Master Thesis in Econometrics and Management Science:
Operations Research & Quantitative Logistics

Optimising the dynamic uncapacitated lot-sizing problem with inventory bounds and co-production units

Name student: Stephan Kroon
Student ID number: 455100

Supervisor: dr. W. van den Heuvel
Second assessor: dr. S. Ağrali

Date final version: 30th of April 2021

Abstract

In this thesis, we solve the uncapacitated lot-sizing problem with inventory bounds and co-production units. This problem has not been solved by literature and is therefore very interesting. We prove that this problem is strongly $\mathcal{NP}$-hard, and that the zero-inventory property can not be applied. Therefore, a mathematical programming heuristic based on a Dantzig-Wolfe formulation is a justified suggestion. This does not result in finding the optimal solution, by cause of binary setup costs variables. That is why we come up with a heuristic inspired by relax and fix & fix and optimise. Besides, we apply some iterations of a branch-and-price algorithm. After that, we make changes to this decomposition to investigate if that would improve the obtained results. Unfortunately, that does not improve the solutions. A more advanced Dantzig-Wolfe decomposition, in which producing multiple co-products in every iterations, is suggested. This has not, in contrast to the other methods, been implemented. The proposed heuristic does find good solutions at a fast pace, and is therefore applicable in real-world instances. In one instance, a lower objective value has been found using the heuristic.

Preface

After finishing all Master's courses at the Erasmus School of Economics, I was able to start writing this Master's thesis. Writing this, is the last step in obtaining my Master's degree. Writing a thesis while being in lockdown for over a year is hard. Imaging hanging out with friends without any limitations, going to a bar and have a wonderful night, while ending at the McDonalds Coolsingel for an amazing afterparty with snacks. Those things are amazing in motivating you to work hard. It also makes studying great! I had so many years in which I started the day studying with friends, drinking some coffee with them, attending some lectures, working together for group assignments and then going out with them in the evening. All of this is actually forbidden while writing this. We are only allowed to drink with at most one friend at a time. Unfortunately, a curfew of at least 3 months made it much harder to meet with friends. Luckily, sleepovers helped through this.

The social contact is limited while writing this, which made every meeting with my supervisor, Wilco van den Heuvel, a relief. I was finally able to speak with someone outside my roommates. Unfortunately, all contact was online, staring at a screen. The feedback from my supervisor was great, and it definitely helped me. Thanks for helping me! I would also like to thank Semra Ağrali for providing me a data set and for being the second assessor. I would also like to thank my friends and family for supporting me, while also having drinks with them. It definitely helped!

The last one I would like to thank is you, the reader, for taking time to read this thesis! Enjoy reading it!

Stephan Kroon

Contents

1 Introduction

In production systems, it is reasonable to assume that multiple products are always produced together. In case that all products have the same importance, it is called co-production. It is also possible that one product is the "selling product" and the others are less important for any kind of reason. Then, those less important products are called by-products.
We provide some examples of co-production. The first example is about glass manufactoring (Öner and Bilgiç (2008)). The quality of glass (calculated as imperfections per unit area) is best at the beginning of production and becomes worse after producing for a while. Therefore, large size glass products are produced at the beginning of the process. As the quality becomes worse, smaller size glass is produced instead. An automated system identifies the imperfections. Then, multiple "different" products are produced. Different meaning that the product glass is the same, but the size of the glass is different, with each glass size having its own market. Another example of co-production is the chicken industry (Lu and Qi (2011)). After a chicken has been killed, different parts of that chickens are sold: chicken wings, breast, and so on. Those products are separated from each other. All of those sold parts are in every chicken, with all parts having a fixed ratio. A last example is in the car industry (Ağralı (2012)). Cars have symmetric parts, and therefore most car production processes are made to get both parts at once (co-producing). Damaged parts, for instance, needs to be replaced, but that does not mean that all symmetric parts needs to be replaced. Then, the demand for each part could be different.

Co-producing differs from normal production since there are always multiple products produced. The relation between the produced products when co-producing is fixed and known. In case co-production is overestimated, it could be the case that one produced too many products. This could lead to unnecessary holding costs or to a full inventory. On the other side, if co-production is considered as production with by-products, demand of those by-products might not be met. In current literature, a lot of research has been done about lot-sizing problems, but most of this research do not include co-production. And if co-production is taken into account (Ağralı (2012)), co-production units (CPUs) are not considered. Therefore, the problem becomes very interesting.

When solving the dynamic uncapacitated lot-sizing problem with inventory bounds and co-production units, we assume that all demand is known and dynamic. This means that there is no uncertainty in the requested demand and that the demand is different in each period. Furthermore, we assume that production is controlled and deliberated. This means that we can produce as much as needed, without worrying about a possible machine failure. Next, all demand must be met on the right time and therefore, no backorders are allowed and it is not possible to satisfy future demand earlier. Finally, the inventory is limited. Products that are produced in a certain period are put in the inventory after all demand is satisfied in that same period. In this thesis, we try developing heuristics which are able to get good optimality gaps within reasonable time.

