Extended Complexity Results for the Production Leveling Problem

Marie-Louise Lackner, Johannes Vass and Nysret Musliu

Christian Doppler Laboratory for Artificial Intelligence and
Optimization for Planning and Scheduling
TU Wien
Favoritenstraße 9-11, 1040 Vienna, Austria

{mlackne1,jvass,musliu}@dbai.tuwien.ac.at

September 27, 2019

CD-TR 2019/2

EXTENDED COMPLEXITY RESULTS FOR THE PRODUCTION LEVELING PROBLEM *

A PREPRINT

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

Johannes Vass
jvass@dbai.tuwien.ac.at

Nysret Musliu
musliu@dbai.tuwien.ac.at

Christian Doppler Laboratory for Artificial Intelligence and Optimization for Planning and Scheduling
Database and Artificial Intelligence Group
TU Wien, 1040 Wien, Austria

September 27, 2019

ABSTRACT

This work takes up the recently introduced Production Leveling Problem and provides extended complexity results. The problem is a multi-objective constrained combinatorial optimization problem that aims assign orders to production periods such that the load between production periods is balanced while at the same time minimizing the number of priority inversions. As the problem is NP-hard, there is currently no algorithm known which could solve arbitrary instances in polynomial time.

We contribute a new variant of the problem called the Fixed-Order Production Leveling Problem which works on a restricted set of instances and has the priorities as hard constraints. We show that these adaptions make the problem tractable by proposing a dynamic programming algorithm and analyzing its complexity.

***K*eywords** Production Leveling · Dynamic Programming

1 Introduction

The Production Leveling Problem is a multi-objective constrained combinatorial optimization problem which aims to distribute a set of orders to production periods. The goal is to achieve a balanced production volume throughout the planning horizon, achieve a constant product mix and take the order priorities into account. It serves as a pre-processing to a subsequent scheduling step, which is executed for each production period separately to determine the concrete production sequence and resource assignment.

The Production Leveling Problem was introduced by Vass et al. 2019 [2], where it was proven to be NP-hard. In this paper we aim to analyze special cases and variants of the problem in order to find tractable cases.

This technical report presents preliminary follow-up work of the original paper. The main contribution is the introduction the Fixed-Order-PLP which is a tractable variant of the Production Leveling Problem (PLP). We show that it is solvable in polynomial time by proposing an algorithm based on dynamic programming.

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

2 The Production Leveling Problem

In the following we state again the mathematical problem formulation of the Production Leveling Problem as introduced by Vass et al. [2].

Input parameters

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

Variables

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

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

- For each period the sum of all planned order's demands (helper variable):

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

- For each period and product the sum of all planned order's demands (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:

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

Objective Function

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

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

$$f_2 = \sum_{t \in M} \left(\frac{1}{d_t^*} \cdot \sum_{i \in N} |d_t^* - w_{i,t}| \right) \tag{2}$$

$$f_3 = \left| \left\{ (i, j) \in K^2 : y_i > y_j \text{ and } p_i > p_j \right\} \right| \tag{3}$$

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 target 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.

In order to combine these three objectives into a single objective function and achieve a weighting which does not change its behavior between instances with different number of orders, periods or product types, the cost components need to be normalized.

$$g_1 = \frac{1}{n \cdot d^*} \cdot f_1 \tag{4}$$

$$g_2 = \frac{1}{n \cdot m} \cdot f_2 \tag{5}$$

$$g_3 = \frac{2}{k \cdot (k-1)} \cdot f_3 \tag{6}$$

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

The final objective function is then a weighted sum of the three normalized objective functions, where the weight a_i of an objective can be seen approximately as its relative importance.

$$\textbf{minimize } g = a_1 \cdot g_1 \ + \ a_2 \cdot g_2 \ + \ a_3 \cdot g_3 \tag{7}$$

3 Fixed-Order Production Leveling Problem

In this section we consider 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.

