Journal of Scheduling (2022) 25:339–370
https://doi.org/10.1007/s10951-022-00721-1

Exact and meta-heuristic approaches for the production leveling problem

Johannes Vass[1] · Marie-Louise Lackner[1] · Christoph Mrkvicka[2] · Nysret Musliu[1] · Felix Winter[1]

Accepted: 22 December 2021 / Published online: 5 April 2022

Abstract
In this paper, we introduce a new problem in the field of production planning, called the production leveling problem. The task is to assign orders to production periods such that the load in each period and for each product type is balanced, capacity limits are not exceeded, and the orders' priorities are taken into account. Production leveling is an important intermediate step between long-term planning and the final scheduling of orders within a production period, as it is responsible for selecting good subsets of orders to be scheduled within each period. We provide a formal model of the problem and study its computational complexity. As an exact method for solving moderately sized instances, we introduce a mixed integer programming (MIP) formulation. For solving large problem instances, metaheuristic local search is investigated. A greedy heuristic and two neighborhood structures for local search are proposed in order to apply them using simulated annealing. Furthermore, three possible extensions that arise from the application in practice are described and implemented, both within the MIP model and within simulated annealing. We make publicly available a set of realistic problem instances from the industry as well as from random instance generators. The experimental evaluation on our test sets shows that the proposed MIP model is well suited for solving instances with up to 250 orders. Simulated annealing produces solutions with less than 3% average optimality gap on small instances, and scales well up to thousands of orders and dozens of periods and product types. The metaheuristic method presented herein is already being successfully used in the industry.

Keywords Production leveling · Multi-objective optimization · Mixed integer programming · Metaheuristics · Simulated annealing · Complexity analysis

1 Introduction

Production systems have been subject to continuous and radical change over the course of the past few decades. The need for productivity improvements causes companies to invest heavily in automation at all levels. Production planning plays a major role in these developments, as the replacement of manual planning with software-assisted or even autonomous systems can lead to considerable efficiency increases.

In this paper, we introduce a real-life combinatorial optimization problem which treats the leveling of production, and we therefore call it the production leveling problem (PLP). It belongs to medium-term planning, which means it is intended to be embedded between long-term planning and the scheduling of the concrete production sequence. The problem is concerned with assigning orders of certain product types and demand sizes to production periods such that the production volume of each product type is leveled across all periods. Furthermore, the overall amount produced in each period is subject to leveling as well. A solution is feasible if the production volumes to be leveled do not exceed given maximum values. The optimization part consists in minimizing the deviation of the production from the optimal balance,

✉ Marie-Louise Lackner
mlackne1@dbai.tuwien.ac.at

Johannes Vass
jvass@dbai.tuwien.ac.at

Christoph Mrkvicka
christoph.mrkvicka@mcp-alfa.com

Nysret Musliu
musliu@dbai.tuwien.ac.at

Felix Winter
winter@dbai.tuwien.ac.at

[1] Christian Doppler Laboratory for Artificial Intelligence and Optimization for Planning and Scheduling, DBAI , TU Wien, Karlsplatz 13, 1040 Vienna, Austria

[2] MCP GMBH, Vienna, Austria

while at the same time taking into account the orders' priorities, which usually model order due dates or customer priorities.

The idea behind this goal is that considering orders only in the order of decreasing priority, as it is often done, frequently leads to spikes and idle times for certain resources involved in the production process. Leveling these highs and lows results in a smoother production process because a similar product mix is produced in every period. It is important to note that the solution to the PLP is not a schedule since the orders are only assigned to production periods; the concrete execution sequence and assignment to machines and workers is not part of this problem. The intention is rather so serve as a step between long-term planning and short-term production scheduling.

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.

The PLP is related to a multitude of planning problems that have been studied in the literature, such as the balanced academic curriculum problem (Castro and Manzano 2001), nurse scheduling problems (Mullinax and Lawley 2002), and simple assembly line balancing (Boysen et al. 2007). However, to the best of our knowledge, the PLP as we define it in this paper is a new combinatorial optimization problem that has not been investigated so far.

The main goal of this work consists in modeling the PLP formally and developing solution strategies for it. We propose a mixed integer programming (MIP) model in order to obtain optimal solutions and lower bounds. We are interested in finding the border between instances which can be solved exactly and those where the exponential nature of the problem makes the usage of MIP impractical. As a primary method to solve larger instances, we propose simulated annealing. For this purpose, two move types for obtaining neighboring solutions are developed. Using simulated annealing and lower bounds obtained through MIP, we investigate the obtained optimality gap for realistically sized instances. Finally, we present several possible extensions to the basic PLP that allow us to address further requirements that arise in practical use. These include allowing orders to be split over several production periods, considering additional resource constraints, and specifying due dates.

To sum up, the main contributions of this paper are as follows:

- a mathematical model for the PLP,
- proofs of $\mathcal{NP}$-hardness for several restricted variants of the PLP,
- a polynomial-time algorithm for the PLP assuming unique priorities that have to be respected,
- a MIP model,
- neighborhood structures for local search,
- three possible extensions to the basic PLP,
- realistic and randomly generated problem instances,
- an extensive evaluation of MIP and metaheuristic methods.

The current paper is an extension of our PATAT 2021 conference paper (Vass et al. 2020).

The rest of this paper is structured as follows: Section 2 first presents the problem informally, and then a precise mathematical formulation is given. We describe the concrete industry application for which our algorithms are currently being used and discuss related problems in the literature. In Sect. 3, the theoretical complexity of the PLP is analyzed: $\mathcal{NP}$-hardness results are provided for restricted variants of the PLP, and a special case of the PLP is identified for which we describe a dynamic programming algorithm that runs in polynomial time. Section 4 introduces a MIP model. In Sect. 5, we turn to local search methods and present the variant of simulated annealing that we apply. Section 6 presents several extensions to the basic PLP. The experimental evaluation, for both the basic and the extended PLP, is described in Sect. 7, providing answers to the research questions formulated above. Finally, Sect. 8 summarizes the results and presents ideas for future work.

2 Problem statement and related work

We give an informal description of the PLP and provide a simple example which allows the reader to approach the problem intuitively. Then we specify the parameters, constraints, and objective functions formally. Next we give a description of the concrete setting in the industry, where our algorithms are currently being used. Finally, related problems presented in the literature are discussed.

2.1 Problem description

The input to the PLP is a list of orders, each of them having a demand value, priority, and product type. Furthermore, we are given a set of periods and the maximum production capacity per period, both for all product types together and for each one separately. We search for solutions by finding an assignment of orders to periods such that the production volume is balanced between the periods while trying to stick to the sequence implied by the order's priorities as well as possible. We can see the objective function of the PLP as the task of finding a good trade-off between the following goals:

1. Minimize the sum of deviations of the planned production volume to the average demand (i.e., the target value) for

each period, ignoring the product types. This ensures that the overall production per period is being leveled.
2. Minimize the sum of deviations of the production volume of each product type to its respective mean (target) value, making sure that the production of each product type is being leveled.
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 ensures that more urgent orders are scheduled in earlier periods.

Let us now explore the three optimization goals by means of a small example:

1. On the left side of Fig. 1 (which is labeled as *Orders*), we see a small example instance with seven orders, which are shown as boxes, where the box height corresponds to the demand value. There are two different product types (red and blue), and the priorities range from $p = 1$ to $p = 3$. The orders should be assigned to the three periods such that the distances between the stacks of orders and the dashed target line are minimized and no stack crosses the solid line which represents the capacity limit. It is easy to see that this solution is optimal w.r.t. the first leveling objective.
2. The solution presented in Fig. 1 is not optimal with respect to the second leveling objective which aims to level the production among the production periods within each product type. An optimal solution with respect to the second objective, thus minimizing the deviation from the dashed target lines for each product type, is shown in Fig. 2, in which the orders of the two product types are spread out more evenly across the three periods. In this example, however, it is not possible to reach the dashed target lines due to the respective sizes of the orders.
3. In both solutions in Figs. 1 and 2, the priorities have been ignored so far, and there are several priority inversions. For instance, in the solution in Fig. 1, the order O7 is scheduled to period 1 and has priority 1, whereas the order O4 has priority 3 and is scheduled to period 2. In Fig. 3, we see an example solution with no priority inversions. This solution is thus optimal with respect to the third objective.

We have seen in the examples that an optimal solution w.r.t. one objective is not necessarily optimal w.r.t. another. As we want to combine the three objectives into one by a weighted sum, the location of the optima will clearly depend on the weights. We developed a sensible default weighting in cooperation with our industrial partner based on their real-life data, and use these weights in all experiments throughout the paper. How the objective functions can be stated formally

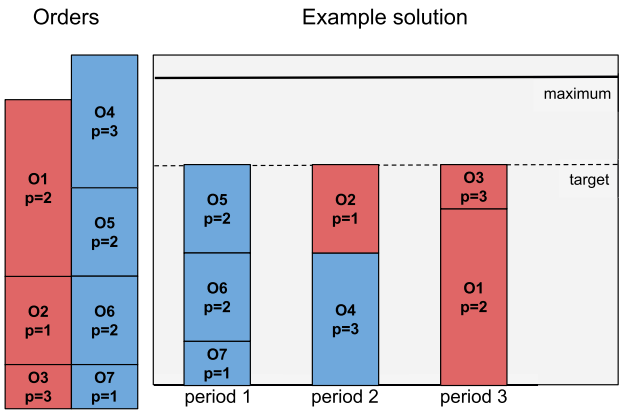

Fig. 1 Example of the leveling objective for the total production amount. The dashed line represents the target value and the solid line the capacity limit. This is an optimal solution w.r.t. the first leveling objective (Color figure online)

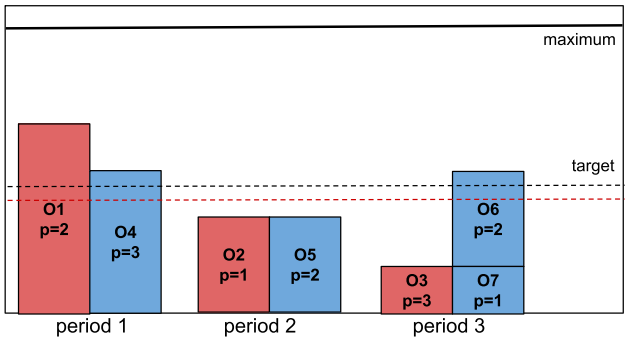

Fig. 2 Example of the leveling objective for each product type (blue, red) (Color figure online)

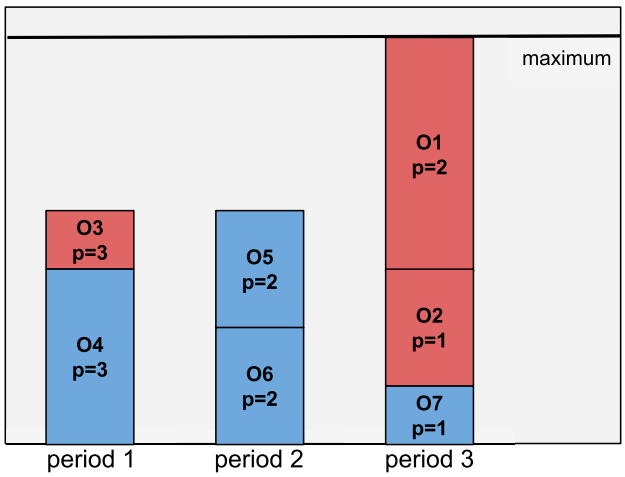

Fig. 3 A valid solution with no priority inversions (Color figure online)

and how the weighting is defined are described in more detail in the following section.

2.2 Mathematical formulation

We provide a formal description of the problem, consisting of parameters, variables, constraints, and the objective function.

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
$a_i \in \mathbb{R}^+$	for each objective function component $i \in \{1, 2, 3\}$ the associated weight
$c \in \mathbb{R}^+$	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
$t_j \in M$	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

– For each order the production period for which it is planned:

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

– The production volume for each period (helper variable):

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

– The production volume for each product type and period (helper variable):

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

Hard constraints

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

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

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

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

Objective function

The following three objective functions represent the three targets to minimize:

$$f_1 = \sum_{i \in N} |d^* - w_i| \tag{obj1}$$

$$f_2 = \sum_{t \in M} \left(\frac{1}{d_t^*} \cdot \sum_{i \in N} |d_t^* - w_{i,t}| \right) \tag{obj2}$$

$$f_3 = \left| \left\{ (i, j) \in K^2 : y_i > y_j \text{ and } p_i > p_j \right\} \right| \tag{obj3}$$

Function f_1 represents the sum over all periods of deviations from the overall target production volume (i.e., all product types at once). Function f_2 states the sum over all product types of sums over all periods of the deviations from the target production volume for that product type, normalized by the respective target value. The normalization is done so that every product has the same influence onto the objective function regardless of whether its average demand is high or low. Function f_3 counts the number of priority inversions in the assignment, or in other words the number of order pairs (i, j) for which i is planned after j even though i has a higher priority than j.

Finding a trade-off among several objective functions is the topic of multi-objective optimization (Deb 2014; Miettinen 2012). Often, lexicographic optimization is chosen in practice. However, we wanted an approach that allows high flexibility and lets users decide on the importance of each objective. Together with our industrial partner we therefore decided to combine the three objective functions into

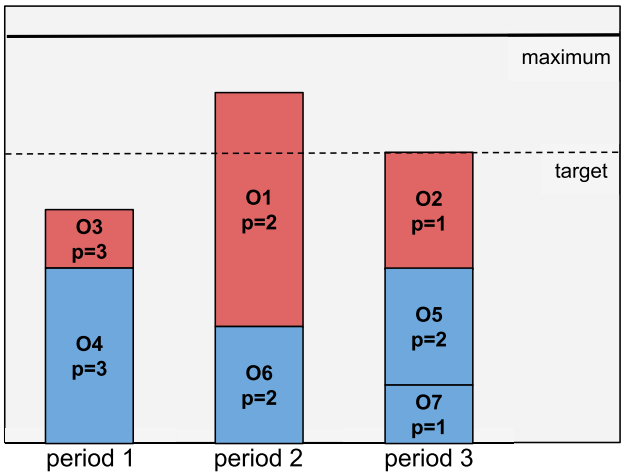

Fig. 4 An optimal solution for the example instance with weights $a_1 = 1$, $a_2 = 1$, and $a_3 = 1/3$ (Color figure online)

a weighted sum. To do so, the cost components need to be normalized:

$$g_1 = \frac{1}{n \cdot d^*} \cdot f_1 \qquad \text{(obj1}')$$

$$g_2 = \frac{1}{n \cdot m} \cdot f_2 \qquad \text{(obj2}')$$

$$g_3 = \frac{2}{k \cdot (k-1)} \cdot f_3 \qquad \text{(obj3}')$$

The normalization ensures that g_1 and g_2 stay between 0 and 1 with high probability. Higher values for g_1 and g_2 are only possible for degenerated instances where, even in good solutions, the target is exceeded by factors ≥ 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 is then a weighted sum of the three normalized objective functions, where the weight a_i of an objective can be seen as approximately its relative importance.

$$\textbf{minimize } g = a_1 \cdot g_1 + a_2 \cdot g_2 + a_3 \cdot g_3 \qquad \text{(obj)}$$

Let us return to the example instance presented in Fig. 1. An optimal solution with respect to the combined objective function with weights $a_1 = 1$, $a_2 = 1$, and $a_3 = 1/3$ is shown in Fig. 4.

For comparison, the (rounded) objective values are given for the four different solutions shown in Figs. 1, 2, 3, and 4 in the following table.

Solution	g_1	g_2	g_3	g
Figure 1	**0**	0.98	0.38	1.1
Figure 2	0.27	**0.62**	0.19	0.95
Figure 3	0.27	0.74	**0**	1.0
Figure 4	0.13	0.74	**0**	**0.87**

2.3 Concrete industry application

The PLP can appear in many areas of industrial manufacturing, and its application is not restricted to any specific industrial branch. However, to further motivate the investigation of the problem, we want to briefly describe a concrete application of the problem in the area of electronic component manufacturing, which has been deployed by our industry partners.

