however the best results are produced with the manually tuned parameter configuration for the set of smaller instances in a few cases. Compared to the exact method, the simulated annealing algorithm can provide a similar solution quality on the majority of the instances, except for instances 5, 7, 8, 13, and 15, where Gurobi is able to produce the best results.

Figure 5 further visualizes the summarized results as box plots. We can see that Gurobi produces the overall best results for the set of small instances. All three parameter configurations for simulated annealing give similar results, with configurations C_2 and C_3 producing slightly better results than configuration $C1$.

Fig. 5: Box plots comparing the overall results achieved on the set of small instances. The vertical axis measures the achieved relative objective values (solution cost produced by each method divided by the overall best found solution cost).

In a second series of experiments we evaluated the performance of the proposed methods on the instance set which contains 986 large randomly generated instances. Similar as with the first series of experiments we conducted 10 repeated runs for each simulated annealing parameter configuration per instance and used the median objective value to compare the results between the evaluated methods. We used a five minute time limit for the set of larger instances.

The results for the experiments with the set of large instances are summarized in Table 4. Similar as in Table 3, the first row of the table shows the number of instances where a feasible solution could be found, the second row counts the number of best upper bounds found by each method, and the third row displays the number of proven optimal solutions.

	Gurobi	**SA** C_1	**SA** C_2	**SA** C_3
# solved	30	927	936	939
# best	3	758	95	95
# optimal	0	0	0	0

Table 4: Summarized results for the experiments with the set of larger instances. The rows of the table display, from top to bottom, the number of feasible solutions found, the number of best upper bounds produced, and the number of optimal solutions achieved by each method.

The results show that no approach is able to produce feasible solutions for all 986 instances within the time limit[3]. The exact method using Gurobi could only obtain 30 feasible solutions and three best upper bounds, whereas the simulated annealing approach is able to solve the large majority of instances in our experiments. We can see that simulated annealing with parameter configurations $C2$ and $C3$ was able to obtain slightly larger number of feasible solutions as $C1$. However, most best solutions was produced using parameter configuration $C1$. No optimality proofs could be achieved within the given time limits.

Figure 6 visually compares the produced solution qualities achieved by simulated annealing with parameter configurations C_1, C_2 and C_3 for the instances that could be solved by all three configurations. One can see that overall that the automatically tuned algorithm configuration C_1 overall produces the best results in our experiments whereas configurations C_2 and C_3 produce solutions of similar quality.

In summary, our experiments show that the exact approach obtains the best results for most of the small instances. However, in experiments with the larger instances the integer programming solver turned out to be not competitive compared to the simulated annealing approach. Overall, the three evaluated parameter configurations for simulated annealing produced a very similar number of feasible solutions, but the automatically tuned configuration that only uses the swap and split neighborhood operators produced the best results for the majority of the larger instances. However, we observe that configurations which make use of all four investigated neighborhood operators produced slightly better results on the set of smaller instances in our experiments.

5 Conclusion

In this paper we have investigated an extended problem formulation of the PLP that allows production orders to be split during the planning process and additionally considers the resource constraints. We provided a detailed formal specification of the extended production leveling problem and further

[3] There is no guarantee, though, that every instance actually has a feasible solution.

Fig. 6: Box plots comparing the overall results achieved with the simulated annealing approach on the set of larger instances. The vertical axis measures the achieved relative objective values (solution cost produced by each method divided by the overall best found solution cost). Note that some outliers have been excluded for an improved visual comparison.

proposed a mixed integer programming formulation that can be used to approach the problem with state-of-the-art solver technology. Additionally, we described a metaheuristic approach using simulated annealing that can be used to tackle realistically sized problem instances and investigated several local search neighborhood relations for the problem.

We empirically evaluated all proposed methods by performing experiments using a large number of instances that have been randomly generated by an instance generation routine that we proposed in this paper. Experimental results show that the exact approach using integer programming formulation was able to prove optimal results on several of the considered smaller instances and overall produced the best results for the experiments with small sized instances. However, results obtained by experiments with larger problem instances revealed that the exact approach was not competitive compared to the evaluated metaheuristics on realistically sized instances in our experiments. The simulated annealing based approach finds feasible solutions for most of the instances within a reasonable time limit and can be used to solve instances of realistic size. Based on the configuration provided by the automated parameter tuner SMAC, we can conclude that the most important neighborhoods are the swap partitions and split moves.

In future work, we plan to investigate an approach that hybridizes the proposed exact and metaheuristic techniques within the framework of large neighborhood search.

Acknowledgements The financial support by the Austrian Federal Ministry for Digital and Economic Affairs, the National Foundation for Research, Technology and Development and the Christian Doppler Research Association is gratefully acknowledged.

References

1. Azizoğlu, M., İmat, S.: Workload smoothing in simple assembly line balancing. Computers & Operations Research **89**, 51–57 (2018)
2. Chiarandini, M., Di Gaspero, L., Gualandi, S., Schaerf, A.: The balanced academic curriculum problem revisited. Journal of Heuristics **18**, 119–148 (2012)
3. Coleman, B.J., Vaghefi, M.R.: Heijunka (?): A key to the toyota production system. Production and Inventory Management Journal **35**(4), 31 (1994)
4. Di Gaspero, L., Schaerf, A.: Hybrid Local Search Techniques for the Generalized Balanced Academic Curriculum Problem. In: M.J. Blesa, C. Blum, C. Cotta, A.J. Fernández, J.E. Gallardo, A. Roli, M. Sampels (eds.) Hybrid Metaheuristics, 5th International Workshop, HM 2008, Málaga, Spain, October 8-9, 2008. Proceedings, *Lecture Notes in Computer Science*, vol. 5296, pp. 146–157. Springer (2008)
5. Kirkpatrick, S., Gelatt, C.D., Vecchi, M.P.: Optimization by Simulated Annealing. Science **220**(4598), 671–680 (1983). DOI 10.1126/science.220.4598.671
6. Lindauer, M., Eggensperger, K., Feurer, M., Falkner, S., Biedenkapp, A., Hutter, F.: Smac v3: Algorithm configuration in python. `https://github.com/automl/SMAC3` (2017)
7. Mullinax, C., Lawley, M.: Assigning patients to nurses in neonatal intensive care. Journal of the Operational Research Society **53**(1), 25–35 (2002)
8. Punnakitikashem, P., Rosenberber, J.M., Buckley-Behan, D.F.: A stochastic programming approach for integrated nurse staffing and assignment. IIE Transactions **45**(10), 1059–1076 (2013)
9. Schaus, P., Hentenryck, P.V., Régin, J.C.: Scalable Load Balancing in Nurse to Patient Assignment Problems. In: W.J.v. Hoeve, J.N. Hooker (eds.) Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems, 6th International Conference, CPAIOR 2009, Pittsburgh, PA, USA, May 27-31, 2009, Proceedings, *Lecture Notes in Computer Science*, vol. 5547, pp. 248–262. Springer (2009)
10. Vass, J.: Exact and metaheuristic approaches for the production leveling problem. Master's thesis, TU Wien, Vienna, Austria (2019). `https://dbai.tuwien.ac.at/staff/jvass/publications/plp_thesis_final_corrected_20191202.pdf`
11. Vass, J., Lackner, M.L., Musliu, N.: Exact and Metaheuristic Approaches for the Production Leveling Problem. submitted to Computers & Operations Research (2019)