This thesis has the following structure. In Section 2 we give a literature review containing existing literature about, for instance, lot sizing problems. After that, we give a formal problem description about the problem we are solving in Section 3. This also contains a MBLP formulation. Then, we show that this problem is indeed hard to solve in Section 4. We do this by proving that this problem is strongly $\mathcal{NP}$-hard. We also show that the optimal solution does, in general, not satisfy the zero-inventory property. Then, we present our proposed methods to solve this problem in Sections 5 and 6. The first one is based on a Dantzig-Wolfe decomposition. This solution does not satisfy all constraints, therefore we come up with a math programming heuristic instead. We also give the outline of a Branch-and-Price algorithm. The latter is not a Dantzig-Wolfe decomposition, but it is inspired from it. Next, we provide another Dantzig-Wolfe decomposition. Unfortunately, the methods are not sufficient to solve the pricing problems. Following, we present the most important results in Section 8 in combination with the obtained data. All other results can be found in the Appendix. We end this thesis with a conclusion in Section 9.

2 Literature review

The dynamic uncapacitated single-item economic lot sizing problem, without extentions, has been introduced and solved exactly using dynamic programming by Wagner and Whitin (1958). The dynamic programming algorithm runs in $\mathcal{O}(T^2)$. The model uses the zero-inventory property. That means that it is only allowed to produce when the inventory is zero. The complexity of the algorithm is reduced to $\mathcal{O}(T \log T)$ in Wagelmans et al. (1992) via adjusting the production costs and removing the holding costs. Extensions to the this lot-sizing problem have been studied. In this literature review, we split literature into three different classes: single-item lot-sizing (Section 2.1), multi-item lot-sizing (Section 2.2) and lot-sizing with co-production (Section 2.3).

2.1 Single-item lot-sizing problems

The lot sizing model has inspired others to come up with extensions. Brahimi et al. (2017) made an overview of existing literature about single-item dynamic lot sizing problems. In the paper, dynamic programming is a useful solution method. This problem can be extended with capacity constraints. There are two types of capacity constraints: production capacity and inventory capacity constraints. Branch-and-bound is used when dealing with production capacity constraints. For this particular problem, Lotfi and Yoon (1994) developed a branch-and-bound algorithm in which concave production and holding costs are assumed. The method performs well in practical cases. Branch-and-cut is more popular for lot sizing problems. Loparic et al. (2003) came up with valid inequalities which are based on the dynamic knapsack problem. It turns out that the valid inequalities can be effective in solving lot sizing problems.

Adding inventory constraints is a relevant extension to the problem we solve. Love (1973) was the first author to come up with an uncapacitated lot sizing problem including capacity limits. The problem is solved in $\mathcal{O}(T^3)$. Gutiérrez et al. (2003) developed an algorithm with the same complexity, but computational results show that it is solved almost 30 times faster. Toczylowski (1995) developed an algorithm which runs in $\mathcal{O}(T^2)$ time.

2.2 Multi-item lot-sizing problems

Another extension to the original ELS problem is adding multiple items which demand needs to be satisfied. These items are independently of each other produced. Melo and Ribeiro (2017) made an overview of multi-item uncapacited lot-sizing problem with inventory bounds, in which different reformulations are made. The first one being discussed is based on Lange (2010) by reformulating the problem into a facility location formulation. Using the linear relaxation, the optimality gap got reduced. Pimentel et al. (2010) solved the multi-item capacitated lot sizing problem via Dantzig-Wolfe decomposition and branch-and-price. In the Dantzig-Wolfe decomposition, two different decompositions were used: product decomposition and period decomposition. In the product decomposition, the capacity constraints were left in the restricted master problem. Every individual product got its own subproblem: single-item uncapacitated lot sizing problem. In the period decomposition the demand constraints were left in the restricted master problem. Every period got its own subproblem: a continuous knapsack problem with setup costs. This is solved using the branch-and-bound algorithm of Michel et al. (2009). Branch-and-price is used to ensure that all constraints, of the original problem, are met. Eppen and Martin (1987) solve the multi-item lot sizing problem via modeling it as a shortest path problem. The models proved that these models have their LP relaxation equal to the lagrangian dual. Pochet and Wolsey (2006) explained more about shortest path formulations to solve various lot-sizing problem. To solve shortest path problems with resource constraints, Desrochers (1988) developed a labeling algorithm in combination with dominance rules. These dominance rules speed up the algorithm. The last exact approach, Melo and Ribeiro (2017), is using valid inequalities. Barany et al. (1984) introduced (l, S)-inequalities. The inequalities imply that the demand is satisfied by the available stock in inventory or by producing. Since there are a lot of those inequalities, it is used in combination with a cutting-plane algorithm. Pochet and Wolsey (1994) used a subset of the

(l, S)-inequalities. Those are called the Wagner-Whitin (l, S)-inequalities. There are a polynomial number of Wagner-Whitin (l, S)-inequalities available.

For the multi-item lot-sizing problem with inventory bounds, Akbalik et al. (2015) proved that this problem is strongly $\mathcal{NP}$-hard, even if the problem is reduced to a problem without setup and holding costs. If non-speculative costs are assumed, then the problem is $\mathcal{NP}$-hard. Next, a dynamic programming algorithm is proposed, which runs in $\mathcal{O}\left(T^{(k+1)}\right)$, with k being the number of products. This algorithm assumes non-speculative costs to ensure that the zero-inventory property holds.