FIXED-ORDER PRODUCTION LEVELING (OPTIMIZATION PROBLEM)	
Instance:	Set of orders $K = \{i \in \mathbb{Z}^+ \mid 1 \leq i \leq k\}$ Set of products $M = \{i \in \mathbb{Z}^+ \mid 1 \leq i \leq m\}$ Set of periods $N = \{i \in \mathbb{Z}^+ \mid 1 \leq i \leq n\}$ For each order $j \in K$ its demand $d_j > 0$, priority p_j (unique among all orders) and product type t_j Maximum production capacity c Maximum production capacity c_t for each product type $t \in M$ Target production capacity: $d^* \in \mathbb{Z}^+$ Target production capacity for product type $t \in M$: $d_t^* \in \mathbb{Z}^+$
Objective:	Find a mapping of orders to periods $y : K \to N$ that minimizes $g = a_1 \cdot g_1 + a_2 \cdot g_2$ (like equation 7 but without the priority objective g_3)
Constraints:	Respect the priorities: $y(i) \leq y(j)$ for orders $i, j \in K$ with $p_i > p_j$

The only difference to 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 allows us to view the construction process of the solution as partitioning the sequence of orders sorted by priority instead of assigning arbitrary subsets to each period, which also explains the name Fixed-Order Production Leveling Problem. The view as an assignment problem would be unnecessarily complex for this problem variant because the new hard constraint assures that no priority inversion can exist and the assumption of unique priority values removes the ambiguity when sorting the orders by priority.

The view as a partition problem reminds strongly of the list partition problem, as described for example in *The Algorithm Design Manual* [1]. This problem is defined as follows: Given a sequence S of length k of non-negative numbers $s_1, s_2, \ldots, s_k$ and an integer n, find a partition of S into n ranges, i.e. consecutive elements of the sequence, so as to minimize the maximum sum over all the ranges. This problem allows for efficient solutions using dynamic programming.

As our problem can be seen as an extension of the list partition problem with a different objective we investigate this solution approach further. The algorithm used for the list partition problem cannot be used directly in our setting, since it is not sufficient that the maximum planned capacity of all period is as small as possible and thus as close to d^* as possible; we require that orders are divided evenly on periods for all other periods as well. However, the algorithm described in the following is heavily inspired by the one presented for list partition in [1].

The main idea of the algorithm is that an optimal partitioning of a sequence according to some objective[2] can be calculated step by step. In each step of the procedure, we extend some prior solution by one more partition until we reach the desired number. The new partition corresponds always to the last period, so it contains always the l last orders in the sorted order list, which are the ones with lowest priority. In order to decide how large the partition should be (i.e. how many orders from the end of the order list it should contain) the cost for all possibilities is calculated. All possibilities where we can have at least one order in each period are considered. For example, assume that we want to compute the optimal total cost of assigning the last l elements of the sequence to the new partition. Intuitively, it consists of the optimal total cost of assigning the first $k - l$ orders (where k is the total number orders) to the previous number of partitions plus the cost of assigning the last l elements to a new partition. The first part is already known from the previous step of the algorithm, while the second part can be easily calculated. After having computed the cost for all possibilities we define the optimal total cost for this subproblem as the minimum of the encountered cost values.

Theorem 1. *The Fixed-Order Production Leveling problem can be solved in polynomial time using a dynamic programming approach: If n denotes the number of periods and k the number of orders, it can be solved in $\mathcal{O}(nk^2)$ time.*

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 capacity demands of 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 is the capacity demand of the order with priority $k - 1$, aso. until d_k which is the capacity demand of the order with priority 1. This sorting requires $\mathcal{O}(k \log k)$ time.

Furthermore we assume that $k \geq n$ which is sensible as otherwise some periods would be forced to stay empty and we could simply reduce the number of periods. With this assumption we can safely exclude all cases from the algorithm, where some period stays empty because compared to planning two orders for one period and leaving one empty it is always at least as good to plan the two orders in different periods.