The PLP is especially beneficial in this area because electronic components manufacturing deals with a large variety of products which can be grouped into product families. Additionally, setup costs are usually low whenever the production process switches from one product family to another, and it thus makes sense to produce small lot sizes for each product family. In such a setting, the main aims of an efficient production are to minimize storage costs, maximize capacity utilization, and deal with fluctuating demands. Therefore, a well-balanced product mix in each of the production periods (which is established by optimized solutions to the PLP) is beneficial, as it leads to increased capacity utilization as well as decreased storage and transport costs.

A just-in-time production in the sense of Coleman and Vaghefi (1994) is encouraged, and actually the optimal case would be to balance all different electronic component families perfectly over the planning periods. However, often there are maximum machine capacities that have to be considered for each product type, which do not allow a perfect distribution. The PLP allows the specification of such capacity constraints and further supports prioritization of orders which is used to model due date requirements and customer priorities in this application. Note that besides the capacity constraints, there are no further resource constraints that need to be considered in the particular real-life application from electronic component manufacturing on which the PLP is based. Therefore, the methods we propose to solve the PLP could be directly applied by our industry partner.

In this paper, we provide and investigate practical instances for this case study from the electronics industry, which are available for download[1].

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

2.4 Related work

The term *production leveling* is commonly associated with the Toyota Production System (TPS), where it is also called heijunka. It is a concept which aims to increase efficiency and flexibility of mass production by leveling the production to keep the stock size low and reduce waste. Ideally, the result of applying heijunka is zero fluctuation at the final assembly line. Heijunka can mean both the leveling of volume at the final assembly line and the leveling of the production of intermediary materials (Ohno and Rosen 1998).

The PLP is clearly inspired by heijunka in the sense that the usage of resources should be leveled to increase production efficiency, but its concepts differ quite substantially from the classical implementation of heijunka (in the TPS) in the following points:

- The PLP does not operate on the level of schedules, but disregards the order in which the items are produced within a period. In other words, it is concerned with planning and not scheduling, which is subsequently performed for each production period.
- Intermediate materials are not part of the PLP. While heijunka also aims to level *their* production to keep stock sizes of intermediary products small, the PLP is currently only concerned with one level.

The main differences and similarities between the PLP and other related problems are summarized in Table 1. In this table, the columns correspond to features of the PLP such as the presence of an overall leveling objective and the presence of a priority objective. The rows correspond to the related problems treated in this section. A cell containing a + sign indicates that the problem in this row has the feature in this column, i.e., that this is a property in common with the PLP. A − sign indicates that the feature is absent in the problem at hand. The last column lists other differences between the PLP and the problems studied in the literature. In the following, we discuss the related literature in more detail.

There exists a whole research area concerning scheduling problems inspired by ideas from the TPS, and especially heijunka. Several problems exist under the umbrella term *level scheduling*, including the output variation problem and the product rate variation problem (Kubiak 1993; Boysen et al. 2009). They have the common aim of finding the best schedule for production at the final assembly line so that the demand for intermediary materials and their production is leveled, which keeps the necessary stock sizes low. However, these problems are quite different from the PLP for the same reasons presented above with respect to heijunka.

In this context, let us also mention the work on the production smoothing problem (PSP) as well as the batch PSP (BPSP) for mixed-model just-in-time production (Miltenburg 1989; Kubiak and Yavuz 2008; Yavuz and Tufekci 2006a, b). For the PSP, one unit of any product can be processed at a time, and a schedule of different products needs to be found to reach overall demands per product. For the BPSP, several copies of products may be grouped into batches to be processed simultaneously; it is part of the problem to find optimal batch sizes and numbers. Ideally, the production of different products is distributed evenly over time so that the amount of a certain product produced up to any time is proportional to time elapsed. For the (B)PSP, the objective is thus to minimize the total squared deviation between the cumulative production volumes of the actual and the ideal schedule, for every product and at every time in the production sequence. Even though this objective is similar to the leveling objectives of the PLP, it differs from those of the PLP in the crucial aspect that cumulative production volumes up to a given point in time are considered, and the PLP aims to level production volumes within every time period. Another difference is that for both the PSP and the BPSP, overall demands per product are given, whereas the PLP consists of a set of distinct orders with given demands and priorities for every product (type).

Another real-life problem where leveling in the sense of heijunka is desired is a variant of the aircraft landing problem (ALP) studied by Boysen and Fliedner (2011). The aim is to find a schedule of aircraft such that the workload of ground staff, which essentially depends on the number of passengers landing (per airline), is balanced over time. Similarly to the (B)PSP, the objective considers cumulative passenger numbers over time, and the goal is to minimize the maximum deviation from a target value. Another difference from the PLP is that a single airplane may be scheduled for every time slot, whereas the PLP allows us to assign an arbitrary number of orders to a time period, as long as all capacity constraints are fulfilled. Moreover, no constraints or preferences on the order of the landed aircraft are considered.

Under the term *balancing problems*, several other problems are known in the literature, which are more closely related to the PLP:

- The balanced academic curriculum problem (BACP): This problem deals with assigning courses to terms such that the students' load per term is balanced and prerequisites are fulfilled (Castro and Manzano 2001; Chiarandini et al. 2012). The balancing of the sum of course sizes assigned to a term is similar to the balancing of the production load per period which we are confronted with in the PLP. However, there is no equivalent to the second objective of the PLP, which aims to balance the production within all product types. In a more general version of the problem, called the generalized BACP, so-called *curricula* are introduced (Di Gaspero and Schaerf 2008): every course is part of one or possibly several curricula,

Table 1 Similarities and differences between PLP and other related problems studied in the literature. A + sign indicates the presence of a problem feature, and a − sign the absence of a problem feature

	Overall leveling objective	Leveling each product type	Priority objective	Assignment to periods	Other differences from the PLP
Level scheduling problems such as *output variation* and *product rate variation* (Kubiak 1993; Boysen et al. 2009)	Various forms of leveling objectives		−	Schedule of orders	Consideration of intermediary materials
(Batch) production smoothing (Miltenburg 1989; Kubiak and Yavuz 2008; Yavuz and Tufekci 2006a, b)	Cumulative production volumes instead of production volumes within every time period		−	Schedule of orders	Overall demands per product type instead of distinct orders
Aircraft landing problem (Boysen and Fliedner 2011)	Cumulative production volumes	−	−	Schedule of orders	
Balanced academic curriculum problem (Castro and Manzano 2001; Chiarandini et al. 2012; Di Gaspero and Schaerf 2008)	+	−	Precedence relations	+	
Nurse scheduling (Warner 1976; Mullinax and Lawley 2002; Schaus et al. 2009; Punnakitikashem et al. 2013)	+	−	−	+	Often solved in two steps—nurses assigned to zones first; different constraints
Multiprocessor scheduling (Schreiber 2014; Alon et al. 1998; Schwerdfeger and Walter 2016)	Similar for Schwerdfeger and Walter (2016)	−	−	+ (machines correspond to periods)	
Simple assembly line balancing (Boysen et al. 2007)	+	−	Precedence relations	+	
Production leveling problem (Vass et al. 2020; Vass 2019)	+	+	+	+	

and every one of the curricula should be balanced. At a first glance, it might seem that curricula can be translated to product types, but this is not the case. For the generalized BACP, there is no need to balance the total load per term for all curricula which would correspond to the total production volume per period.
Another difference from our problem is the additional constraints of the BACP, which enforce prerequisites between (some of the) courses—a concept appearing frequently in balancing problems. They differ from the PLP's priorities in the following aspect: Prerequisites are hard constraints, while priorities are part of the objective function, which makes a difference especially for exact solvers.

– Nurse scheduling problems are an active field of research since their introduction by Warner (1976). While most of the contributions do not consider workload balancing, a few of them, starting with Mullinax and Lawley (2002), do consider also a fair distribution of the nurses' workload. They propose an integer programming model for the nurse-to-patient assignment problem in neonatal intensive care, which is concerned with finding the optimal assignment of patients to a set of working nurses, so that the workload of the team is balanced and a number of restrictions are fulfilled. The main difficulty is the variability in the infants' conditions, which greatly influences the amount of work needed. The problem is often solved in two steps by first assigning nurses to zones of the nursery and then assigning infants to nurses. More recently, Schaus et al. (2009) investigated a constraint programming (CP) approach using the `spread` constraint for balancing. Furthermore, stochastic programming-based approaches with Bender's decomposition have been proposed (Punnakitikashem et al. 2013).
The balancing objective of the nurse-to-patient assignment problem is again very similar to the objective function which we introduced for the PLP. However, we cannot compare the results directly because the priorities of the PLP have no equivalent in this problem, and vice versa, some side constraints and the zone assignment cannot be expressed.
– Multiprocessor scheduling: This $\mathcal{NP}$-hard scheduling problem consists in assigning n jobs with integer processing times $p_1, p_2, \ldots, p_n$ to one of $m < n$ identical machines that run in parallel. The goal is to minimize the overall maximum completion time of any machine. An analogy to the PLP with a single product type can be seen by identifying jobs with orders, job processing times with order demands, and machines with production periods. The multiprocessor scheduling problem can equivalently be formulated as an optimization variant of the multi-way number partitioning problem. For several different variants of this problem and algorithmic approaches, we refer to the PhD thesis of Schreiber (2014). Polynomial-time approximation schemes were proven to exist for a large class of multiprocessor scheduling problems where the objective depends on the completion times of the machines (Alon et al. 1998). Recently, Schwerdfeger and Walter (2016) studied a variant of multiprocessor scheduling that accounts for balancing issues between the machines and for which the objective incorporates the completion times of all machines (by minimizing the NSSWD [normalized sum of squares for workload deviations] criterion). This objective resembles the leveling objectives of the PLP, with the only difference being that we consider linear instead of quadratic deviations from the target demand with respect to completion time. However, one major difference from the PLP remains for all these problem variants: for multiprocessor scheduling problems, the machines are identical; for the PLP, periods are sorted in time, and the choice of the production period is crucial when aiming to respect the order's priorities.
– Simple assembly line balancing (SALB): An assembly line consists of identical work stations aligned along a conveyor belt. Workpieces move along the conveyor belt, and at each station a set of (assembly) tasks is carried out, where each of them has a task time. The cycle time denotes the time after which workpieces are moved on to the next station. The goal is either to minimize the number of work stations needed given a fixed cycle time or to minimize the cycle time given a fixed number of work stations.
The SALB problem is the simplest and most intensively studied variant of assembly line balancing. A comprehensive overview over the different variants is provided by Boysen et al. (2007). When comparing the SALB problem to the PLP, tasks map to orders, task times to order sizes, and the fixed cycle time to the maximum capacity per production period. Hence, minimizing the cycle time is equivalent to minimizing the maximum load of a production period of the PLP, which would also be an admissible balancing objective.
However, the difference between precedence relations on the one hand and priority inversion minimization on the other hand once again precludes a direct comparison between the problems.

For a more extensive list of balancing problems, please refer to the dissertation of Pierre Schaus, which investigates CP modeling approaches for a very diverse set of balancing and bin-packing problems (Schaus 2009).

3 Complexity analysis

3.1 Hardness results

To demonstrate the computational hardness of the PLP optimization problem, we provide two $\mathcal{NP}$-hardness results, both of which hold even for very restricted versions of the PLP.[2] First, we introduce the following feasibility variant of the problem where the objective function is dropped completely. Hence, the task is solely to decide whether a feasible assignment of orders to periods is possible:

PRODUCTION LEVELING FEASIBILITY (DECISION PROBLEM)	
Instance:	A set of orders K, of product types M and of periods N. For each order $j \in K$ its demand d_j with $d_j > 0$ and product type t_j. The maximum production capacity per period c and for each product type $t \in M$ its associated maximum production capacity per period c_t.
Question:	Does there exist an assignment $y : K \to N$ of orders to periods such that the capacity limits c and $(c_t)_{t \in M}$ are not exceeded for any period?

Note that this feasibility variant of the PLP neither contains priorities p_j nor target production capacities d^* and d_t^*, since these are only involved in the objectives but not in the hard constraints of the PLP.

Theorem 1 *The production leveling feasibility problem is $\mathcal{NP}$-complete even on instances with $|M| = 1$, i.e., with a single product type.*

Proof To prove $\mathcal{NP}$-hardness, we give a polynomial-time reduction from the $\mathcal{NP}$-complete bin-packing decision problem (Vazirani 2003), which is defined as follows:

BIN PACKING (DECISION PROBLEM)	
Instance:	A list of k items of respective sizes $a_1, a_2, \ldots, a_k$, a number of bins n, and a bin capacity $b \in \mathbb{N}$.
Question:	Can the items be packed into the bins; i.e., is there a partition of the set $\{1, 2, \ldots, k\}$ into n disjoint subsets $S_1, S_2, \ldots, S_n$ such that $\sum_{i \in S_j} a_i \leq b$ for all $j \in \{1, \ldots, n\}$?

Given a bin-packing instance, the construction of an instance of the production leveling feasibility problem is straightforward:

$$
\begin{aligned}
M &= \{1\} & d_j &= a_j \quad \forall j \in K \\
N &= \{1, 2, \ldots, n\} & t_j &= 1 \quad \forall j \in K \\
K &= \{1, 2, \ldots, k\} & c &= c_1 = b
\end{aligned}
$$

That is, bins are converted to periods, each item with size a_i to an order with demand d_i, and the bin capacity V becomes the maximum capacity per period c. Since this reduction can be done in linear time, we have proven that the production leveling feasibility problem is $\mathcal{NP}$-hard, even when considering only a single product type ($|M| = 1$).

To verify whether an assignment $y : K \to N$ of orders to periods is a valid solution, a total of $|N| \cdot (1 + |M|)$ inequalities need to be checked for the capacity constraints. This proves $\mathcal{NP}$-membership. □

As an immediate consequence, we obtain that the decision variant of the production leveling optimization problem which asks whether a feasible assignment with objective value $\leq k \in \mathbb{N}$ exists is $\mathcal{NP}$-hard as well.

Our second hardness result shows that the hardness of the PLP is not only caused by the capacity constraints but also by the leveling objective itself:

Theorem 2 *The production leveling optimization problem minimizing the objective function f_1 as defined in Eq. (obj1) is $\mathcal{NP}$-hard even in instances with unbounded capacities and a single product type.*

Proof To prove $\mathcal{NP}$-hardness, we show that it is even $\mathcal{NP}$-hard to verify whether there exists a solution of the PLP with $f_1 = 0$. We give a polynomial-time reduction from the $\mathcal{NP}$-hard 3-partition problem (Garey and Johnson 1979):

3- PARTITION (DECISION PROBLEM)	
Instance:	A multiset of $3q$ positive integers $a_1, \ldots, a_{3q}$ and a positive integer B such that $\sum_{p=1}^{3q} a_p = q \cdot B$ and $B/4 < a_p < B/2$ holds for all $1 \leq p \leq 3q$.
Question:	Is there a partition of $\{1, \ldots, 3q\}$ into q subsets $\{A_1, \ldots, A_q\}$ such that $\sum_{p \in A_i} a_p = B$ for all $1 \leq i \leq q$?

Note that the condition on the size of the integers a_p for $1 \leq p \leq 3q$ guarantees that any such partition has exactly three elements per subset A_i, $1 \leq i \leq q$.