2.3 Lot-sizing with co-production

The dynamic uncapacitated ELS problem with co-production has been studied by Ağralı (2012). In this problem, only co-production exists. The problem is reduced to a single-item lot-sizing problem and after that a dynamic program has been used to solve the problem. A small adjustment to the dynamic program can be made when dealing with by-production. Pamuk (2016) studied the dynamic uncapacitated lot-sizing problem with co-production units and provided a MIP formulation. Next, the problem is proven to be $\mathcal{NP}$-hard. Furthermore, solution methods have been proposed for special cases, problems that can be solved within polynomial time. Finally, valid inequalities lowered the LP-bound by more than 20%.

3 Problem description

Assume that we have a set of $\mathcal{K}$ products, denoted with index k, that need to be produced over time horizon $\mathcal{T} = 1, \ldots, T$, denoted with index t. Those products are produced via different CPUs. Denote those units as a set $\mathcal{C} = 1, \ldots, C$. When co-product c is co-produced, all other products are produced $\alpha^{c,k}$ times. We assume that $\alpha^{c,k} = 0$ or $\alpha^{c,k} \geq 1$ $(\forall c \in \mathcal{C}, \forall k \in \mathcal{K})$. In case that $\alpha^{c,k}$ is between 0 and 1, the co-product is (slightly) changed to ensure it holds the assumption. This assumption is made to ensure that the setup costs are implemented correctly in the proposed model. Besides, we call $\alpha^{c,k}$ the co-production factor. Denote d_t^k as the demand for product k at time t $(\forall k \in \mathcal{K}, \forall t \in \mathcal{T})$ respectively. Denote c_t^c as the production costs at time t, $\forall t \in \mathcal{T}$, for co-product c. Setup costs at time t for co-product c are denoted by f_t^c. Every co-product has its own setup costs, because it is reasonable to assume that every co-product is produced by another machine. We also assume that all costs are nonnegative.

We also assume that economies of scale hold for all the CPUs. To be more precise, it is possible that a CPU i can be made of a linear combination of other CPUs. If this linear combination only has nonnegative and at least two positive factors, then we call the combination valid. If this does not hold, then it means that this CPU is worse than another CPU and can therefore be disregarded from the list of CPUs. Examples 3.1 and 3.2 give more clarity of why the combination should exist of only nonnegative and at least two positive terms. Denote $\omega^{i,c}$ as the factor to make CPU i using CPUs c, $c \in \mathcal{C} \setminus i$, using valid linear combinations. Mathematically speaking, economies of scale for co-product i means the following:

$$\beta \sum_{c \in \mathcal{C} \setminus i} \min\left\{\omega^{i,c}, 1\right\} f_t^c \leq f_t^i \leq \sum_{c \in \mathcal{C} \setminus i} \min\left\{\omega^{i,c}, 1\right\} f_t^c \qquad (0 < \beta < 1) \tag{1}$$

$$\gamma \sum_{c \in \mathcal{C} \setminus i} \omega^{i,c} c_t^c \leq c_t^i \leq \sum_{c \in \mathcal{C} \setminus i} \omega^{i,c} c_t^c \qquad (0 < \gamma < 1) \tag{2}$$

In the first equation (1), the fixed costs of producing CPU c are either taken into account or not taken into account. The equation of the variable costs are in Equation (2). This equation no longer needs to take the minimum of 1 and $\omega^{i,c}$. Next, both equation contain a parameter, β and γ, which indicate the economies of scale. Theoretically, these parameters could be 0 and 1. Since we want the strongest bounds, we exclude the parameters to be 0 and 1 to ensure that we benefit from economies of scale. If a parameter is 0, then the fixed or variable costs are 0. If both parameters are 1, then the co-products does not benefit from economies of scale. It is also possible that multiple valid linear combinations hold for a CPU i, then all of those combinations satisfy equations (1) and (2).

Example 3.1. Assume that we have 3 CPUs and 2 products that can be produced. Assume, we have co-production factor $\alpha^{c,k} = \{\{1, 0\}, \{0, 1\}, \{1, 1\}\}$. The first CPU only produces the first product, the second only the second, and the last CPU produces both CPUs. The last CPU is a combination of the first CPU and the second CPU: 1 time the first CPU and 1 time the second CPU. We verify that all factors are nonnegative, and at least two factors are positive. Hence, the last CPU in this example should benefit from economies of scale.

Example 3.2. If we assume the same products and CPUs as in Example 3.1, then the first CPU is a combination of producing the last CPU 1 time minus the second CPU 1 time. Since this combination does not only contain nonnegative factors, the first CPU does not need to benefit from economies of scale of this particular combination.

Next, we need to pay holding costs h_t^k for every product k that is not used in the current period t, but instead put in the inventory. Furthermore, denote $d_{i,j}^k$ as the cumulative demand for product k between periods i and j, $i \leq j$, $\forall i, j \in \mathcal{T}$. The inventory has a fixed capacity of B. The demand being satisfied in period t does not count towards the capacity limit of period t. Therefore, only the products put in the inventory count towards the inventory limit each period.

3.1 MBLP Formulation

To present a formulation of the problem, we need to come up with various decision variables. In the model, we use 3 different decision variables: setup costs variables (y_t^c), production variables (x_t^c) and inventory variables (s_t^k).

$$y_t^c = \begin{cases} 1, & \text{if setup costs needs to be paid at time } t \text{ for CPU } c, \\ 0, & \text{otherwise,} \end{cases}$$
$$x_t^c = \text{ amount of CPU } c \text{ produced in period } t,$$
$$s_t^k = \text{ amount of product } k \text{ put in inventory at time } t.$$

The setup costs variables are binary variables, since CPU c at time t is either produced or not produced at all. The production variables are nonnegative, since negative production is not possible. In contrary with the other decision variables, the total number of inventory variables are dependent on the number of products. We start without products in the inventory.