In the following, we describe how the Fixed-Order Production leveling optimization problem can be solved using a dynamic programming approach. For this purpose, 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 periods. Thereby it is important to point out that the normalization factors inside the objective function are always those of the original problem, regardless of the choice of j and l. The optimal objective value of the original problem is clearly given by $O(k, n)$ because k and n are the original number of orders and periods.

As indicated in the intuitive explanation of the algorithm above, $O(j, l)$ for $l \leq j \leq k$ and $1 \leq l \leq n$ can be calculated recursively by selecting the variant with minimum cost out of the following two extreme cases and all cases in between:

- assigning only one order to the new period and $j - 1$ to the $l - 1$ periods before
- assigning $j - l$ orders to the new period and $l - 1$ to the $l - 1$ periods before.

The table entries $O(j, l)$ with $j < l$ are not defined because we do not consider cases whose optimal solution involves empty periods. Formally, the recursion for calculating the values $O(j, l)$ is given by

$$O(j,l) = \min_{l \leq i \leq j} \left(O(i-1, l-1) + h_1(i,j) + h_2(i,j) \right) + constr(i,j) \tag{8}$$

The functions $h_1(i, j)$ and $h_2(i, j)$ denote the respective cost increase of f_1 and f_2, if we would assign the set of orders $\{i, \ldots, j\}$ to a new and empty period. Formally they are defined as follows for $1 \leq i \leq j + 1$:

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

$$h_2(i,j) = \frac{a_2}{n \cdot m} \cdot \sum_{t \in M} \left| \frac{d_t^* - \sum_{s=i|t_s=t}^{j} d_s}{d_t^*} \right|$$

The penalty function $constr(i, j)$ checks whether the capacity restrictions c and c_t allow for assigning the set of orders $\{i, \ldots, j\}$ to the same period and returns 0 if so and ∞ otherwise. If the final value $O(k, n)$ happens to be ∞ we

[2]The objective must have the property that the cost of each partition is independent from the cost of other partitions. This condition on the objective is clearly met in the case of the PLP, as each period (which corresponds to a partition) contributes independently to the total cost.

immediately know that the instance is infeasible. The reason why this approach works is that both constraints can be checked for each period separately.

The calculation of $O(j, l)$ results indeed in the optimal objective value of the subproblem where the first j orders are assigned to l periods because

- all open possibilities are enumerated (due to the fixed ordering and the requirement of having at least one order in each period their number is only $j - l$), and
- the previously computed values $O(i - 1, l - 1)$ stay meaningful also when adding another period because the associated costs of the periods are independent.

The base cases of the recursion are those, where between 1 and k orders are assigned to one period, which is trivially given by :

$$O(j, 1) = h_1(1, j) + h_2(1, j) + \mathit{constr}(1, j) \qquad \text{for } 1 \leq j \leq k.$$

In order 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). In order 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 $h_1(i, j)$ and $h_2(i, j)$ for every $1 \leq i \leq j$. Since these are also required for further elements of the table, these values $h_1(i, j)$ and $h_2(i, j)$ are pre-computed for all i, j with $1 \leq i \leq j \leq k$, which requires $\mathcal{O}(k^2)$ time.

Given these pre-computations, the time needed to compute each entry $O(j, l)$ is in $\mathcal{O}(k)$ because the minimum of $j + 1 \leq k$ values needs to be found, each of which requires only access to 3 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 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 this minimum was achieved in Equation (8)[3]. This is stored in the array $M = M(j, l)_{1 \leq j \leq k, 1 \leq l \leq n}$ with

$$M(j, l) = i \iff O(j, l) = O(i - 1, l - 1) + f_{i,j} + h_{i,j}.$$

Computing $M(j, l)$ in addition to $O(j, l)$ adds only a constant amount to the computational complexity, so that the asymptotic behaviour 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. When we assess the asymptotic complexity of the whole dynamic programming algorithm, the computation of the $O(j, l)$ values with complexity $\mathcal{O}(n \cdot k^2)$ clearly outweighs all other parts. Therefore the total complexity is also $\mathcal{O}(n \cdot k^2)$. □