Given a 3-partition instance with $a_1, \ldots, a_{3q}$ and B, the construction of an instance of the production leveling optimization problem with q periods and a single product type is again straightforward:

$$
\begin{aligned}
M &= \{1\} \quad \text{(product types)} & d_j &= a_j \quad \forall j \in K \\
N &= \{1, \ldots, q\} \quad \text{(periods)} & p_j &= t_j = 1 \quad \forall j \in K
\end{aligned}
$$

[2] All $\mathcal{NP}$-hardness proofs in this section are in the strong sense; i.e., we do not require assumptions on how the integers in a PLP instance are encoded.

$$K = \{1, 2, \ldots, 3q\} \quad \text{(orders)} \qquad c = c_1 = \infty$$

Now, note that a solution to the PLP is an assignment $y : K \to N$ of orders to periods which corresponds to a partition of $\{1, \ldots, 3q\}$ into q subsets $\{A_1, \ldots, A_q\}$. It holds that

$$f_1 = \sum_{i \in N} |d^* - w_i| = \sum_{i=1}^{q} \left| \frac{\sum_{j \in K} d_j}{q} - w_i \right|$$
$$= \sum_{i=1}^{q} \left| B - \sum_{j \in K : y_j = i} d_j \right| = 0$$

if and only if

$$\sum_{p \in A_i} a_p = B \quad \text{for all } 1 \le p \le q,$$

where $A_i = \{a_j : y_j = i\}$ for $i \in \{1, \ldots, q\}$. Thus, $f_1 = 0$ if and only if the 3-partition instance is a yes-instance. □

Note that Theorem 2 also holds for the objective function f_2 as defined in Eq. (obj2′), since f_1 and f_2 are identical on instances with a single product type.

Moreover, the PLP admits a trivial solution if we assume unbounded capacities and the sole objective is to minimize f_3 as defined in Eq. (obj3′), i.e., to minimize priority inversions. Indeed, all orders can be assigned to the same period in this case, and thus no priority inversions occur. If we however assume bounded capacities, hardness already follows from Theorem 1. Consequently, the hardness of the PLP with respect to objective f_3 is already handled by Theorem 1.

3.2 A polynomial-time algorithm for the fixed-order production leveling problem

Having established the computational hardness of restricted cases of the PLP, we are now interested in identifying tractable fragments of our problem. Such tractable fragments complement our understanding of where the computational difficulty of PLP originates. In this section, a tractable case of the production leveling problem is analyzed, which we formalize as the fixed-order PLP. For this problem, we present a dynamic programming algorithm running in polynomial time.

The fixed-order PLP is a variant of the production leveling optimization problem, where the priority values of all orders are unique and the correct ordering with respect to the priorities is enforced. We will show that these two restrictions render the problem solvable in polynomial time.

This variant of the PLP is not only of theoretical interest, but also reflects a plausible practical scenario. Indeed, priority values can be used to reflect due dates of orders. In the case of a high number of different customers that need to be served, it is plausible that all due dates are different; they can thus be encoded using distinct priorities. Moreover, let us assume that it is crucial to keep stock sizes low (just-in-time production). This means that orders should never be completed before other orders with earlier due dates; in other words, priority inversions are not permitted. In this scenario, the PLP reduces to the fixed-order production leveling problem which we formally define as follows:

FIXED- ORDER PRODUCTION LEVELING (OPTIMIZATION PROBLEM)

Instance:	A set of orders K, of product types M and of periods N. For each order $j \in K$ its demand d_j with $d_j > 0$, priority p_j and product type t_j. The maximum production capacity per period c and for each product type $t \in M$ its associated maximum production capacity per period c_t. The target production capacity d^* and the target production capacity d_t^* for each product type t. Priorities are unique: $p_i \neq p_j$ for $i \neq j \in K$
Objective:	Find an assignment $y : K \to N$ of orders to periods that minimizes $g = a_1 \cdot g_1 + a_2 \cdot g_2$, with objectives g_1 and g_2 as defined in Eqs. (obj1′) and (obj2′).
Constraints:	Respect the priorities: $y(i) \le y(j)$ for orders $i, j \in K$ with $p_i > p_j$. Respect the capacity limits c and $(c_t)_{t \in M}$ for all periods.

The only difference from the original version of the PLP is that the priorities are treated as hard constraints instead of soft constraints. This new constraint, together with the assumption of unique priority values, enables a drastic reduction of the search space: Given the sequence of orders sorted by decreasing priority values, solving the fixed-order PLP is equivalent to finding the best partitioning of this sequence into periods. That is, instead of assigning arbitrary subsets of orders to periods, the sorted sequence of orders needs to be divided into contiguous subsequences which are assigned to periods in the correct order.

Viewing the fixed-order PLP as a partition problem reminds of the list partition problem, as described for example by Skiena (1998). In this problem, a sequence of non-negative numbers $s_1, s_2, \ldots, s_k$ and an integer n is given and a partition of the sequence into n ranges, i.e., consecutive elements of the sequence, so as to minimize the maximum sum over all the ranges is sought for. In the following, we present an adaption of the dynamic programming approach for the list partition problem to the fixed-order PLP; in particular, the objective to be minimized differs for the two problems.

Theorem 3 *The fixed-order production leveling problem can be solved in polynomial time using a dynamic programming approach: If n denotes the number of periods, k the number of orders, and m the number of product types, it can be solved in* $\mathcal{O}((n + m) \cdot k^2)$ *time.*

A detailed example illustrating the fixed-order PLP can be found in a technical report (Lackner et al. 2019). It can be read alongside the following formal description of the dynamic programming algorithm.

Proof Without loss of generality, let us assume that the unique priority values p_j for $j \in K$ are elements of the set $\{1, \ldots, k\}$. In a preprocessing step, sort the orders in decreasing order of their priorities. That is, after sorting, d_1 denotes the capacity demand of the order with priority k, d_2 the capacity demand of the order with priority $k-1$, and so on until d_k which denotes the capacity demand of the order with priority 1. This sorting requires $\mathcal{O}(k \log k)$ time.

Furthermore, we assume that $k \geq n$. This is sensible, as for $k < n$ some periods would have to remain empty, and we could simply ignore $(n-k)$ of these periods without changing the outcome of the problem. With this assumption, we can in general exclude all assignments $y : K \to N$ for which some period remains empty: With respect to the objective function g and the capacity constraints, it is never a disadvantage to plan two orders in different periods compared to planning two orders in one period and leaving one period empty.

Given these assumptions for the fixed-order PLP, an assignment $y : K \to N$ of orders to periods can uniquely be characterized by the choice of the first order $f(l) \in K$ assigned to period $l \leq n$, where it has to hold that $f(1) = 1$ and $f(l) < f(l+1)$ for all $l \leq n-1$. Indeed, given $f(l)$ and $f(l+1)$, the set of orders assigned to period l is $\{f(l), \ldots, f(l+1)-1\}$. This results in $\binom{k-1}{n-1} = \frac{(k-1)!}{(n-1)!(k-n)!}$ possibilities for the function f among which an optimal solution needs to be found. In the following, we will see how a dynamic programming approach can be used to solve the fixed-order production leveling optimization problem in polynomial time and thus to escape the exponential growth of the binomial coefficients.[3]

Let us denote by $O(j, l)$ the optimal value of the objective function under consideration, i.e., the minimum value of g, when assigning the first j orders $\{1, \ldots, j\}$ to $l \leq n$ periods. The optimal objective value of the given instance of the fixed-order PLP is then given by $O(k, n)$. The main idea of the dynamic programming algorithm is that $O(j, l)$ for $l \leq j \leq k$ and $1 \leq l \leq n$ can be calculated recursively:

- First, pick $f(l) = i$, the first order to be assigned to period l. Since no period can remain empty, $l \leq i \leq j$.
- Second, assign orders $1, \ldots, i-1$ to periods $1, \ldots, l-1$ in such a way that the optimal value $O(i-1, l-1)$ is reached for these periods.
- Third, compute the objective value of this combined solution for j orders and l periods. This can be done by adding the objective value for the first $(l-1)$ periods to the objective value for period l, because the assignments for each period and for each product type within a period contribute independently to the total cost of a solution.
- Repeat these steps for every possible choice of i and take the overall minimum objective value.

Formally, the recursion for calculating the values $O(j, l)$ is given by

$$O(j,l) = \min_{l \leq i \leq j} \big[O(i-1, l-1) + h_1(i,j) + h_2(i,j) + constr(i,j) \big] \quad \forall l \leq j \leq k. \tag{1}$$

Note that $O(j, l)$ is not defined for $j < l$ because we do not consider solutions that involve empty periods. For all $1 \leq i \leq j+1$, the functions $h_1(i, j)$ and $h_2(i, j)$ denote the respective cost increase of g_1 and g_2 as defined in Eqs. (obj2′) and (obj3′), if we assign the set of orders $\{i, \ldots, j\}$ to a new period:

$$h_1(i,j) = \frac{a_1}{n} \cdot \left| \frac{d^* - d(i,j)}{d^*} \right|, \quad \text{where } d(i,j) = \sum_{s=i}^{j} d_s \tag{2}$$

$$h_2(i,j) = \frac{a_2}{n \cdot m} \cdot \sum_{t \in M} \left| \frac{d_t^* - d_t(i,j)}{d_t^*} \right|, \quad \text{where } d_t(i,j) = \sum_{\substack{s \in \{i,\ldots,j\} \\ \wedge t_s = t}} d_s. \tag{3}$$

The functions d and d_t denote the overall cumulative demand for the new period and the cumulative demand for a specific product type t.

Furthermore, the penalty function $constr(i, j)$ used in Eq. (1) checks whether the capacity constraints c and c_t are satisfied when assigning the set of orders $\{i, \ldots, j\}$ to the same period and is defined as follows:

$$\text{constr}(i,j) = \begin{cases} 0 & \text{if } d(i,j) \leq c \\ & \text{and } d_t(i,j) \leq c_t \quad \forall t \in M \\ \infty & \text{otherwise,} \end{cases}$$

using the cumulative demand functions as defined in Eqs. (2) and (3). If the final value $O(k, n)$ is equal to ∞, this immediately indicates that the instance of the fixed-order PLP is infeasible.

[3] Recall that $\frac{k^n}{n^n} \leq \binom{k}{n} \leq \frac{(k \cdot e)^n}{n^n}$ for all $1 \leq n \leq k$. For more precise asymptotics, we refer to Flajolet and Sedgewick (2009).

The base cases of the recursion consist in assigning j orders to a single period:

$$O(j,1) = h_1(1,j) + h_2(1,j) + constr(1,j) \quad \forall 1 \leq j \leq k.$$

To calculate $O(k,n)$, we store the partial results $O(j,l)$ for $1 \leq j \leq k$ and $1 \leq l \leq n$ in a table of size (k,n). To compute one of these entries $O(j,l)$, we require the values $O(i-1,l-1)$ with $1 \leq i \leq j+1$, i.e., all elements in the column to the left of and not below $O(j,l)$. We thus fill in the table column by column from left to right and top to bottom within a column. Moreover, we require the values $h_1(i,j)$, $h_2(i,j)$ and $constr(i,j)$ for every $1 \leq i \leq j \leq k$. Since these values are also required for further elements of the table, they are computed in a preprocessing step. The cumulative demands $d(i,j)$ and $d_t(i,j)$ are calculated first for all i,j with $1 \leq i \leq j \leq k$ and all product types $t \in M$ and then used to calculate $h_1(i,j)$, $h_2(i,j)$ and $constr(i,j)$. These pre-computations require $\mathcal{O}(m \cdot k^2)$ time where m is the number of product types, assuming that arithmetic operations can be performed in constant time.

Given these pre-computations, the time needed to compute each entry $O(j,l)$ is in $\mathcal{O}(k)$ because the minimum of $j-l+1 \leq k$ values needs to be found, each of which requires only access to three previously computed values. As the table has the size $k \cdot n$, computing all elements of the table can be done in $\mathcal{O}(n \cdot k^2)$ time.

We are not merely interested in computing the value of g for an optimal solution, but also in describing this optimal solution. That is, we need to know which orders are assigned to which period. While we compute the values $O(j,l)$, we thus also store the value of i for which the minimum was achieved in Eq. (1); if this value of i is not unique, we pick the smallest such i. We let $M(j,l)$ denote the index of the first order assigned to period l in a solution to the subproblem finding an optimal assignment of j orders to l periods. More formally:

$$\begin{aligned} M(j,l) = i \implies O(j,l) = \; & O(i-1,l-1) \\ & + h_1(i,j) + h_2(i,j) \\ & + \text{constr}(i,j), \end{aligned}$$

for all $1 \leq l \leq n$ and $l \leq j \leq k$. Computing $M(j,l)$ in addition to $O(j,l)$ adds only a constant amount to the computational complexity, so that the asymptotic behavior does not change.

Once all values for $O(j,l)$ and $M(j,l)$ have been computed, the assignment of orders to periods can be reconstructed as follows, starting with the last period and ending with the first one:

- The orders $o_{M(k,n)}, \ldots, o_k$ are assigned to the last period.
- Given that the first order assigned to period l with $l > 2$ is $o_i = o_{M(j,l)}$ for some j, the orders assigned to period $l-1$ are $o_{i'}, \ldots, o_{i-1}$, with $i' = M(i-1, l-1)$.
- The remaining orders are assigned to the first period.

Reconstructing the solution requires a linear amount of time in the number of periods n and the number of orders k. Therefore, the total asymptotic runtime of the dynamic programming algorithm is $\mathcal{O}((n+m) \cdot k^2)$. □

The dynamic programming algorithm that we developed to solve the fixed-order PLP can also be used to construct a solution for an instance of the more general PLP. Indeed, an instance of the PLP can be converted to an instance of the fixed-order PLP in the following way: If orders i and j with $i \leq j \in K$ have the same priority $p = p_i = p_j \in \mathbb{Z}^+$ and if p^* is the next largest priority value present in the instance, set $p_i = (p + p^*)/2$ and repeat this step until all orders have distinct priority values. Replace the obtained priority values by the set of integers $\{1, \ldots, k\}$ while preserving the order of priorities. If the dynamic programming algorithm described above finds a solution for the converted fixed-order PLP instance, this is also a solution for the PLP instance. Note however that it is possible that the fixed-order PLP instance obtained is unsatisfiable even though the PLP instance is satisfiable. This can be the case for the conversion described here but also for any other conversion that conserves the priority order. Indeed, it might be necessary to introduce priority inversions to satisfy the capacity constraints for certain PLP instances.

4 Integer programming model for the PLP

The PLP is a weakly constrained optimization problem as the only existing constraints are upper bounds on the planned production volume per period and product. There are no constraints involved in the prioritization, and the objective function is a trade-off, which means for example that pruning solutions with a bad priority objective is not immediately possible as long as there is enough room for improvement in the balancing objectives. Hence, the feasible solution space is very large. We propose an integer programming model for the PLP which is capable of providing exact solutions to the optimization problem for moderately sized instances.

The model is based on the mathematical formulation presented in Sect. 2.2. The problem's input parameters are exactly the same, which is why they are not repeated in this section. However, the set of variables differs from the one in the mathematical formulation: A binary view is introduced on the order-period assignment through the set of variables X and the replacement of the helper variables w_i and $w_{i,t}$

by two variables representing the deficiency and the surplus demand.

Variables

$x_{ij} \in \{0, 1\}$	for each $i \in N$, $j \in K$ stating if order j is planned in period i
$y_j \in N$	for each $j \in K$, whose value is the assigned period of order j
$z_{ij} \in \{0, 1\}$	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 deficiency 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 deficiency production volume for period i and product type t

Formulation

Minimize $a_1 g_1 + a_2 g_2 + a_3 g_3$

such that

$$\sum_{i \in N} x_{ij} = 1 j \in K \tag{4}$$

$$\sum_{i \in N} i \cdot x_{ij} = y_j j \in K \tag{5}$$

$$y_i - y_j \leq (n-1) z_{ij} i, j \in K \mid p_i > p_j \tag{6}$$

$$\sum_{j \in K} d_j x_{ij} + s_i^+ - s_i^- = d^* i \in N \tag{7}$$

$$\sum_{j \in K \mid t_j = t} d_j x_{ij} + s_{it}^+ - s_{it}^- = d_t^* i \in N, \ t \in M \tag{8}$$

$$d^* + s_i^+ \leq c i \in N \tag{9}$$

$$d_t^* + s_{it}^+ \leq c_t i \in N, \ t \in M \tag{10}$$

$$y_i \leq y_j i, j \in K \mid p_i \geq p_j, d_i = d_j, t_i = t_j \tag{11}$$

$$\sum_{t \in M} (s_{it}^- - s_{it}^+) = s_i^- - s_i^+ i \in N \tag{12}$$

Constraints (4) to (8) are the model's required helper constraints. Constraints (4) ensure that there is exactly one period to which an order is assigned. Constraints (5) link the x_{ij} to the y_i variables. Constraints (6) link the y_i to the $z_{i,j}$ variables. It ensures that for every pair of orders i, j where i has a higher priority than j, z_{ij} is 1 (representing an inversion) if i is planned later than j. Constraints (7) state for each period that the total demand planned plus the surplus minus the deficiency equals d^*. As both variables have positive domains and they are subject to minimization, at most one of them will be nonzero in any optimal solution. Constraints (8) repeat this relationship over the variables s_{it}^+ and s_{it}^- for each product type t.