Using these decision variables, we now give a model to describe the uncapacitated lot sizing problem with inventory bounds and co-production units:

$$\min \quad \sum_{t\in\mathcal{T}} \left(\sum_{c\in\mathcal{C}} f_t^c y_t^c + \sum_{c\in\mathcal{C}} c_t^c x_t^c + \sum_{k\in\mathcal{K}} h_t^k s_t^k \right) \tag{3}$$
$$\text{s.t.} \quad \sum_{c\in\mathcal{C}} \alpha^{c,k} x_t^c + s_{t-1}^k = d_t^k + s_t^k \qquad \forall t \in \mathcal{T},\ \forall k \in \mathcal{K} \tag{4}$$
$$s_0^k = 0 \qquad \forall k \in \mathcal{K} \tag{5}$$
$$M y_t^c \geq x_t^c \qquad \forall t \in \mathcal{T},\ \forall c \in \mathcal{C} \tag{6}$$
$$\sum_{k\in\mathcal{K}} s_t^k \leq B \qquad \forall t \in \mathcal{T} \tag{7}$$
$$y_t^c \in \mathbb{B} \qquad \forall t \in \mathcal{T},\ \forall c \in \mathcal{C} \tag{8}$$
$$x_t^c \geq 0 \qquad \forall t \in \mathcal{T},\ \forall c \in \mathcal{C} \tag{9}$$
$$s_t^k \geq 0 \qquad \forall t \in \mathcal{T},\ \forall k \in \mathcal{K} \tag{10}$$

The objective (equation (3)) is to minimize setup costs, production costs and holding costs. The first constraint set (equation (4)) is to ensure that the demand is met and the correct amount of products are put in the inventory. We start without products in the inventory (equation (5)). Constraints (6) are to ensure that setup costs are incurred in the correct way. This constraint contains a M, which is defined below. Because of the fact that the co-production factor is undefined between 0 and 1 (exclusive), the term before y_t^c is always greater than or equal to the value of x_t^c. Equation (10) is to ensure that the inventory limit is never exceeded.

We want M as tight as possible to get the best objective value of the LP relaxation. To bound M as tight as possible, we take into account the capacity limits and the maximum demand. Then, the production variables are also bounded by the capacity limit. For CPUs, the maximum amount that can be produced to reach the inventory capacity is calculated as inventory capacity divided by the amount of produced items. The maximum demand can be calculated using:

$$\widetilde{d_{t,T}^{c,k}} = \begin{cases} \frac{d_{t,T}^k}{\alpha^{c,k}}, & \text{if } \alpha^{c,k} > 0, \\ -\infty, & \text{otherwise} \end{cases}$$

For a CPU c, the maximum CPUs needed to satisfy all demand, which this CPU is able to cover, is to take the maximum over every covered product k. And therefore, the following constraints are the tightest constraints to bound M and to obtain the lowest bound obtained when solving the LP-relaxation:

$$\min \left\{ \frac{B + \sum_{k\in\mathcal{K}} d_t^k}{\sum_{k\in\mathcal{K}} \alpha^{c,k}}, \max_{k\in\mathcal{K}} \widetilde{d_{t,T}^{c,k}} \right\} y_t^k \geq x_t^c \tag{11}$$

4 Complexity of the problem

In this section, we show the complexity of the problem. In Section 4.1, we prove that the problem is strongly $\mathcal{NP}$-hard. In the next subsection, Section 4.2, we show that we can not use the zero-inventory property when optimising. Even when we change it to a generalised variant of it, it does not hold. Finally, we give the outline of a branch-and-bound method to solve the problem in Section 4.3. Since the problem is strongly $\mathcal{NP}$-hard and the zero-inventory property does not hold, it is likely to use heuristics to find solutions.

4.1 Strongly $\mathcal{NP}$-Hard

To prove that this problem is strongly $\mathcal{NP}$-hard, we use the proof of Akbalik et al. (2015), in which the multi-item lot sizing problem is proven to be strongly $\mathcal{NP}$-hard. In the proof, we refer to set K as the set of all products, denoted with index k. After that, we reduce the multi-item uncapacitated lot-sizing problem with inventory bounds by restriction to the uncapacitated lot-sizing problem with inventory bounds and CPUs.

Theorem 4.1 (Akbalik et al. (2015))**.** The Multi-item uncapacitated lot-sizing problem with inventory bound is strongly $\mathcal{NP}$-hard, even with a stationary setup costs identical for all items and without holding costs.