Example 1. *As an example, consider the following instance with $k = 20$ orders:*

type	*gray*	*red*	*blue*	*green*
order =	*(20,10) (15,10)*	*(18,5) (14,20)*	*(19,5) (13,10)*	*(17,5) (16,20)*
(priority, demand)	*(10,10) (7,15)*	*(9,10) (8,5)*	*(6,15) (3,5)*	*(12,25) (11,20)*
	(4,5)			*(5,30) (2,5) (1,5)*

[3] If this value of i is not unique, we pick the smallest such i.

These orders need to be scheduled to $n = 5$ periods and we have $d^* = \frac{1}{n}\sum_{s=1}^{k} o_s = 47$, $d^*_{gray} = 10$, $d^*_{red} = 8$, $d^*_{blue} = 7$, $d^*_{green} = 22$. We choose the weights of objectives as follows: $a_1 = 1$ and $a_2 = 1$. The sorted list of orders is given as follows:

$$(o_1, \ldots, o_{20}) = (10, 5, 5, 5, 20, 10, 20, 10, 25, 20, 10, 10, 5, 15, 15, 30, 5, 5, 5, 5).$$

The values for $O(j, l)$ and $M(j, l)$ with $0 \leq j \leq 20$ and $1 \leq l \leq 5$ are given in the table below:

j \ l	1	2	3	4	5
1	0.31	–	–	–	–
2	0.25	0.65	–	–	–
3	0.20	0.60	1.00	–	–
4	0.17	0.57	0.97	1.37	–
5	0.05	0.43	0.83	1.23	1.63
6	0.12	0.34	0.74	1.14	1.54
7	∞	0.30	0.68	1.08	1.48
8	∞	0.22	0.61	1.01	1.41
9	∞	0.31	0.48	0.86	1.26
10	∞	∞	0.58	0.74	1.13
11	∞	∞	0.49	0.65	1.04
12	∞	∞	0.41	0.57	0.96
13	∞	∞	0.42	0.58	0.92
14	∞	∞	∞	0.67	0.83
15	∞	∞	∞	0.61	0.77
16	∞	∞	∞	0.62	0.79
17	∞	∞	∞	∞	0.80
18	∞	∞	∞	∞	0.75
19	∞	∞	∞	∞	0.89
20	∞	∞	∞	∞	**0.86**

j \ l	1	2	3	4	5
1	1	–	–	–	–
2	1	2	–	–	–
3	1	2	3	–	–
4	1	2	3	4	–
5	1	5	5	5	5
6	**1**	5	5	5	6
7	∞	6	6	7	6
8	∞	6	6	7	8
9	∞	**7**	8	8	9
10	∞	∞	10	10	10
11	∞	∞	10	10	10
12	∞	∞	10	10	10
13	∞	∞	**10**	10	13
14	∞	∞	∞	13	13
15	∞	∞	∞	13	13
16	∞	∞	∞	**14**	14
17	∞	∞	∞	∞	16
18	∞	∞	∞	∞	16
19	∞	∞	∞	∞	17
20	∞	∞	∞	∞	**17**

The optimal objective value is given by $O(20, 5) \approx 0.86$. The entries showing a '–' are those where $j < l$, which are unnecessary to compute, as stated above. In all cases where $O(j, l) = \infty$ there does not exist a feasible solution to the subproblem.