Constraints (9) ensure that the capacity bound per period is satisfied. This is elegantly achieved by stating that the sum of target demand d^* and the surplus variable s^+ does not exceed the threshold. Analogously, Constraints (10) enforce the capacity limit per period and product type.

Finally, there are two redundant sets of constraints for strengthening the formulation: Constraints (11) enforce 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 occurs no later than the lower prioritized one, which is sensible because we could otherwise swap the two orders to obtain a better solution. This cuts off parts of the search space where the optimal solution cannot reside. Constraints (12) link the $s_i^{\{+,-\}}$ and $s_{it}^{\{+,-\}}$ variables together, which also leads to improvements in the average runtime.

The following objective function is equivalent to the one presented in Sect. 2.2, but here it is stated on the variable set of the MIP formulation. It is not hard to see that in function g_1, the sum of the deficiency 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 solution, and the other one holds the absolute difference. The same holds true for the analogous variables in g_2.

$$g_1 = \frac{1}{n \cdot d^*} \cdot \sum_{i \in N} (s_i^+ + s_i^-) \tag{obj1MIP}$$

$$g_2 = \frac{1}{n \cdot m} \cdot \sum_{t \in M} \left(\frac{1}{d_t^*} \cdot \sum_{i \in N} (s_{it}^+ + s_{it}^-) \right) \tag{obj2MIP}$$

$$g_3 = \frac{2}{k \cdot (k-1)} \cdot \sum_{i,j \in K} z_{i,j} \tag{obj3MIP}$$

5 Local search for the PLP

As shown earlier, the PLP is an $\mathcal{NP}$-hard optimization problem; i.e., it belongs to a class of problems for which polynomial-time algorithms do not exist unless $\mathcal{P}$=$\mathcal{NP}$. Consequently, not all instances of the PLP can be solved in a feasible amount of time with the currently known exact

methods. Therefore, we should also take heuristic solution methods into account.

This section presents metaheuristic local search techniques to solve the PLP. First, a greedy approach to obtain initial solutions is presented. Afterwards, two neighborhood structures for the PLP are described, and finally, we explain the local search algorithm, namely simulated annealing.

5.1 Construction of initial solutions

We developed a greedy construction heuristic which is capable of constructing good initial solutions in $\mathcal{O}(n \cdot k^2)$ time. The parameters of the algorithm are a list of k orders and the number of periods n. The first step of the algorithm is to sort the orders by decreasing priority which is already an approximate handling of objective 3. Then we loop over all periods i from 1 to n, performing the following steps:

1. Examine sequentially the orders from the head of the sorted order list. Calculate for each order j that still fits into period i (considering the capacity limits) the change on the objective functions g_1 (obj1′) and g_2 (obj2′) which the inclusion of this order into period i would cause. We further call these changes in the objective functions the delta cost δ. The delta cost δ_{g_1} of g_1 of inserting order j in period i can be calculated as $|d^* - (w_i + d_j)| - |d^* - w_i|$. The calculation of δ_{g_2} works analogously, taking the product types into account, and the total delta cost δ is $\delta_{g_1} + \delta_{g_2}$. If $\delta < 0$ (i.e., including the order improves the objectives), order j is added to a list of suitable orders. The orders from the head of the sorted list are processed in this way until the suitable order list has size k/n (i.e., the average number of orders for each period) or there are no orders left.
2. Afterwards, if the list of suitable orders is not empty, pick one of the orders having the smallest delta cost, plan it for period i, remove it from the sorted order list, and go back to step 1.
3. Otherwise (if there was no suitable order), repeat with $i := i + 1$.

Finally, we check whether there are any orders left which could not be assigned due to the capacity limits. If that is the case, they are assigned one by one to the period with maximal remaining capacity. In this way, particularly those periods which are not filled well are assigned the remaining orders, and the probability of a hard constraint violation is minimized. However, violating the maximum capacity constraint is allowed in this step because a complete assignment is required for the subsequent local search.

Let us briefly turn to the runtime of this construction heuristic: Sorting of the orders by priority can be done in $\mathcal{O}(k \cdot \log k)$ time. Then, for every period, to find the next order to include, we might need to check the entire list of orders. Calculating the delta costs for a given order can be done in constant time, as introducing one new order only affects one period and one product type. This process needs to be repeated at most k times per period which results in a runtime of $\mathcal{O}(k^2)$ per period or $\mathcal{O}(n \cdot k^2)$ in total. The last step of assigning remaining orders can be done in $\mathcal{O}(n \cdot k)$ time which leads to a total runtime of $\mathcal{O}(n \cdot k \cdot (\log k + k)) = \mathcal{O}(n \cdot k^2)$.

5.2 Neighborhood structures

We devised two types of moves for generating different neighborhoods of a solution which will be introduced in the following subsections. Furthermore, we briefly describe our delta evaluation approach that speeds up the calculations dramatically.

5.2.1 Move-order neighborhood

The move-order neighborhood (or simply move neighborhood) of a solution s consists of all solutions whose only difference from s is that one order has been moved to a different period. Figure 5 visualizes such a move. The figure on the left shows the leveling objective per product type before the move, and on the right side we can see the result of applying the move. Order 2 is moved from $P2$ to $P3$, which yields in this case a better solution.

Enumerating the move neighborhood involves iterating over k orders for each of $n - 1$ possible target periods; i.e., the neighborhood size is exactly $k \cdot (n - 1)$.

5.2.2 Swap-orders neighborhood

The swap-orders neighborhood (or simply swap neighborhood) of a solution s consists of all solutions s' whose only difference from s is that two orders not assigned to the same period in s appear with swapped period assignments in s'. Figure 6 visualizes such a move. Order 1 is swapped with order 2, which in this case again yields a better solution.

Enumerating the swap neighborhood involves iterating over all pairs of orders not assigned to the same period. Hence, the neighborhood size is in $O(k^2)$.

5.2.3 Move evaluation

If we want to compare two moves a and b with respect to their quality, we define that the first criterion to check is the number of hard constraint violations which each of them introduces or resolves. If a introduces fewer or resolves more of them, we say that a is better than b. Otherwise—if the number of hard constraint violations is equal—the comparison is done by selecting the one which has the lower move cost, which

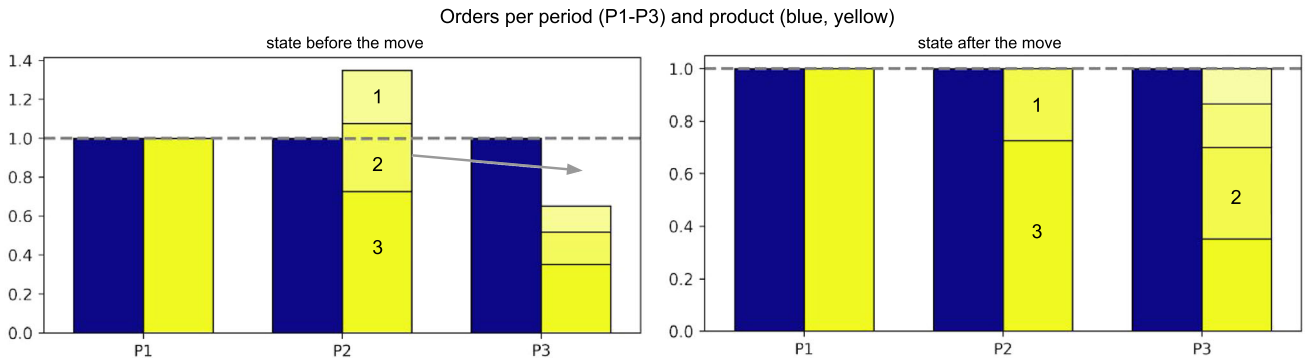

Fig. 5 Example of a move-order move: solutions before (left) and after (right) (Color figure online)

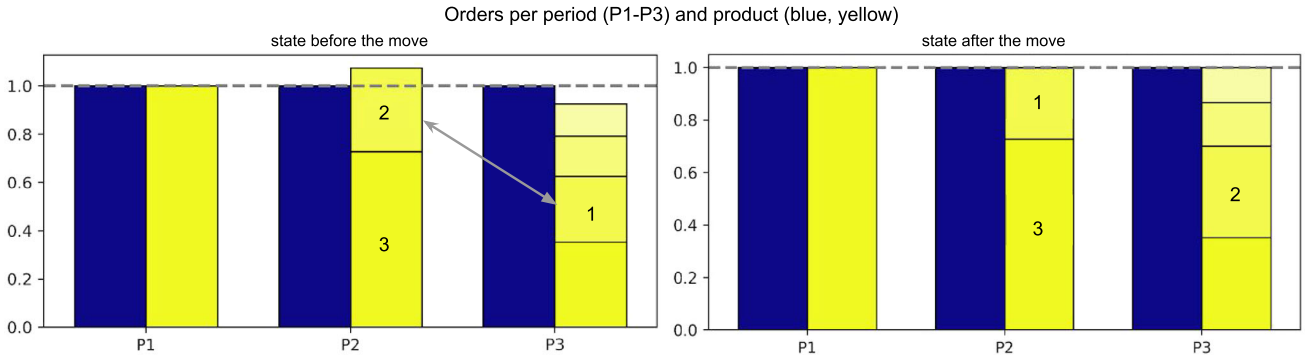

Fig. 6 Example of a swap-orders move: solutions before (left) and after (right) (Color figure online)

is defined as the change of the current solution's objective value in case the move is performed.

To avoid costly complete evaluations of whole solutions, a delta evaluation procedure is proposed that efficiently evaluates how much the objective value changes for a given move. The delta cost of moving an order to a different period is calculated for the three objective function components separately:

1. For the leveling objective, the planned total production volume for each period needs to be cached. This makes it possible to calculate the planned production volume after the move efficiently.
2. The per-product leveling objective allows a similar approach, given that we keep track of the planned production volume for each period and product type.
3. The priority objective is the hardest and most time-consuming part of delta evaluation because moving an order from period i to j can introduce or resolve inversions between the moved order and every order assigned to a period between i and j. When the number of orders is very large, it is inefficient to iterate over all such orders and perform comparisons because the delta evaluation needs to be repeated for every candidate move. Our idea for optimizing this evaluation is based on the insight that the only thing we care about when moving an order past a period is the number of orders in that period which have smaller and larger priorities, respectively, not the actual priority values. Therefore, the priority values of all orders assigned to a certain period can be maintained in a sorted list (one for each period). This allows us to efficiently retrieve via binary search how many orders have smaller/larger priorities than the order which we currently want to move.

This efficient delta evaluation function for moving an order to a different period can be reused to create an efficient implementation of delta evaluation for the swap neighborhood as well. The cost of swapping two orders o_1 and o_2 is equivalent to the cost of moving o_1 to the period of o_2 plus the cost of moving o_2 to the period of o_1 plus a compensation term for priority inversions that have been potentially introduced or resolved by the swap.

The delta cost of the three objective function components is aggregated to a single value by the usual formula for the objective value (obj).

5.3 Simulated annealing algorithm

This section presents details of the metaheuristic local search method which we investigated for solving the PLP, namely simulated annealing.

Simulated annealing is a metaheuristic optimization method introduced by Kirkpatrick et al. (1983). It resembles the physical process of annealing in metallurgy insofar as both methods use a cooling schedule to control the amount of random movements in the process, which encourages convergence to the optimal state. Even though convergence to the optimal solution is usually not achieved in practical settings, simulated annealing is still one of the most widely used metaheuristic optimization methods.

The version of simulated annealing we propose is described in Algorithm 1. Given an initial solution, a set of neighborhoods $\mathcal{N}_i$ with associated probabilities p_i, the starting temperature $t_{\max}$, the minimum temperature $t_{\min}$, and the number w of iterations per temperature, a solution to the PLP is returned once the given time limit or the given iteration limit is reached. The objective value of the returned solution is inferior to or equal to the objective value of the initial solution.