Proof. We use a polynomial reduction from a strongly $\mathcal{NP}$-complete problem: 3-partition problem. An instance I consists of an integer D and a list of $3n$ integers $(a_1, \ldots, a_{3n})$ such that $\sum_k a_k = nD$ and $\frac{D}{4} < a_k < \frac{D}{2}$ for all indices. The goal is to find a partition $A_1 \cup \ldots \cup A_n$ of $\{1, \ldots 3n\}$ such that $\sum_{k \in A_j} a_k = D,\ \forall j$. Denote set $\mathcal{K}$ An instance I is transformed into an instance $\tau(I)$ of multi-item uncapaciated lot sizing with inventory bound as follows:

- We have $K = 3n + 1$ products. Items 1 till $3n$ are associated with integers $a_1, \ldots, a_{3n}$. The last product is the item to control the number of units in stock.
- We have $T = n + 1$ periods, in which every period has an inventory limit equal to $B = nD$.
- The setup costs for all products is $f = n(n + 1)D$ in every period. The production cost of product k is $c_{kt} = 2t$. For the last product, the production cost is 0, for all periods. We assumed that we do not have holding costs.
- The demands for product k are 0, with the exception of the last period. The demand in the last period is a_k. The last product has a demand of B for periods $1, \ldots, T - 1$. The total demand for every product is then nD.

We now try to find a feasible planning with maximum costs of $Z = Kf + n(n + 1)D$.

This transformation τ is polynomial in the size of I. Lemma 4.1 in Johnson (1985) proved that the 3-partition problem is strongly $\mathcal{NP}$-hard. We immediately show that τ is a pseudo-polynomial: the largest value in instance I is $\max(I) = D$. The largest value in $\tau(I)$ is:
$\max \tau(I) = f \in \mathcal{O}\left(I^2 \max(I)\right)$. This means that the sizes are polynomially related.

Besides, τ needs to be a reduction, that means that instance I is a positive instance of 3-partition if and only if $\tau(I)$ is a positive instance of the lot-sizing problem.

Assume I is positive: Consider a valid partition $A_1, \ldots, A_n$ for I, we construct a planning of cost at most Z for $\tau(I)$: for every period t, produce all products k such that $i \in A_t$. Next, produce all demands for the last product in the first period. This schedule is feasible, because we have precisely B products in the inventory at the end of the first period: $(n - 1)D$ of the last products and D from the products of A_1. In all upcoming periods, D of the last product are obtained from the inventory while D of the other products are produced. As a consequence, the inventory bound is tight at all periods. In this schedule, every product is produced in one period, this means that the setup costs are Kf. The production cost of period t is $2tD$: $2D + 4D + \ldots + 2nD = n(n+1)D$. Concluding, the solutions is: $Z = Kf + n(n + 1)D$.

Assume $\tau(I)$ is positive: This means that a feasible policy π^* of cost at most Z exists. We prove this by showing that two requirements hold: we produce every product once and keeping full capacity in all periods $t < T$. Any feasibly policy must produce each product at least once to satisfy demand. This means, pay at least Kf setup-costs. Besides, setup-costs are chosen such that $(k+1)f > Z$. This means that we verified the first requirement. Using this, the total demand for the last item has to be produced in te first period to satisfy positive demand: $d_1^k = D$. Denote $\bar{H}_t$ as the remaining inventory capacity. The last item has a demand of D in every period, we have $\bar{H}_t = tD$ for $t = 1, \ldots, n$. Consider a feasible policy π, in which we produce every product exactly once. To find a lowerbound, we consider a relaxation in which we do not have setup costs and inventory limits $\bar{H}_t$, and need to satisfy items 1 till $3n$. A feasible policy that is restricted to items 1 till $3n$ is a feasible solution for the relaxation. The production costs are exactly the same as π. Denote z_r^* as the optimal solution for the relaxation. Consequently, the lowerbound for a feasible policy $\tau(I)$ is $kf + z_r^*$. Next, the relaxation is a LP and can therefore be solved in polynomial time. The optimal solution can easily be found as producing D in every period. This means that the production costs are $2D + 4D + \ldots + 2nD = n(n+1)D$. Then, Z is a lowerbound on every feasible policy $\tau(I)$. The only way to attain this lowerbound, $z_r^* = n(n+1)D$, is to maintain full inventory at every period, except the last period. Thus, the feasible policy π^* also verifies the second requirement: produce D of the $3n$ first products in all periods $t = 1, \ldots, n$. The result is then obtained, because sets $A_t = \{k \leq 3n |$ product k is produced in period t in $\pi^*\}$ define a valid partition for instance I. ■

Theorem 4.2. The dynamic uncapacitated lot-sizing problem with inventory bounds and CPUs is strongly $\mathcal{NP}$-hard.

Proof. The multi-item lot-sizing problem with inventory bounds is proven to be strongly $\mathcal{NP}$-hard (Theorem 4.1). We only have to show that an instance from the multi-item lot-sizing problem with inventory bounds can be reduced to an instance of uncapacitated lot-sizing problem with inventory bounds and CPUs. Reduce an instance as follows. In the multi-item lot-sizing problem with inventory bounds, we introduce the set of CPUs $\mathcal{C}$. This set is in this particular problem the same as the set of products (set $\mathcal{K}$). We directly obtain the reduction by restriction. Since we are able to set the co-production factor $\alpha^{c,k}$ in this new setting as:

$$\alpha^{c,k} = \begin{cases} 1, & \text{if c = k,} \\ 0, & \text{otherwise.} \end{cases}$$

This means that all CPUs of set $\mathcal{C}$ only produce one product of set $\mathcal{K}$. Therefore, the multi-item uncapacitated lot-sizing problem with inventory bounds can be reduced to the uncapacitated lot-sizing problem with inventory bounds and co-production units. Hence, the uncapacitated lot-sizing problem with inventory bounds and co-production units is strongly $\mathcal{NP}$-hard. ■