The $M(j, l)$ values printed in bold face visualize the path through the table which allows us to reconstruct the optimal solution. The reconstruction works as follows:

- $M(20, 5) = 17$*, thus the orders assigned to period 5 are:* $o_{17}, \ldots, o_{20}$.
- $M(16, 4) = 14$*, thus the orders assigned to period 4 are:* $o_{14}, \ldots, o_{16}$.
- $M(13, 3) = 10$*, thus the orders assigned to period 3 are:* $o_{10}, \ldots, o_{13}$.
- $M(9, 2) = 7$*, thus the orders assigned to period 2 are:* $o_7, \ldots, o_9$.
- $M(7, 1) = 1$*, thus the orders assigned to period 1 are:* $o_1, \ldots, o_6$.

Figure 1 visualizes the obtained solution in two different ways. The view on the left shows the overall production volume assigned to each period, where the dashed line is the target value d^*. The sum of absolute differences between the bar and the dashed line corresponds to f_1. The view on the right shows the production volume per period, and the sum of absolute differences corresponds to f_2. The bold colored lines are the maximum capacities c and c_t, which are obviously not exceeded in this solution.

4 Conclusion

We introduced the Fixed-Order Production Leveling Problem, which is a variant of the PLP as reported in [2]. This difference lies in that the priorities are treated as a hard constraint instead of a soft constraint, which means that no priority inversion is allowed at all for a solution to be feasible. We pointed out the similarities to the PARTITION problem and claimed, that it is also solvable in polynomial time. As a proof, we introduced a dynamic programming algorithm which can solve the Fixed-Order Production Leveling Problem in $O(n \cdot k^2)$ time.

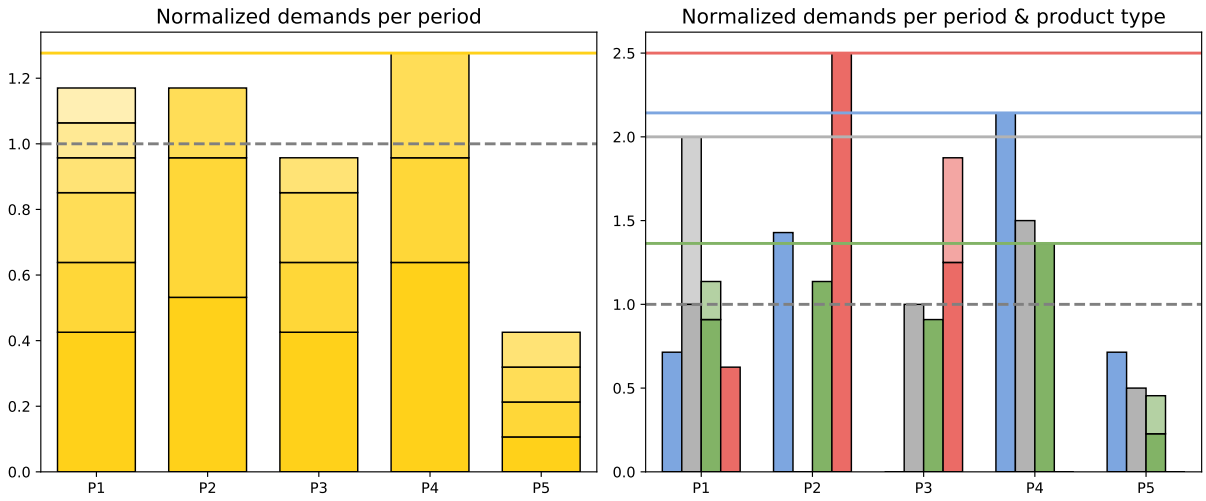

Figure 1: Optimal solution of the presented example instance. To the left a visualization of objective 1 and on the right a visualization of objective 2.

References

[1] Steven S. Skiena. *The Algorithm Design Manual*. Springer Science & Business Media, 1998.

[2] Johannes Vass, Marie-Louise Lackner, and Nysret Musliu. Exact and Metaheuristic Approaches for the Production Leveling Problem. *submitted to Computers & Operations Research*, September 2019. preprint available at `https://dbai.tuwien.ac.at/staff/jvass/publications/plp_paper_preprint.pdf`.