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 the 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;
    t ← Cool-Down(t);
return bestSolution;
```

At every iteration step for a given temperature t, a neighborhood $\mathcal{N}$ is chosen at random according to the probabilities p_i, and a random move m from $\mathcal{N}$ is selected. The selected move is accepted and applied on the current solution if the acceptance criterion **Accept**(m, t) is met. Once the number w of iterations per temperature is reached, the temperature is reduced according to the cooling schedule **Cool-Down**(t). These two functions are defined as follows:

– *Acceptance criterion* We use the metropolis criterion as acceptance function, which was introduced in the original paper by Kirkpatrick et al. (1983). The probability of acceptance $P(i \Rightarrow j)$ of a move from solution i to solution j (for the case of minimization), with $f(x)$ standing for the objective value of solution x, can be defined as follows:

$$P(i \Rightarrow j) = \begin{cases} 1, & \text{if } f(j) \leq f(i). \\ exp\left(\frac{f(i)-f(j)}{t}\right), & \text{otherwise.} \end{cases} \tag{13}$$

If the candidate solution j is at least as good as the current solution i, it is accepted unconditionally. Otherwise, it is accepted with a probability which is decreasing exponentially as a function of the negative delta cost divided by the current temperature. This means that if a candidate solution is much worse than the current one, it will be accepted with a lower probability than a solution which is just slightly worse.

– *Cooling schedule* The temperature is decreased during the search process by means of a cooling schedule, which is usually a geometric row. In our case it depends on the cooling rate α and the iterations per temperature level w. The function **Cool-Down()** reduces the temperature after every w iterations by the following formula:

$$t_i = \alpha \cdot t_{i-1} \tag{14}$$

We now want to briefly stress how α and w interact. Given an iteration limit l, the initial temperature $t_{\max}$ and the final temperature $t_{\min}$, there exist many different options to reach $t_{\min}$ after l iterations, namely all combinations of α and w such that $t_{\min} = \alpha^n \cdot t_{\max}$, where the number of temperature steps $n = \left\lfloor \frac{l}{w} \right\rfloor$. Two examples of schedules following that formula with $l = 30000$, $t_{\max} = 1$, and $t_{\min} = 0.001$ are depicted in Fig. 7. Please observe that for both options depicted in the figure, the temperature at each time is approximately the same, as the different step sizes and widths compensate each other. Therefore, it is sufficient to fix the cooling rate α when tuning the parameters of simulated annealing and let the cooling schedule be determined only by the variation of w.

If the number l is unknown, we can also derive a formula which relates two cooling schedules (α_1, w_1) and (α_2, w_2) that have the same slope:

$$\frac{w_1}{w_2} = \frac{\log \alpha_1}{\log \alpha_2} \tag{15}$$

Using this relationship, one can construct alternative cooling schedules which decrease equally rapidly on average.

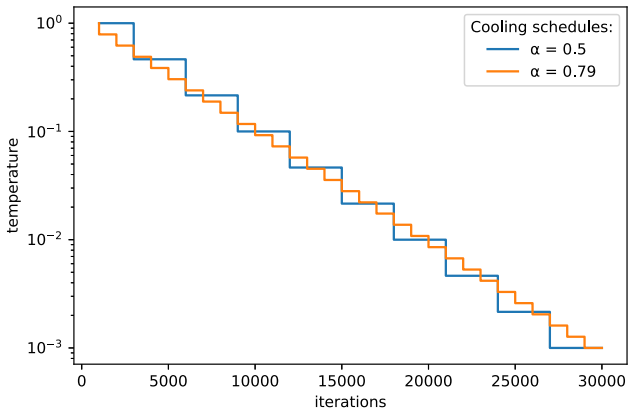

Fig. 7 Two cooling schedules with different cooling rates and iterations per temperature but identical start and end points (Color figure online)

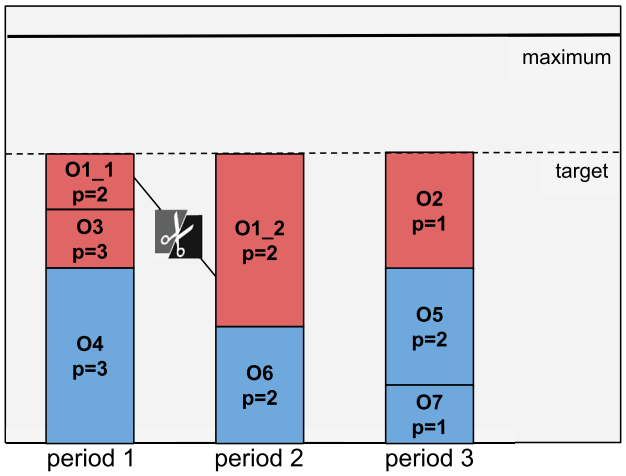

Fig. 8 Example solution, where order $O1$ (with priority 2) is split between periods 1 and 2. This allows for an even better leveling with respect to the first two objectives without introducing new priority inversions (Color figure online)

6 Extensions to the basic production leveling problem

Some real-life midterm planning scenarios cannot be tackled with the standard PLP problem formulation, as it does not consider the availability of resources used during production. This prevents us, for example, from taking discrete resources such as staffing and the availability of machines or continuous resources such as power consumption into account when distributing orders over production periods.

Furthermore, the original specification of the PLP assumes that each order is indivisible. This implies that it cannot be used in any practical context where single customer orders can actually be distributed over multiple periods in the planning horizon. Figure 8 shows with the help of an example why splitting orders can be useful: The additional flexibility that is obtained by allowing orders to be split can help to create solutions that are good in terms of both leveling and prioritization. For this specific example, splitting one order into two parts allows us to achieve a substantially better solution: the objective value of the solution presented in Fig. 8 is $g = 0.67$, whereas the optimal solution without splits reaches $g = 0.87$ (see Fig. 4).

Another aspect that is important in many practical applications of the PLP is the incorporation of due dates. Earlier in the paper, we mentioned that our industrial partner has deployed solution methods to the PLP in the area of electronic component manufacturing, where due dates and customer priorities are modeled in terms of order priorities. However, other industrial applications need explicit control of due dates, where orders should be finished at latest by a certain production period. Furthermore, it can be useful to combine such a due date objective with the consideration of order priorities, as for example customer priorities could hardly be modeled in form of due date constraints.

In this section, we introduce an extended problem formulation of the PLP that supports order splitting, the consideration of resource constraints and due dates. First, the extended production leveling problem PLP$^+$ is formally defined. Then we provide an extension to the MIP model introduced in Sect. 6.2 as well as four neighborhood relations that allow us to use the simulated annealing algorithm presented in Sect. 5.3 for PLP$^+$.

6.1 The extended production leveling problem

Instances to the PLP$^+$ are defined by the same parameters as specified for the PLP in Sect. 2, but include additional parameters that determine how many splits per order are feasible and provide the parameters for the additional constraints. The following lists the additional input parameters for the PLP$^+$:

Input parameters

$R = \{1..o\}$	Set of resources where o is the number of 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
$ps_j^{\min} \in \{1..d_j\}$	the minimum size of any part of order $j \in K$
$pc_j^{\max} \in \mathbb{Z}^+$	the maximum number of parts of order $j \in K$
$ru_{j,r} \in \mathbb{R}_0^+$	the amount of usage of 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$^+$have to capture additional information to the PLP. The following list contains the variables for the PLP$^+$:

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 part 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 part of an order j are planned:

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

– 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 $u_{i,r} \in \mathbb{R}_0^+$ capture the total usage of resources in each of the planning periods, where the resource usage of a single order part is determined relative to the total order usage.

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

Hard constraints

In addition to the two production volume hard constraints from the PLP, the PLP$^+$defines another two hard constraints that restrict the minimum size of a part and the maximum number of parts an order can be split into:

– H3: The minimum part size is satisfied for every part 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 number of parts 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 Sect. 2 are also used in the multi-objective function of the PLP$^+$. 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$^+$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$^+$:

– 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 in which i finishes only after j starts.

$$f_3 = \left| \left\{ (i, j) \in K^2 : p_i > p_j \text{ and } y_i^{\text{end}} > y_j^{\text{start}} \right\} \right| \tag{obj3}$$

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

$$f_4 = \sum_{j \in K} \Big(\max(pd_j^{\min} - y_j^{\text{start}}, 0) + \max(y_j^{\text{end}} - pd_j^{\max}, 0) \Big) \tag{obj4}$$

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

$$f_5 = \sum_{r \in R} \sum_{i \in N} \begin{cases} 1 - \frac{u_{i,r}}{ru_r^{\min}} & \text{if } u_{i,r} < ru_r^{\min} \\ \frac{u_{i,r}}{ru_r^{\max}} - 1 & \text{if } u_{i,r} > ru_r^{\max} \\ 0 & \text{otherwise.} \end{cases} \tag{obj5}$$

Whereas objectives f_1, f_2, f_3 can be normalized as specified in Sect. 2, objectives f_4 and f_5 are normalized as follows:

$$g_4 = \frac{1}{n \cdot k} \cdot f_4 \tag{obj4'}$$

$$g_5 = \frac{1}{n \cdot o} \cdot f_5 \tag{obj5′}$$

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 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$^+$is the following weighted sum (with user-defined weights $a_1 - a_5$).

$$\begin{aligned}\textbf{minimize } g = & a_1 \cdot g_1 + a_2 \cdot g_2 + a_3 \cdot g_3 \\ & + a_4 \cdot g_4 + a_5 \cdot g_5\end{aligned} \tag{obj+}$$

6.2 Integer programming model

In Sect. 4, an integer programming model for the PLP was proposed. This model is now extended to the PLP$^+$based on the formal problem description specified in the previous section.

In contrast to the integer programming model for the PLP, the former binary decision variables x_{ij} are converted to an integer domain to model the planned production amount for order j in period i and thus account for order splits.

The additional variables are defined as follows:

$x_{ij} \in \mathbb{Z}^+$	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 part exists in period i. A part exists for order j in period i iff $x_{ij} > 0$.
$y_j^{\text{start}} \in N$	for each order $j \in K$ the period assignment of its first part
$y_j^{\text{end}} \in N$	for each order $j \in K$ the period assignment of its last part
$u_{ir}^+ \in \mathbb{R}^+$	for each $i \in N, r \in R$ the amount of overusage of resource r in period i
$u_{ir}^- \in \mathbb{R}^+$	for each $i \in N, r \in R$ the amount of underusage of resource r in period i
$v_j^{\text{start}} \in \mathbb{Z}^+$	for each $j \in K$ the amount of violation of the earliest period soft constraint
$v_j^{\text{end}} \in \mathbb{Z}^+$	for each $j \in K$ the amount of violation of the latest period ($\widehat{=}$ due date) soft constraint

This allows us to express the two additional objective functions as follows:

$$g_4 = \frac{1}{2k} \cdot \sum_{j \in K} \left(v_j^{\text{start}} + v_j^{\text{end}}\right) \tag{obj4MIP}$$

$$g_5 = \frac{1}{n \cdot o} \cdot \sum_{r \in R} \sum_{i \in N} \left(\frac{u_{ir}^-}{ru_r^{\min}} + \frac{u_{ir}^+}{ru_r^{\max}}\right) \tag{obj5MIP}$$

Together with the definition of g_1 and g_2 given in Sect. 4, we have the following formulation of the PLP$^+$:

minimize $a_1 g_1 + a_2 g_2 + a_3 g_3 + a_4 g_4 + a_5 g_5$

such that

$$\sum_{i \in N} x_{ij} = d_j \; j \in K \tag{16}$$

$$x_{ij} \leq d_j \cdot \hat{x}_{ij} \; j \in K \tag{17}$$

$$y_j^{\text{start}} \leq i + (n-1) \cdot (1 - \hat{x}_{ij}) \; j \in K, i \in N \tag{18}$$

$$y_j^{\text{end}} \geq i \cdot \hat{x}_{ij} \; j \in K, i \in N \tag{19}$$

$$x_{ij} \geq ps_j^{\min} \cdot \hat{x}_{ij} \; j \in K \tag{20}$$

$$\sum_{i \in N} \hat{x}_{ij} \leq pc_j^{\max} \; j \in K \tag{21}$$

$$y_i^{\text{end}} - y_j^{\text{start}} \leq (n-1) z_{ij} \; i, j \in K \mid p_i > p_j \tag{22}$$

$$v_j^{\text{start}} \geq pd_j^{\min} - y_j^{start} \; j \in K \tag{23}$$

$$v_j^{\text{end}} \geq y_j^{\text{end}} - pd_j^{\max} \; j \in K \tag{24}$$

$$\sum_{j \in K} ru_{j,r} \cdot \frac{x_{ij}}{d_j} - u_{ir}^+ \leq ru_r^{\max} \; i \in N, r \in R \tag{25}$$

$$\sum_{j \in K} ru_{j,r} \cdot \frac{x_{ij}}{d_j} + u_{ir}^- \geq ru_r^{\min} \; i \in N, r \in R \tag{26}$$

$$y_i^{\text{end}} \leq y_j^{\text{start}} \; i, j \in K \mid p_i \geq p_j, d_i = d_j, t_i = t_j \tag{27}$$

and that Eqs. (7)–(10) as well as (12) from Sect. 4 are fulfilled.

Constraints (16)– (19) link auxiliary variables to the decision variables. Constraints (16) ensure that the total demand of all parts of j equals the order's demand d_j. Constraints (17) link the binary variables $\hat{x}_{ij}$ to the decision variables x_{ij} such that $x_{ij} > 0 \rightarrow \hat{x}_{ij} = 1$. Constraints (18) and (19) 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 part of an order, respectively.

The sets of constraints (20)– (21) model the problem's hard constraints. Constraints (20) enforce the minimum part size and (21) enforce the maximum number of parts into which an order may be split.

Constraints (22)– (26) populate penalty variables for the objective function. Constraints (22) link the y_i^{start} and y_i^{end} to

the $z_{i,j}$ variables which track the number of priority inversions. This ensures 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 set of constraints (23) and (24) force the variables v_j^{start} and v_j^{end} to keep track of the violations of the allowed assignment range. Constraints (25) and (26) force u_{ir}^+ and u_{ir}^- to contain the amount of over-usage and under-usage of resource r in period i. This is achieved by comparing with the amount of planned resource usage which is given by the first summand.

Finally, there is one additional set of redundant constraints for strengthening the formulation: Constraints (27) enforce a dominance relation for all pairs of orders which have the same product type and demand value, in a similar fashion as in Eq. (11).

6.3 Neighborhood relations for simulated annealing

To be able to use our simulated annealing algorithm described in Sect. 5.3 for the PLP$^+$as well, we propose four neighborhood relations when approaching the PLP$^+$with local search: The first two are extensions of the search neighborhoods to the PLP that we defined in Sect. 5. Instead of moving and swapping complete orders, the new versions move and swap single order parts. The third neighborhood splits and merges order parts and is able to exchange demands between parts of an order. Finally, the fourth neighborhood shifts all parts of a single order at once.

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

Move-part neighborhood The move-part neighborhood of a solution s consists of all solutions s' whose only difference from s is that one part of some order has been moved to a different period. When splits are disallowed and thus every order has exactly one part, 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 part as well as the target period.

Swap-parts neighborhood The swap-parts neighborhood of a solution s consists of all solutions s' whose only difference from s is that two parts 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 parts to be swapped are chosen uniformly at random among all pairs of parts 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 part of that order is split into two parts, and the new part is moved to some other period.
- Two parts of that order are merged.
- The sizes of two parts of that order are changed such that the total size of the two parts together stays the same, and both parts remain non-empty. In other words, demands are moved from one part to another.

When the 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, and whose parts all have been shifted one period to the left or to the right. If a part cannot be shifted any more because it is already in the first or last period, it remains in that period. When the sampling random shift order moves, orders are shifted to right or left with equal probability.

7 Experimental evaluation

In this section, we evaluate the practical contributions of our work. As the PLP is a new problem, we initially elaborate on the problem instances and propose two instance generation procedures. The next subsection describes properties of the test set, defines parameters, and describes the processing environment. After that, we turn towards the actual evaluation and look at the MIP model in detail. Then the metaheuristic approach is extensively evaluated. Finally, we evaluate the exact and metaheuristic approach for the PLP$^+$.

7.1 Problem instances