4.2 Zero-inventory property

For some lot-sizing problems, polynomial time algorithms exist, since the zero-inventory property holds. When the zero-inventory property holds, dynamic programming algorithms based on this property exist. For our problem, it is (almost) trivial that this property does not hold without assumptions. Akbalik et al. (2015) proved that this property holds for multi-item lot sizing problems with inventory bounds if non-speculative costs is assumed. Non-speculative costs mean that the following equations hold:

$$c_{t+1}^k \leq c_t^k + h_t^k \quad \forall t \in \mathcal{T} \setminus \{T-1\}, \quad \forall k \in \mathcal{K} \tag{12}$$

Non-speculative costs in terms of CPUs can be defined as:

$$c_{t+1}^c \leq c_t^c + \sum_{k \in \mathcal{K}} \alpha^{c,k} h_t^k \quad \forall t \in \mathcal{T} \setminus \{T-1\}, \quad \forall c \in \mathcal{C} \tag{13}$$

We show in a later example (Example 4.1) that the zero-inventory property does not hold for our problem under non-speculative costs assumption.

Ağralı (2012) came up with a dynamic program for the dynamic uncapacitated lot-sizing problem with co-production, which only included one CPU such that $\alpha^{1,k} = 1, \forall k \in \mathcal{K}$. This property even holds with a varying co-production factor $\alpha^{1,k}$ ($\alpha^{1,k} \geq 1$). For both problems, the zero-inventory property is changed to a generalisation version of it: If at least one product has zero inventory, then the CPU is allowed to be produced. This generalised variant holds, because this problem can be reduced to the single-item lot-sizing problem in which the zero-inventory property holds.

Example 4.1. In this example, we assume non-speculative costs. We have 2 periods, 2 CPUs and 2 products that need to be produced: $\alpha^{c,k} = \{\{1,1\},\{1,0\}\}$ (first CPU produces both products, the second only produces the first product), demand $= ((0,10),(11,0))$. This means that the demand in the first period is 10 of the second product. The demand in the second period is 11 of the first product. Next, the inventory has a capacity of 10. Setup costs are 1 for both CPUs in both periods. Variable costs are 2 for the first CPU and 1 for the second CPU. Holding costs are always 0.

The optimal solution is to produce 10 times the first CPU in the first period and 1 time the second CPU in the second period: 23. In the first period, the second product can not be produced, otherwise the inventory limit is violated.

Theorem 4.3. The generalised zero-inventory property does, in general, not hold for the uncapacitated lot-sizing problem with inventory bounds and CPUs, even under non-speculative costs and at least one product of the produced co-product has no inventory left.

Proof. Using Example 4.1, in the optimal solution we produce the first product while the inventory is not zero. Producing the other cpu (which contains a product with zero inventory) does not lead to the optimal solution (solution value 24 instead of 23). ■

4.3 Branch-and-Bound

Since we are left with a problem that is strongly $\mathcal{NP}$-hard and in which we can not take advantage of the (generalised) zero-inventory property. Consequently, we have to come up with other solution methods. Branch-and-bound is a well-known solution method. The outline of a branch-and-bound algorithm is as follows: We take formulation (3)-(10). Then, we change Equation (8) to $y_t^c \in [0,1]$. That gives an LP formulation, which can easily be solved. For branching, we branch on the most fractional variable, which is strictly lower than 1:

$$\max_{t \in \mathcal{T}, c \in \mathcal{C}} y_t^c \quad \text{s.t.} \quad y_t^c < 1 \tag{14}$$

Then we branch on a variable y_i^o as: $y_i^o = 1$ and $y_i^o = 0\,(i \in \mathcal{T}, o \in \mathcal{C})$. After that, the new LP of both branches are solved. The branch with the lowest upperbound is explored first. In the upperbounds, the setup costs are calculated in the right way. This means that the upperbounds take the missed setup costs into accounts, and can therefore be calculated as:

$$UB = LB + \sum_{t \in \mathcal{T}} \sum_{c \in \mathcal{C}} \left(f_t^c \lceil y_t^c \rceil \right) \tag{15}$$

In the worst-case, we need to solve 2^{CT} problems. The total number of problems that need to be solved exponentially grows. Even for small problems, this is a problem. Therefore, we do not expect a branch-and-bound algorithm to solve to optimality within reasonable time, even for small instances.

5 Dantzig-Wolfe: Math programming heuristic

The decompositions are based on Pimentel et al. (2010). We perform a combination of the two decompositions they proposed: period and product decomposition. In Section 5.1, we decompose the MBLP formulation of Section 3.1. After that, we propose a Restricted Master Problem (RMP) in Section 5.2 with corresponding pricing problems in Section 5.3. Then, we generate solutions which satisfy all constraints in Section 5.4. A more exact approach (Branch-and-Price) can be found in Section 5.5. Finally, we give some more information about column management in Section 5.6.