In this paper, we have used real-life instances and randomly generated instances, both of which will be described below. The set of test instances is publicly available on the following web page: https://dbai.tuwien.ac.at/staff/jvass/production-leveling.

7.1.1 Real-life instances

The PLP emerges from a real-life use case of our industrial partner which also provided us some data from the production system. In total, we received 27 real-life PLP instances which all have 20 periods, four to eight product types, and 79 to 1585 orders. This set of instances is referred to as R_1. As the small number of instances in this instance set does not suffice for a thorough evaluation, and because we also want to test the scalability of our methods on even larger instances, we designed two random instance generation procedures which are described in the following.

7.1.2 Randomly generated instances

We devised two random instance generators, one which produces instances whose optimal solution is known by design and another one producing random instances without any known solution. Both random generation algorithms as well as their parameters have been developed and selected together with expert practitioners to achieve instances that are similar to real-life instances from the electronics industry.

Perfectly solvable instances We devised a construction method for generating instances which allow for a perfectly balanced solution with zero cost. That is, of course, a restriction of generality, but it is useful as a means of evaluating the solution quality for large instances for which no solutions can be found using exact methods and reasonable computational resources. Despite the existence of a perfectly balanced solution with no priority inversions, the instances are still not easy to solve to optimality, at least not as long as the information of perfect realizability to the solvers is not provided.

The instance generation process relies on the subroutine for random integer partitioning shown in Algorithm 2. It takes as arguments the integer to partition, the number of parts in the partition, and a minimum value *minV* for each partition. The main idea is to represent the number n as an array of $n - k \cdot minV$ zeros and then inserting $k - 1$ ones at random positions. In the resulting array, an integer partition of the number $n - k \cdot minV$ into k parts can be found by looking at the number of zeros between every two neighboring ones. Finally, we add $minV$ to every element of the resulting array to obtain the requested partition with minimum value *minV*.

Algorithm 2: Integer partitioning algorithm

Data: n, k, $minV$
Result: An array with k integers whose sum equals n, each of which being $\geq minV$
1 **let** *array* ← an array consisting of $n - (k \cdot minV)$ zeros;
2 Insert $k - 1$ ones into *array* at random positions;
3 **let** *spaces* ← number of zeros between the ones in *array*;
4 add *minV* to every element of *spaces*;
5 **return** *spaces*;

Using this partitioning algorithm, Algorithm 3 defines the procedure for generating random instances with a fixed number of orders, periods, and product types, as well as a given average demand per order. First, the total number of orders is partitioned into one part for each period, where each part has to have at least as many orders as we have product types. This is important because the target for every product type needs to be met in every period to achieve an objective value of 0. The same thing is done for each period to decide upon the number of orders for each product type.

Next, we draw the overall target value for the production volume (which is the same for each period) by taking the desired *avgDemandPerOrder* and multiplying with the average number of orders per period plus a random deviation of at most 10%. This value is partitioned into one part for each product, which is the demand for each product per period.

Finally, we need to partition the demand for each product, which has been determined in line 4, into the number of orders for each period and product that has been calculated in line 2. The priorities must be chosen such that no inversion can exist, which is achieved by assigning each period a range of priority values decreasingly such that the ranges do not overlap, and randomly choosing for each order one of the allowed values. From these data, the list of orders can be built. Finally, the capacity limits are set to be 10% above the planned demands per period and product type. The optimal solution is also known from the construction process.

Algorithm 3: Procedure for the creation of a perfectly solvable instance

Data: m, n, k, *avgDemandPerOrder*
Result: A realizable instance with m product types, n periods, k orders, and the optimal solution
1 **let** *ordersPerPeriod* ← partition(k, n, m);
2 **let** *ordersPerPeriodAndProduct* ← partition(*ordersPerPeriod*[o], m, 1) for every order o;
3 **let** *plannedDemand* ← $\frac{k \cdot avgDemandPerOrder}{n} \pm 10\%$;
4 **let** *plannedDemandPerProduct* ← partition(*plannedDemand*, m, max(*ordersPerPeriodAndProduct*));
5 **let** *orderDemands* ← partition(*plannedDemandPerProduct*[p], *ordersPerPeriodAndProduct*[t, p], 1) for every period t and product p;
6 **let** *allowedPriorities* ← for each period n a distinct set of priorities s.t. they decrease with increasing n;
7 **let** *orderPriorities* ← choose for each order one of the priorities which are allowed according to the period of the order;
8 build the list of orders and product types and shuffle them;
9 assign random product names;
10 **let** *maxCapacity* ← *plannedDemand* + 10 %;
11 **let** *maxCapacityPerProduct*[p] ← *plannedDemandPerProduct*[p] + 10 % for every product type p;
12 **return** a new instance and an optimal solution;

Using Algorithm 3, we generated 1000 instances, sampling the parameters for each one independently as follows: The number of orders k is chosen from {100..4000}, the number of periods n from {2..80}, the number of product types m from {1..20}, and *avgDemandPerOrder* from {5..500}. The resulting set of instances is subsequently called R_2.

Random instances without a known solution We also devised a second instance generation procedure where the optimal solutions are not known by design, and it is not guaranteed that there exists a feasible one. The instances are designed to share some properties of the 27 realistic instances:

– There exist only a limited number $l \ll k$ of different order demand values. This means orders are frequently repeated (with varying priorities though).
– Orders of different product types draw their demand data from different distributions. Whereas product a may have demand values between 0 and 1000, product b may have it between 0 and 5000.
– Sometimes there exist product types where fewer orders than periods exist. This circumstance implies that in any solution, the planned demand in some of the periods will exceed the target, while for others it must be zero because orders can only be assigned to periods as a whole.

The actual generation process is very simple. Given a number of orders k, periods n, and product types m, the algorithm works as follows:

1. Partition the number of orders k into m parts $c_1 \dots c_m$.
2. Choose the maximum priority of all orders $p_{max} \in [1; 3n]$
3. Choose between 1 and 50 allowed demand values $d \in [1; random(1000 - 5000)]$ for each product p, named D_p.
4. For each product $p \in [1; m]$, generate c_p orders, choosing the demand from the set D_p and the priority from $[0; p_{max}[$.
5. 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.

Using this procedure, we generated the instance set R_3 consisting of 1000 instances by sampling the parameters randomly. The number of orders k is chosen from {100..4000}, the number of periods n from {2..80} and the number of product types m from {1..20}. Furthermore, we generated a set of 10 small instances, named R_4, where the number of orders k is chosen from {30..100}, the number of periods n from {5..20} and the number of product types m from {1..5}.

7.2 Experimental setting

The instances which are described above are split into a training and a test set, so that the parameter tuning runs on the training set and the evaluations are performed on the test set. The test set consists of the whole set of realistic instances R_1, 50 instances of R_2, 50 instances of R_3, and all 10 instances in R_4. Table 2 provides an overview of the instance sets and the way they were split.

We chose to build the test set out of four different instance types to ensure that our algorithms can cope with different characteristics and sizes. The size distribution is shown by Table 3, which states for each instance parameter—k (number of orders), m (number of product types), and n (number of periods)—the minimum, maximum, and mean value on each part of the test set. The smallest instances are R_4, followed by the realistic instances R_1. The instances coming from R_2 and R_3 are much larger on average, as we also want to evaluate the scalability of our algorithms.

Hard constraint violations are not part of the objective function but undergo a special treatment by reporting the number of violated constraints as a separate number or separate plot. In some cases, e.g., statistical significance tests, we handle objective and constraint violations at once by adding them up. Due to the small magnitude of the objective, a penalization factor for hard constraint violations is not necessary.

In cooperation with our industrial partner, we decided to set the default values for the weights of the objective function components as follows: $a_1 = 1$, $a_2 = 1$, $a_3 = 1/3$, $a_4 = 1$, and $a_5 = 1$. All experiments of the evaluation use this weighting.

Wherever nothing different is stated, the algorithm parameters are defined as follows:

– The parameters of simulated annealing are tuned automatically. The concrete process and the results are described later on.
– The MIP model is executed using Gurobi Optimizer 8.1.1 (Gurobi Optimization 2019) on a single thread, a time limit of 1 h, and otherwise the default settings.

All experiments were conducted 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. The metaheuristic algorithms are implemented in C# and executed using Mono 4.2.1.

7.3 Evaluation of the MIP model

In this subsection, the MIP model presented in Sect. 4 is investigated with respect to its empirical performance. We first break down the results by the different instance sets of which the test set is composed. Afterwards, we will investigate how the instance size affects the solution quality.

The first experiment explores how well each part of the test set can be solved using MIP. For a description of the different parts, please refer to Table 3. Figure 9 visualizes the shares of optimally solved, feasible but not optimally solved, infeasible, and unsolved instances per group R_1–R_4.

The most noticeable difference between the sets is that in R_2 and R_3, the vast majority of the instances are unsolved, while for the other two, most of the instances are solved (but still not proven optimal). Presumably, this is because most

Table 2 Overview of the different instance sets and the split into training and test sets

	Name	Count	Description	Training set selection	Test set selection
R_1	`realistic_instance`	27	Realistic instances	–	01–27
R_2	`randomly_perfect`	1000	Randomly generated perfectly solvable	0001–0950	0951–1000
R_3	`randomly_generated`	1000	Randomly generated	0001–0950	0951–1000
R_4	`randomly_generated_small`	10	Randomly generated, small	–	1–10

Table 3 Minimum, maximum, mean, and standard deviation of number of orders k, number of product types m, and number of periods n for every part of the test set

Parameter	Test set	min	max	mean	std
k	R_1	79	1585	307.19	412.56
	R_2	105	3896	1595.86	954.09
	R_3	112	3991	2076.76	1207.02
	R_4	34	98	61.20	19.70
m	R_1	4	8	6.93	1.24
	R_2	1	19	8.82	5.50
	R_3	1	19	9.04	5.48
	R_4	1	4	2.80	1.03
n	R_1	20	20	20.00	0.00
	R_2	4	78	39.50	22.48
	R_3	4	77	39.26	22.04
	R_4	7	18	10.90	4.04

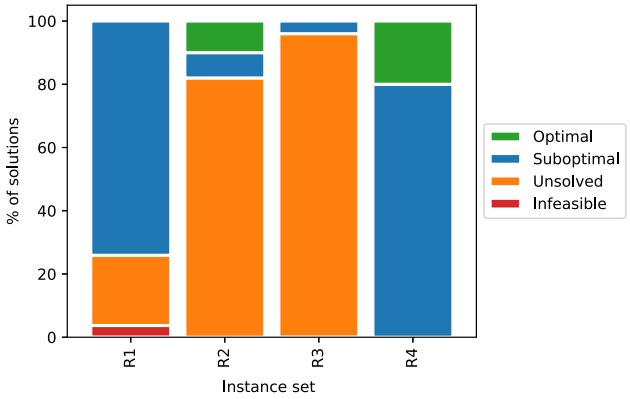

Fig. 9 Share of solution statuses of MIP for each subset of the test set. Optimal means proven optimal. Suboptimal implies that a solution has been found but it has not been proven that it is optimal. Unsolved means that within the time limit, no solution has been found, and it is thus unclear whether there exists a feasible solution at all. Infeasible means that the solver proved that no feasible solution exists (Color figure online)

of the instances in these sets are very large. Interestingly, however, about 10% of the instances in R_2 could be solved to proven optimality, but not a single one in R_3, even though the instance sizes of the two sets were sampled from the same distribution. One potential reason is that the instances in R_2 are designed to have optimal solutions with objective value 0. This should make optimality proofs easy for the solver once the optimal solution has been found, because no part of the objective function can be negative.

For over 70% of the instances in the sets R_1 and R_4, the MIP model was able to find a solution which, however, could not be proven to be optimal. We want to investigate the quality of these solutions and use for that purpose the *relative optimality gap* with respect to the best lower bound. It is defined as the following percentage:

$$gap = 1 - \frac{b}{s} = \frac{s - b}{s} \in [0, 1], \tag{28}$$

where s is the incumbent cost, i.e., the best upper bound, and b is the best lower bound on the solution cost provided by the MIP solver. Clearly, if the solution found is optimal, the gap is equal to zero; the larger the deviation between upper and lower bounds, the closer the gap is to a value of 1.

Table 4 Optimality gap of MIP for suboptimal instances in R_1 and R_4

	Min (%)	Max (%)	Mean (%)	Std dev (%)
R_1	0.99	11.65	3.96	2.55
R_4	0.63	98.85	31.14	41.52

Table 4 shows the minimum, maximum, and mean optimality gap as well as the standard deviation for all suboptimal solutions in R_1 and R_4. With an average gap of only 3.96%, the realistic instance set R_1 is solved very well, so that the MIP model might be usable in practice when the instances are not too large and a runtime of 1 h is not an issue. On the other hand, R_4 has a low minimum and a high maximum gap, as well as a large standard deviation. This means that the randomly generated instances are quite difficult to solve using MIP, even though those in R_4 are generally smaller than the realistic ones.

Finally, we investigate in greater detail how the instance size correlates with the results of the MIP model. Figure 10 visualizes the solution statuses of all instances in the test set, grouped by the number of orders k, the number of product types m, and the number of periods n, from left to right. The most apparent relationship is a correlation between the num-

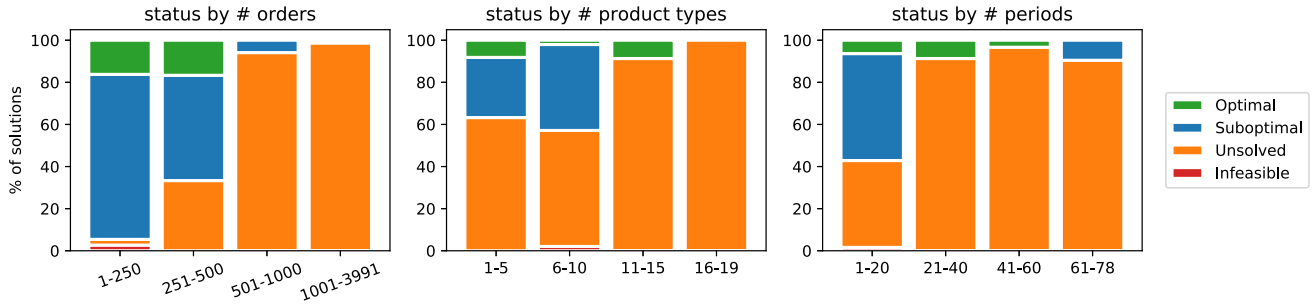

Fig. 10 Solution statuses of MIP on the test set, grouped by value ranges of instance features. Optimal means proven optimal. Suboptimal implies that an integer solution has been found, but it has not been proven that it is optimal. Unsolved means that within the time limit, no integer solution has been found, and it is thus unclear whether there exists a feasible solution at all. Infeasible means that the solver proved that no feasible solution exists (Color figure online)

ber of orders k and the percentage of unsolved instances. While below 250 orders almost every instance has been proven either feasible or infeasible, the share of unsolved solutions increases drastically when increasing k. The *status by # product types* and *status by # periods* plots show that the share of unsolved instances also increases with an increasing number of periods and product types, but it starts already quite high in the smallest bin. We can conclude that instances with 250 or fewer orders can be solved with high probability by the MIP model, but there is no such bound which we can state on the number of periods or product types. While increasing n and m clearly complicates the problem, making them small does not automatically make the problem easy to solve.

7.4 Evaluation of metaheuristics

As we have seen, the MIP formulation is not suitable for solving large instances, which is why local search methods for the PLP have been investigated as well. In this subsection, we will first deal with the automatic parameter tuning for simulated annealing and validate the claim that it is sound to fix the cooling rate. Thereafter, another experiment investigates the sensitivity of simulated annealing to variations in the weighting of the neighborhoods. Finally, we examine how close the metaheuristic solutions get to the global optima by using dual bounds obtained through MIP and the perfectly solvable instance set R_2.

7.4.1 Algorithm configuration

As described in Sect. 5.3, simulated annealing depends on parameters whose setting has a huge influence on the algorithm's efficiency and effectiveness. We deal with their configuration by means of sequential model-based algorithm configuration (SMAC), an automatic algorithm configuration tool written in python. It relies on Bayesian optimization in combination with an aggressive racing mechanism to efficiently search through huge configuration spaces (Lindauer et al. 2019).

We applied SMAC to tune the parameters of simulated annealing as it was presented above. The set of instances which was used for the tuning can be found in the column *Training set selection* of Table 2. The parameter optimization was executed for 24 h on 24 cores in parallel with a time limit of 5 min per run and no iteration limit. The cooling rate was not tuned but was set to a value of 0.95, which is not a restriction of generality as long as the number of iterations per temperature can still be adjusted (see Fig. 7). This claim will be verified in a separate experiment later on.

We tuned the initial temperature $t_{\max}$, the number of iterations per temperature w, and the probability p that the move neighborhood is used to generate the next random move (hence, $1 - p$ is the probability of the swap neighborhood). Tuning the minimum temperature $t_{\min}$ is not necessary, because the results cannot get worse when simulated annealing is run until the time limit instead of aborting when the minimal temperature is reached. Indeed, preliminary results showed that setting the minimum temperature to zero instead of using the tuning results of SMAC improves results to a small but significant extent. The configuration space with minimum and maximum values as well as the defaults and the tuning result is shown in Table 5.

7.4.2 Experiments about fixing the cooling rate

We claimed in Sect. 5.3 that the cooling rate α could be set to a constant value because it was redundant as long as the number of iterations per temperature w is free. During the algorithm configuration, this had already been taken into account by setting $\alpha \leftarrow 0.95$. Now we want to verify this claim by means of an experiment. Four more cooling schedules are derived from the one defined by the result of parameter tuning whose temperature profile follows the same slope. Then each

Table 5 Configuration space of simulated annealing

Parameter	Type	Minimum	Maximum	Default	Tuned
Iterations per temperature	Integer	10^3	10^6	10^3	2.52×10^5
Move neighborhood probability (%)	Integer	0	100	50	40
Initial temperature	Real	0.1	10.0	5.0	0.22
Minimum temperature	Real (fixed)	0	0	0	0
Cooling rate	Real (fixed)	0.95	0.95	0.95	0.95

configuration is benchmarked on the whole test set ten times with different random seeds and taking the median of the objective values and number of constraint violations for each instance.

Section 5.3 already introduced Eq. (15), which allows us to derive cooling schedules with equal average slopes. We selected the alternative cooling rates of 0.5, 0.75, 0.9, and 0.99 and computed the associated values of w. A summary of the resulting cooling schedules is shown in Table 6.

Figure 11a shows a box plot for each of the schedules, each of them plotting the median objective value resulting from the derived schedule divided by the median objective value resulting from the original schedule, per instance. The original schedule does, of course, not differ from itself, while the other schedules bring an improvement for some instances and worse results for others. For the three higher values of α, more than 50% of the data are in the range $\pm 1\%$, as are nearly all the rest in $\pm 2\%$ (except for a couple of outliers outside the plotting range). The median shown by the box plots of the schedules $\alpha = 0.9$ and $\alpha = 0.99$ is almost exactly at 1, whereas the other two variants have median differences slightly higher than 1, and the whiskers reach farther, although the differences are still very small. Figure 11b visualizes the same for the number of hard constraint violations, except that we do not divide but take the difference between the alternative schedules and the original one because the interpretation is more intuitive in this case. The whole box plot except for some outliers is zero, which means that for most of the instances there is no change in the number of hard constraint violations. However, there are a few outliers going down to -1, which means that for these instances, the alternative schedules have one fewer violation. Compared to the 137 instances of the test set, however, the number of outliers is very small.

To clarify whether the differences in the median of $\alpha = 0.5$ and $\alpha = 0.75$ are statistically significant, we conducted a Wilcoxon signed-rank test between the original schedule and each of the derived ones. To also take into account the hard constraint violations, their number was added to the objective value as a penalty. The null hypothesis was that the median of the differences would be zero, and the alternative that it would be different from zero. We want to reject the null hypothesis in the case of a p-value smaller than 0.05. The result of the tests yielded a p-value of $2.6 \cdot 10^{-8}$ for $\alpha = 0.5$, 0.041 for $\alpha = 0.75$, 0.966 for $\alpha = 0.9$, and 0.26 for $\alpha = 0.99$. This implies that we must reject the null hypothesis for the schedules with $\alpha \leq 0.75$. For the other two schedules, the statistical test does not let us reject the null hypothesis that the differences which we see are the result of chance.

To sum up, the claim that α can be changed without changing the result is valid in our setting as long as α is set high enough (i.e., in our experiment at least 0.9). This implies that we were on the safe side when fixing it to 0.95 during parameter tuning. For values $\alpha \leq 0.75$, the experiment showed a statistically significant 0.1% increase of the median objective value compared to the default configuration.

Table 6 Five equivalent cooling schedules which have the same slope on average. The value for w in the line with $\alpha = 0.95$ comes from parameter tuning, and the rest have been derived so that the slope does not change

Cooling rate α	Iterations per temperature w
0.50	3,412,581
0.75	1,416,349
0.90	518,723
0.95	*252,533*
0.99	49,481

7.4.3 Sensitivity to neighborhood weightings in simulated annealing

Before selecting the next random move in the simulated annealing algorithm, the neighborhood is chosen randomly according to some weighting. This weighting has been tuned by SMAC, resulting in a probability of 0.4 for the move neighborhood and 0.6 for the swap neighborhood. The tuning progress of SMAC revealed quite large fluctuations in this weighting. Therefore, we conduct a sensitivity analysis to find out what impact different weightings have on the results.

We evaluate six different weightings, one of which is the result of SMAC. The probability p for the move neighborhood in the six scenarios ranges from 0 to 1 in steps of 0.2,

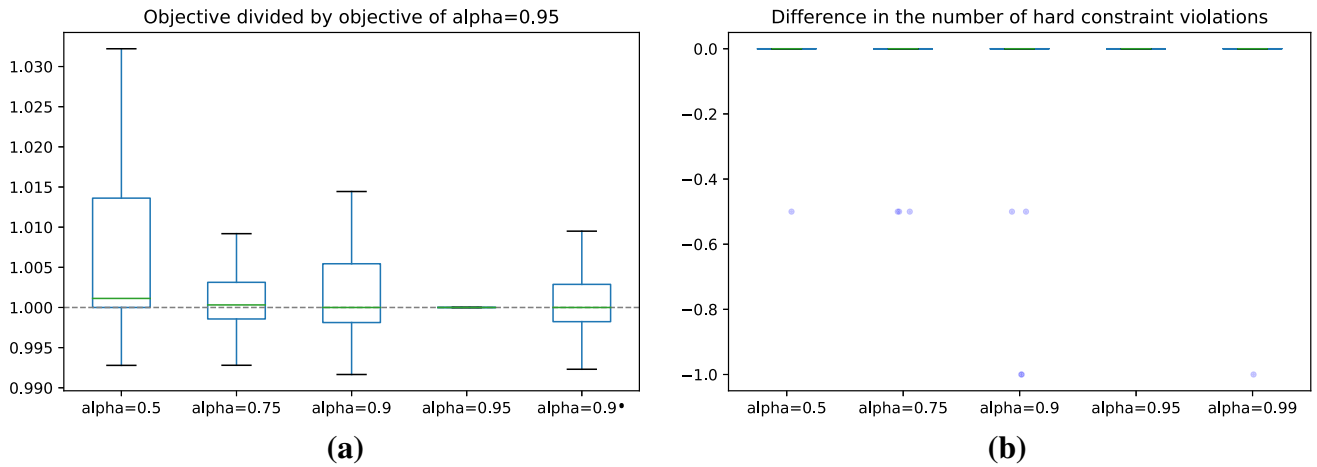

Fig. 11 Results of the experiment regarding different cooling schedules. All the results are comparisons per instance against the schedule with $\alpha = 0.95$

and the probability of the swap neighborhood is the complementary probability $1 - p$. Each configuration is executed on the test set 10 times with the usual time limit of 5 min. The runs are again aggregated using the median.

Figure 12 shows on the top a box plot for each of the alternative weightings, each of them plotting the associated objective value divided by the objective value of the original weighting. The labels "x - y" mean that the move neighborhood has weight x and that the swap neighborhood has weight y. The objective value becomes worse in the extreme cases, which can be seen because the leftmost and the two rightmost boxes lie completely above the dashed line. The other cases are practically equal, which means that the objective value does not change compared to the reference weighting.

In the plot shown at the bottom of Fig. 12, the difference of the number of hard constraint violations to the respective number of the reference weighting "40–60" is shown. All boxes contained in {0} are completely flat, which means that the interquartile range is equal for all weightings. The outliers show that a few instances of the extreme configurations have one or two more hard constraint violations than the reference configuration, while the weightings "20–80" and "60–40" have tendentially fewer. However, compared to the number of 137 instances contained in the test set, the number of outliers is very small, so that the significance of this difference must be doubted.

To sum up, the tested neighborhood weightings between "20–80" and "60–40" are equally good; therefore, it can be expected that the untested weightings in between are good as well. Using one of the other weightings leads to worse results, especially in the case where only the move neighborhood is used.

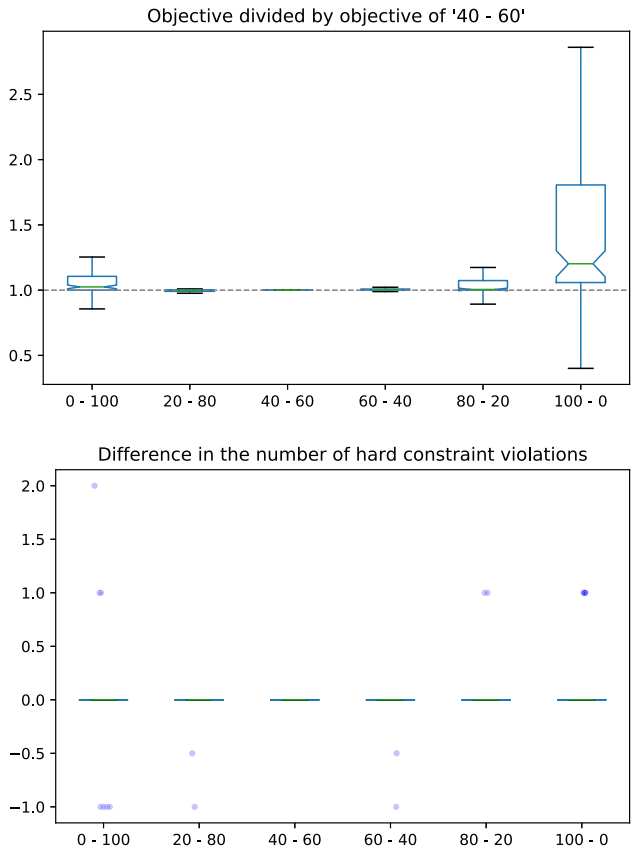

Fig. 12 Results of the experiment regarding neighborhood weightings. All the results are comparisons per instance against the schedule "40–60." The first number of each label is the weight of the move neighborhood and the second the weight of the swap neighborhood

7.4.4 Optimality gap of metaheuristic solutions

Experiments with our test data set showed that simulated annealing (with 5-min timeout) manages to solve 115 out of

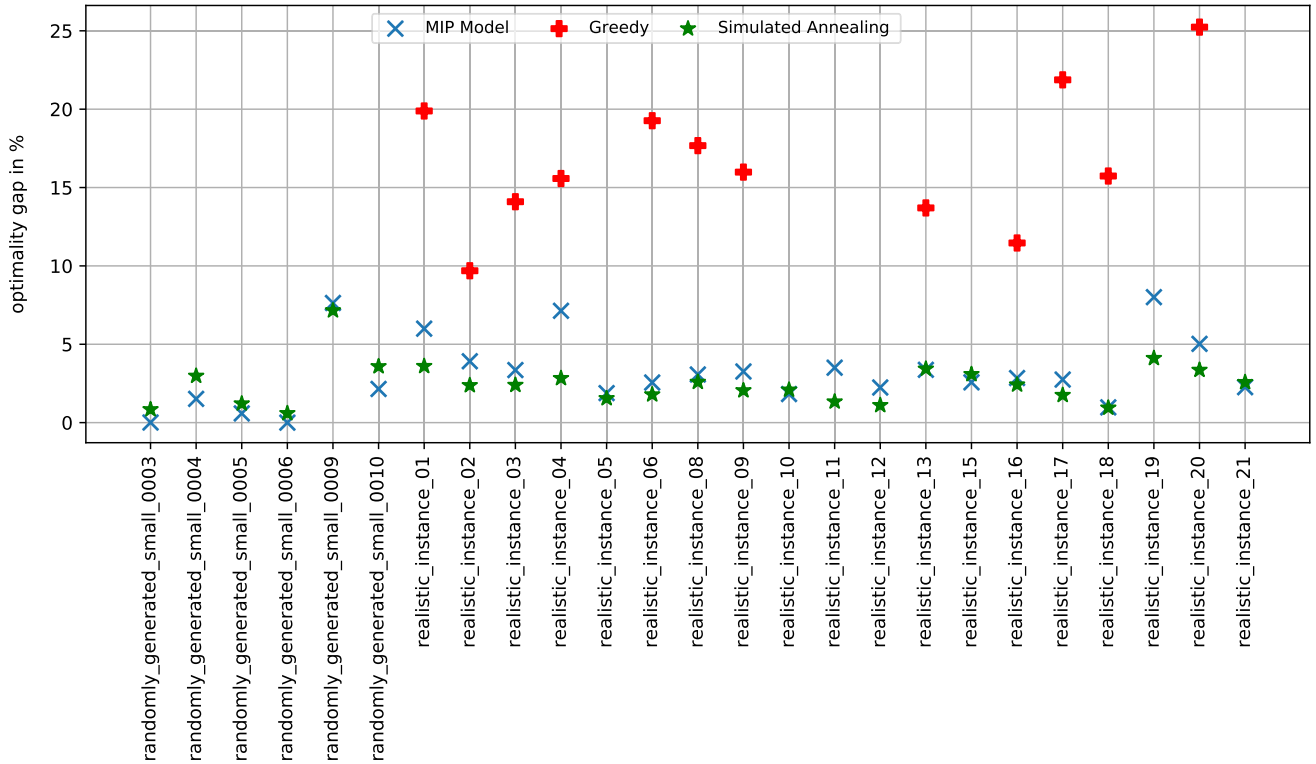

Fig. 13 Comparison of the optimality gap of valid solutions obtained through greedy, simulated annealing, and MIP. Marks are missing for solutions that violate constraints. The gap is computed using the dual bounds of MIP. The evaluation contains all instances of R_1 and R_4 which could be solved with an optimality gap of 10% or lower using MIP

137 instances without violations of hard constraints. This is in contrast to only 74 for the greedy approach. To be able to judge the quality of simulated annealing in a more systematic fashion, we consider the deviation from the optimal solution. As stated previously, this deviation is expressed in terms of the relative optimality gap as defined in Eq. (28) and is calculated by taking 1 minus a lower bound divided by the objective value. The rest of this section presents an evaluation of the size of the optimality gap of solutions obtained with simulated annealing. The optimality gap is calculated by using the lower bounds obtained through MIP. However, as MIP only solved a small fraction of large instances well enough to obtain good lower bounds, we needed to restrict this analysis to the instance sets R_1 and R_4.

Optimality gap for small instances: We want to analyze the optimality gap of the solutions produced by our metaheuristic approaches. Therefore, the best dual bound found by the MIP solver can be used. To obtain even better bounds, we executed the solver again with a time limit of 10 h and used for each instance the best available bound. The optimality gap evaluation is restricted to the instances in R_1 and R_4 because these are the only sets where sufficiently good bounds could be obtained within the time limit. Furthermore, we select the subset of instances whose optimality gap of the best MIP solution is below 10% because it can only be safely assumed that the dual bound is good if MIP's optimality gap is small. This step eliminates eight instances of R_1 (30%) and four of R_4 (40%), which means that 25 instances remain in the evaluation. By using the best bound, the optimality gap for each of the metaheuristic approaches is calculated on the selected instances.

Figure 13 shows the optimality gap for each instance in the reduced set for the greedy heuristic, simulated annealing, and MIP. For solutions which are not valid because of constraint violations, no mark is shown. The figure conveys that the solutions found by the greedy construction heuristic have gaps between 10% and 25%, and a considerable number of instances are not solved to feasibility at all. Simulated annealing always produced valid solutions which have a similar optimality gap as the MIP solutions, and in several cases even better. The gap is almost always below 5% and with an average of about 3%.

The resulting numbers state how large the relative difference between metaheuristic and optimal solutions is *at most* [4]. This result is interesting because it proves that our simulated annealing approach solves the majority of the small

[4] The gap is calculated using the best lower bound which was proven by the MIP solver. As most of the solutions were not proven optimal,

instances (which includes most of the realistic instances) extremely well. On average, the solutions are at most 3% above the optimal one.

Comparison on large randomly perfect instances $\mathbf{R_2}$: The instances in R_2 have been constructed in a way that the optimal solutions are known and have the objective value 0. This enables us to also draw some conclusions about the solution quality for larger instances than those for which we can obtain bounds using MIP. However, the optimality gap as defined in Eq. (28) does not shed light on the solution quality for instances in R_2. Indeed, since the best known lower bound is zero for all instances in R_2, the gap of non-optimal solutions (i.e., for solutions with objective value greater than zero) is equal to 1, irrespective of the actual value of the objective. Instead, we can assess the solution quality by taking a closer look at the components of the objective function. The first objective function component g_1, defined in Eq. (obj1′), states, informally speaking, the gap to a hypothetical perfectly leveled solution averaged over all periods. In case of the instances R_2, this hypothetical solution actually exists, and the value g_1 can be interpreted as the average percentage by which planned demand for each period exceeds or falls short of the target. The same argument holds for g_2, defined in Eq. (obj2′), which states the average percentage by which planned demand for each period exceeds or falls short of the target, averaged over the different product types and periods. The objective component g_3, defined in Eq. (obj3′), can be interpreted as the percentage of actual priority inversions measured against the theoretical maximum number of inversions $(k-1)\cdot k/2$.

Table 7 shows the values of g_1, g_2, and g_3 averaged over all instances in R_2 where a valid solution has been reached. The fourth column reports the percentage of valid solutions. We can see that simulated annealing reaches negligible mean deviations for the first two objectives. However, the greedy heuristic also produces leveling which deviates from the target by only 2% on average, so it can be assumed that this part of the task is not very hard for this instance set. With respect to the priority objective, the results leave more room for improvement. The greedy heuristic reaches a better value here than simulated annealing, but at the cost of fewer valid solutions. It is not entirely clear why the results on R_2 differ so much from the results which are obtained on the other parts of the test set. We suspect that it can be understood as follows: the greedy heuristic can find very good initial solutions on this instance set; however, some of this good structure is lost again by simulated annealing when it starts with a random search.

For the instance set R_3, simulated annealing finds feasible solutions for more than 65% of instances. Detailed results of simulated annealing on this instance set are given in Vass (2019).

Table 7 Results of metaheuristic methods on R_2. The values of the objective function components g_1, g_2, and g_3 multiplied by 100 so that they can be interpreted as percentages for each algorithm for each instance of R_2 where a valid solution has been reached. The rightmost column states the percentage of the solutions for which each algorithm reached a valid solution