5.1 Decomposing MBLP formulation

To perform a Dantzig-Wolfe decomposition, we need to have an LP formulation. Therefore, we relax the binary variable y_t^c in a similar way as how we proposed to solve branch-and-bound:

$$\min \quad \sum_{t\in\mathcal{T}}\left(\sum_{c\in\mathcal{C}} f_t^c y_t^c + \sum_{c\in\mathcal{C}} c_t^c x_t^c + \sum_{k\in\mathcal{K}} h_t^k s_t^k\right) \tag{16}$$

$$\text{s.t.} \quad \sum_{c\in\mathcal{C}} \alpha^{c,k} x_t^c + s_{t-1}^k = d_t^k + s_t^k \qquad \forall t\in\mathcal{T},\ \forall k\in\mathcal{K} \tag{17}$$

$$s_0^k = 0 \qquad \forall k\in\mathcal{K} \tag{18}$$

$$My_t^c \geq x_t^c \qquad \forall t\in\mathcal{T},\ \forall c\in\mathcal{C} \tag{19}$$

$$\sum_{k\in\mathcal{K}} s_t^k \leq B \qquad \forall t\in\mathcal{T} \tag{20}$$

$$\mathbf{y_t^c} \in [\mathbf{0},\mathbf{1}] \qquad \forall t\in\mathcal{T},\ \forall c\in\mathcal{C} \tag{21}$$

$$x_t^c \geq 0 \qquad \forall t\in\mathcal{T},\ \forall c\in\mathcal{C} \tag{22}$$

$$s_t^k \geq 0 \qquad \forall t\in\mathcal{T},\ \forall k\in\mathcal{K} \tag{23}$$

The Equations (16)-(23) are the same as Equations (3)-(10), with the exception of Equation (21).

Since we have a LP formulation, we need to split the constraints in two different types of constraints: linking constraints and independent constraints. We propose constraints (17), (18) and (20) to be the linking constraints and we propose constraints (19) to be the independent constraints. In this way, the constraint matrix is a block-angular constraint matrix.

To solve the problem, we reformulate the problem using production plans. A production plan contains the quantity to co-produce at a specific time. These production plans are obtained via solving the pricing problems. Let set $\mathcal{P}$ be the set of all production plans. For production plans, we use the following parameters in combination with production plans:

$$\beta_{tp}^c = \begin{cases} 1, & \text{if setup costs needs to be paid at time } t \text{ in product plan } p \text{ for CPU } c, \\ 0, & \text{otherwise} \end{cases}$$

γ_{tp}^c = amount of co-produced products at time t in product plan p for CPU c.
In a reformulated model, we use other decision variables than introduced before. The decision variables are v_{tp}^c: the weight of production plan p at time t for CPU c. We also use variables to keep track of inventory levels for every product, which is s_t^k. These variables are the same as introduced before. Since the pricing problems (as shown later) are bounded sets, we only consider production plans that are extreme points in the restricted master problems.

5.2 Restricted Master Problem

A formulation of the RMP of the Dantzig-Wolfe decomposition using production plans can be found below. This formulation remains a LP-relaxation of the linking constraints.

$$\min \quad \sum_{t\in\mathcal{T}}\sum_{k\in\mathcal{K}} h_t^k s_t^k + \sum_{p\in\mathcal{P}}\sum_{t\in\mathcal{T}}\sum_{c\in\mathcal{C}} \left(f_t^c \beta_{tp}^c + c_t^c \gamma_{tp}^c\right) v_{tp}^c \tag{24}$$

$$\text{s.t.} \quad \sum_{p\in\mathcal{P}} v_{tp}^c \leq 1 \qquad \forall t \in \mathcal{T},\ \forall c \in \mathcal{C}\ (\pi_t^c) \tag{25}$$

$$\sum_{p\in\mathcal{P}}\sum_{c\in\mathcal{C}} \left(\alpha^{c,k}\gamma_{tp}^c\right) v_{tp}^c + s_{t-1}^k = d_t^k + s_t^k \qquad \forall t \in \mathcal{T},\ \forall k \in \mathcal{K}\ \left(\mu_t^k\right) \tag{26}$$

$$\sum_{k\in\mathcal{K}} s_t^k \leq B \qquad \forall t \in \mathcal{T}(\xi_t) \tag{27}$$

$$s_0^k = 0 \qquad \forall k \in \mathcal{K} \tag{28}$$

$$v_{tp}^c \geq 0 \qquad \forall t \in \mathcal{T},\ \forall p \in \mathcal{P},\ \forall c \in \mathcal{C} \tag{29}$$

$$s_t^k \geq 0 \qquad \forall t \in \mathcal{T},\ \forall k \in \mathcal{K} \tag{30}$$

The objective (equation (24)) is to minimize the total costs. Constraints (25) are the convexity constraints to ensure only a selection of production plans at each time and for every CPU are chosen. The dual variables corresponding to these constraints are π_t^c. Constraints (26) ensure demand is met and correct inventory is stored, with corresponding dual variables μ_t^k. Constraints (27) ensure that the inventory bound is never exceeded. The corresponding dual variables are ξ_t. Constraints (28) are to ensure that we have no products at the start. Constraints (29) ensure that a production plan is (partially) chosen. Constraints (30) ensure no negative inventory exists.
The original variables can be obtained using the following equations:

$$x_t^c = \sum_{p\in\mathcal{P}} \gamma_{tp}^c v_{tp}^c \tag{31}$$

$$y_t^c = \sum_{p\in\mathcal{P}} \beta_{tp}^c v_{tp}^c \tag{32}$$

In the original constraints, we have $y_t^c \in \mathbb{B}$ $(\forall c \in \mathcal{C},\ \forall t \in \mathcal{T})$. To ensure these constraints hold, we give an outline of a branch-and-price algorithm in Section 5.5.