	g_1(%)	g_2(%)	g_3(%)	valid (%)
Greedy	1.78	2.25	1.78	78.00
Sim. Ann. (median)	0.32	0.47	7.95	92.00
Optimum	0.00	0.00	0.00	100.00

To sum up, the above analysis of the optimality gap on small and realistic instances as well as the analysis of the solution cost for larger perfectly solvable instances revealed that our metaheuristic methods produce solutions which are only a few percentage points from the optimum. We can conclude that simulated annealing is well suited for solving the PLP in practice, because it both solves small instances almost perfectly and is able to handle large real-world instances.

7.5 Experimental evaluation of the PLP$^+$

In this section, we give an overview of our experiments conducted for the extended production leveling problem (PLP$^+$). The goal of these experiments is to verify whether our chosen approach for solving the PLP, i.e., MIP for small and simulated annealing for larger instances, is also suitable for solving the PLP$^+$.

Instance generation

To generate instances for the PLP$^+$, we extend the random instance generator proposed in 12 for the PLP with additional input parameters regarding splitting, resources, and due dates. First, an instance is created with a given a number of orders k, periods n, and product types m according to the process described in 12. This instance is extended using the following process:

1. Resource usage:
   (a) We randomly select a usage value between 0.0 and 1.0 for each product type. This usage value determines the resource usage per unit of demand for the particular product type.
   (b) 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 $(1+d_{\max})\cdot\bar{u}_r$.

the bounds are most probably smaller than the optimal objective value and thus the calculated gap an upper bound of the actual gap.

(c) For every order, 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.

2. Due dates:
   (a) Randomly choose the earliest start $pd_j^{\min}$ from $\{1..\lfloor 3/4 \cdot n \rfloor\}$ and
   (b) the latest end $pd_j^{\max}$ from $\{pd_j^{\min}+1, \ldots, n\}$ for each order.
3. Splitting of orders:
   (a) For each order j, randomly choose the maximum number of parts per order $pc_j^{\max}$ from $\{1..10\}$.
   (b) For each order j, randomly choose the minimum part size $ps_j^{\min}$ from $\{1..\lceil 1/2 \cdot d_j \rceil\}$[5]

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 $\{100..4000\}$, the number of periods n from $\{2..80\}$, the number of product types m from $\{1..20\}$ and the number of resources o from $\{1..5\}$. These instances form the set R_5.

Additionally, we generated another set denoted as R_6 of 20 smaller instances, using the following random parameters: The number of orders k is chosen from $\{10..100\}$, the number of periods n from $\{5..10\}$, the number of product types m from $\{1..3\}$ and the number of resources o from $\{0..3\}$.

Parameter Tuning As for the basic PLP, we used SMAC to tune 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 automatically tuned parameters set the weights for the move part and shift order neighborhoods to zero, which disables these two neighborhood operators. We therefore evaluated, in addition to the automatically tuned algorithm parameters (C_1), two manually selected parameter configurations 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 8 shows the details for the three parameter configurations evaluated in our experiments.

Computational results In a first series of experiments, we evaluated the performance of our methods on the instance set R_6 containing 20 small instances. In order to give the MIP approach sufficient time to prove optimal solutions, the time limit for all experiments was set to 1 h. 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 9 gives an overview of the experimental results for the set R_6. 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.

Detailed results of experiments with the small instances are visualized in Fig. 14. 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

Table 8 Overview of the algorithm parameter configurations used for experimental evaluation

Parameter	C_1	C_2	C_3
Initial temperature	6.2	0.01	0.1
Minimum temperature	0.00076	10^{-9}	10^{-9}
Iterations per temperature	154,788	300, 000	300, 000
Move part weight	0	4	2.5
Swap parts weight	7	2	2.5
Split order weight	3	3	2.5
Shift order weight	0	1	2.5
Cooling rate	0.95	0.95	0.95

Table 9 Summarized results of the experiments on the test set R_6. 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

	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

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

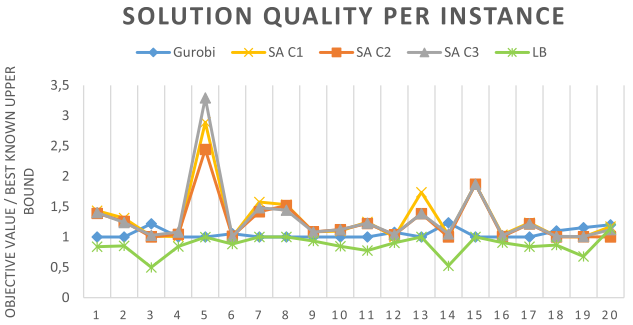

Fig. 14 A visualization of the experimental results for the 20 small instances in the set R_6. The horizontal axis represents the 20 evaluated instances, whereas the vertical axis measures the achieved relative objective values (solution cost produced by each method divided by the overall best found solution cost) (Color figure online)

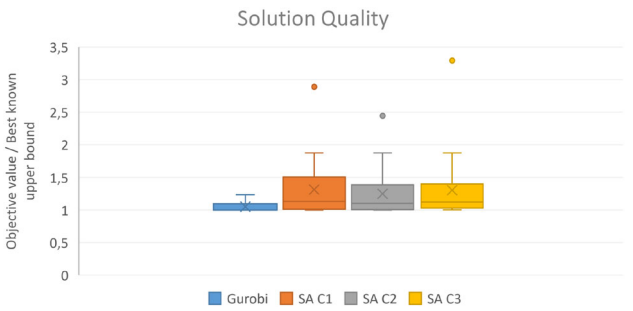

Fig. 15 Box plots comparing the overall results achieved on the set R_6. The vertical axis measures the achieved relative objective values (solution cost produced by each method divided by the overall best found solution cost) (Color figure online)

(results with a lower bound value of 1 denote proven optimal solutions).

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 15 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 $C1$.

In a second series of experiments, we evaluated the performance of the proposed methods on the instance set R_5 which contains 986 large randomly generated instances. Similarly 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 5-min time limit for the set of larger instances.

Table 10 Summarized results of the experiments on the set R_5. 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

	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

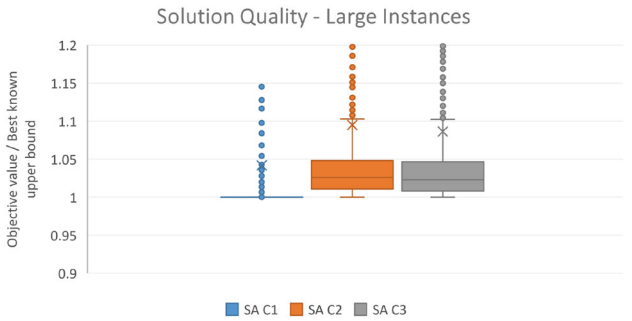

Fig. 16 Box plots comparing the overall results achieved with the simulated annealing approach on the set R_5. The vertical axis measures the achieved relative objective values (solution cost produced by each method divided by the overall best found solution cost). Note that some outliers have been excluded for an improved visual comparison (Color figure online)

The results of the experiments on the set R_5 are summarized in Table 10. Similarly as in Table 9, 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.

The results show that no approach is able to produce feasible solutions for all 986 instances within the time limit[6]. 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 a slightly larger number of feasible solutions than $C1$. However, most best solutions were produced using parameter configuration $C1$. No optimality proofs could be achieved within the given time limits.

Figure 16 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,

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

the automatically tuned algorithm configuration C_1 produces the best results in our experiments, whereas configurations C_2 and C_3 both produce solutions of slightly lower quality.

In summary, our experiments show that the exact approach obtains the best results for most of the small instances (set R_6). However, in experiments with the larger instances (set R_5), the integer programming solver turned out not to be 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, which also explains why this configuration was selected by automated parameter tuning as it produces the overall best configuration for all benchmark 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 R_6 in our experiments.

8 Conclusion

We introduced a new real-life combinatorial optimization problem in the area of production planning, which concerns the assignment of orders to production periods. It involves several production capacity constraints, a work balancing objective, and an order prioritization objective that must be optimized.

The main findings of this work are:

- The PLP is $\mathcal{NP}$-hard. A tractable case, the fixed-order PLP, is identified, and a polynomial-time algorithm presented.
- The proposed MIP model solves instances up to medium size. The complexity of solving grows with every dimension of the problem, but most notably with the number of orders k. For instances with fewer than 250 orders, either a feasible solution or the proof of infeasibility can be expected within 1 h of time, while we cannot count on finding any solution for instances with $k \geq 300$.
- With simulated annealing, very good solutions can be obtained within 5 min. The real-life instances can all be solved well, and for most of them we can show through the use of dual bounds that the solutions are within 3% of optimality on average. Experiments based on the set of instances with perfectly leveled solutions indicate that simulated annealing is also capable of providing very good solutions for much larger instances.
- An experiment regarding the weighting of the two neighborhoods in simulated annealing showed that it is clearly advantageous to use both neighborhoods instead of either of them alone.
- We introduced the PLP$^+$, an extension of the PLP which allows us to tackle order splitting, due dates, and resources. Via a series of experiments on randomly generated instances, similar results as for the PLP were shown: our MIP formulation allows us to find optimal solutions for small instances with up to 100 orders. For larger instances of a realistic size, an extension of our simulated annealing algorithm with four new neighborhoods finds feasible solutions for most of the instances within a reasonable time limit. Based on the configuration provided by the automated parameter tuner SMAC, we can conclude that the most important neighborhoods are the swap parts and split moves.

To sum up, this paper introduces a new planning problem that arises in the field of industrial production and presents methods that enable very good solutions to be found for real-life instances. Building upon these results, several directions for future work present themselves: First, it would be interesting to investigate other metaheuristic approaches such as tabu search to escape local optima. Second, one could investigate an approach that hybridizes the proposed exact and metaheuristic techniques within the framework of large neighborhood search. Moreover, it would be interesting to investigate how the dynamic programming algorithm presented for the fixed-order PLP can be exploited to improve solution methods for the PLP. For instance, it could be used to produce initial solutions for the PLP.

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.

Funding Open access funding provided by TU Wien (TUW).

References

Alon, N., Azar, Y., Woeginger, G. J., & Yadid, T. (1998). Approximation schemes for scheduling on parallel machines. *Journal of Scheduling, 1*(1), 55–66.

Boysen, N., & Fliedner, M. (2011). Scheduling aircraft landings to balance workload of ground staff. *Computers & Industrial Engineering, 60*(2), 206–217.

Boysen, N., Fliedner, M., & Scholl, A. (2007). A classification of assembly line balancing problems. *European Journal of Operational Research, 183*(2), 674–693.

Boysen, N., Fliedner, M., & Scholl, A. (2009). The product rate variation problem and its relevance in real world mixed-model assembly lines. *European Journal of Operational Research, 197*(2), 818–824.

Castro, C., & Manzano, S. (2001). Variable and value ordering when solving balanced academic curriculum problems. In *Proceedings of 6th Workshop of the ERCIM WG on Constraints.*.

Chiarandini, M., Di Gaspero, L., Gualandi, S., & Schaerf, A. (2012). The balanced academic curriculum problem revisited. *Journal of Heuristics, 18*(1), 119–148.

Coleman, B. J., & Vaghefi, M. R. (1994). Heijunka (?): A key to the toyota production system. *Production and Inventory Management Journal, 35*(4), 31.

Deb, K. (2014). Multi-objective optimization. In *Search methodologies*, (pp. 403–449), Springer .

Di Gaspero, L., & Schaerf, A. (2008). 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, and M. Sampels, (Eds.), *Hybrid Metaheuristics, 5th International Workshop, HM 2008, Málaga, Spain, October 8-9, 2008. Proceedings*, volume 5296 of *Lecture Notes in Computer Science*, pp. 146–157. Springer .

Flajolet, P., & Sedgewick, R. (2009). *Analytic combinatorics*. Cambridge University Press.

Garey, M. R., & Johnson, D. S. (1979). *Computers and intractability* (Vol. 174). W. H.

Gurobi Optimization, L. (2019). Gurobi optimizer reference manual . URL http://www.gurobi.com.

Kirkpatrick, S., Gelatt, C. D., & Vecchi, M. P. (1983). Optimization by simulated annealing. *Science, 220*(4598), 671–680.

Kubiak, W. (1993). Minimizing variation of production rates in just-in-time systems: A survey. *European Journal of Operational Research, 66*(3), 259–271.

Kubiak, W., & Yavuz, M. (2008). Just-in-time smoothing through batching. *Manufacturing & Service Operations Management, 10*(3), 506–518.

Lackner, M.-L., Vass, J., & Musliu, N. (2019). *Extended Complexity Results for the Production Leveling Problem*. Technical Report CD-TR 2019/2.

Lindauer, M., Eggensperger, K., Feurer, M., Falkner, S., Biedenkapp, A., & Hutter, F. (2019) SMAC v3: Algorithm configuration in python. URL https://github.com/automl/SMAC3.

Miettinen, K. (2012). *Nonlinear multiobjective optimization* (Vol. 12). Springer Science & Business Media.

Miltenburg, J. (1989). Level schedules for mixed-model assembly lines in just-in-time production systems. *Management Science, 35*(2), 192–207.

Mullinax, C., & Lawley, M. (2002). Assigning patients to nurses in neonatal intensive care. *Journal of the Operational Research Society, 53*(1), 25–35.

Ohno, T., & Rosen, C. B. (1998). *Toyota production system: Beyond large-scale production*. Productivity Press.

Punnakitikashem, P., Rosenberber, J. M., & Buckley-Behan, D. F. (2013). A stochastic programming approach for integrated nurse staffing and assignment. *IIE Transactions, 45*(10), 1059–1076.

Schaus, P. (2009). *Solving balancing and bin-packing problems with constraint programming*. PhD thesis, UCL - Université Catholique de Louvain. URL https://dial.uclouvain.be/pr/boreal/en/object/boreal%3A23871.

Schaus, P., Hentenryck, P. V., & Régin, J.-C. (May 2009). Scalable load balancing in nurse to patient assignment problems. In W. J. v. Hoeve and J. N. Hooker (Eds.) *Integration of AI and OR Techniques in Constraint rogramming for Combinatorial Optimization Problems, 6th International Conference, CPAIOR 2009, Pittsburgh, PA, USA, May 27-31, 2009, Proceedings*, volume 5547 of *Lecture Notes in Computer Science*, pp. 248–262. Springer .

Schreiber, E. L. (2014). *Optimal multi-way number partitioning*. PhD thesis, University of California.

Schwerdfeger, S., & Walter, R. (2016). A fast and effective subset sum based improvement procedure for workload balancing on identical parallel machines. *Computers & Operations Research, 73*, 84–91.

Skiena, S. . S. (1998). *The algorithm design manual*. Springer Science & Business Media.

Vass, J. (2019). *Exact and metaheuristic approaches for the production leveling problem*. Master's thesis, TU Wien, https://repositum.tuwien.at/handle/20.500.12708/6542.

Vass, J., Musliu, N., & Winter, F. (2020). Solving the production leveling problem with order-splitting and resource constraints. In *Proceedings of the 13th International Conference on the Practice and Theory of Automated Timetabling - PATAT 2021: Volume I*, pp. 261–284. PATAT .

Vazirani, V. V. (2003). *Approximation algorithms*. Springer-Verlag.

Warner, D. M. (1976). Scheduling nursing personnel according to nursing preference: A mathematical programming approach. *Operations Research, 24*(5), 842–856.

Yavuz, M., & Tufekci, S. (2006). A bounded dynamic programming solution to the batching problem in mixed-model just-in-time manufacturing systems. *International Journal of Production Economics, 103*(2), 841–862.

Yavuz, M., & Tufekci, S. (2006). Dynamic programming solution to the batching problem in just-in-time flow-shops. *Computers & Industrial Engineering, 51*(3), 416–432.