In the first iteration, we do not have any production plans added to the problem. Therefore, we introduce artificial variables to create feasible solutions to the restricted master problem. These variables, however, should never be part of the final solution. A new artificial variable is made for every product $k \in \mathcal{K}$ and in every period $t \in \mathcal{T}$. These artificial variables produce all products and ensure nothing is put in the inventory. To ensure that they are not part of the final solution, we have to put the costs of these variables at a large value. This value should be at least the final objective. To ensure that we obtain a value that is at least the final objective, we calculate the costs of the artificial variables as:

$$\sum_{t\in\mathcal{T}}\left(\sum_{c\in\mathcal{C}}\left(f_t^c + c_t^c \frac{B}{\sum_{k\in\mathcal{K}}\alpha^{c,k}} + \sum_{k\in\mathcal{K}} h_t^k \frac{B}{\alpha^{c,k}}\right)\right) \tag{33}$$

These costs are calculated as: the total of all fixed costs plus the maximum possible variable costs for every period and every CPU plus the maximum holding costs for every product for every period. It is trivial that the artificial variables with these costs are never part of the final solution.

5.3 Pricing Problems

The Restricted Master Problem has CT pricing problems. The pricing problems (PPs) contain the remaining constraints. The reduced costs of a variable v_{tp}^c are subtracted from the objective

of these pricing problems. The reduced costs of a variable v_{tp}^c of constraint (26) is:

$$\mu_t^k \left(\sum_{p \in \mathcal{P}} \sum_{c \in \mathcal{C}} \left(\alpha^{c,k} \gamma_{tp}^c \right) v_{tp}^c \right) \tag{34}$$

Using equations (31), it becomes:

$$\mu_t^k \alpha^{c,k} x_t^c \tag{35}$$

To obtain the reduced costs of constraints (27), we need to rewrite the constraints:

$$\sum_{k \in \mathcal{K}} s_t^k \leq B \tag{36}$$

$$\Leftrightarrow \sum_{p \in \mathcal{P}} \left(\sum_{k \in \mathcal{K}} \sum_{c \in \mathcal{C}} \alpha^{c,k} \gamma_{tp}^c \right) v_{tp}^c \leq B - \sum_{k \in \mathcal{K}} \left(s_{t-1}^k - d_t^k \right) \tag{37}$$

Equation (37) contains the production plan variables. Those variables are used to obtain the dual constraints.

$$\xi_t \left(\sum_{p \in \mathcal{P}} \left(\sum_{k \in \mathcal{K}} \sum_{c \in \mathcal{C}} \alpha^{c,k} \gamma_{tp}^c \right) v_{tp}^c \right) \tag{38}$$

$$\Leftrightarrow \xi_t \left(\sum_{k \in \mathcal{K}} \sum_{c \in \mathcal{C}} \alpha^{c,k} x_t^c \right) \tag{39}$$

The reduced costs of a variable v_{tp}^c of constraints (25) are trivial: π_t^c.

Now, we provide a formulation of the pricing problems that need to be solved. The formulation of the pricing problems can be found below:

$$\min \quad f_t^c y_t^c + c_t^c x_t^c - \left(\sum_{k \in \mathcal{K}} \alpha^{c,k} x_t^c \right) \mu_t^k - \left(\sum_{k \in \mathcal{K}} \alpha^{c,k} x_t^c \right) \xi_t - \pi_t \tag{40}$$

$$\text{s.t.} \quad \min \left\{ \frac{B + \sum_{k \in \mathcal{K}} d_t^k}{\sum_{k \in \mathcal{K}} \alpha^{c,k}}, \max_{k \in \mathcal{K}} \widetilde{d_{t,T}^{c,k}} \right\} y_t^c \geq x_t^c \tag{41}$$

$$x_t^c \geq 0 \tag{42}$$

$$y_t^c \in [0, 1] \tag{43}$$

The objective (Equation (40)) is to minimize the costs taking the dual variables of the restricted master problem into account. Constraint (41) are to ensure that the problem becomes bounded, as we have shown before. Constraints (42) and (43) are the constraints that belong to the decision variables.

To solve these pricing problems, we can rewrite the objective. A rewritten objective helps to find the optimal solution. The objective can easily be rewritten to:

$$\min \quad f_t^c y_t^c - \pi_t + x_t^c \left(c_t^c - \left(\sum_{k \in \mathcal{K}} \alpha^{c,k} \mu_t^k \right) - \left(\sum_{k \in \mathcal{K}} \alpha^{c,k} \right) \xi_t \right) \tag{44}$$

A problem with this objective and constraints (41)-(43) can easily be solved. By inspection, we observe that two solutions are possible: $y_t^c = 0$ and $y_t^c = 1$. In case that $y_t^c = 0$, then $x_t^c = 0$, because of constraint (41). In case that $y_t^c = 1$, then $x_t^c = \min \left\{ \frac{B + \sum_{k \in \mathcal{K}} d_t^k}{\sum_{k \in \mathcal{K}} \alpha^{c,k}}, \max_{k \in \mathcal{K}} \widetilde{d_{t,T}^{c,k}} \right\}$. y_t^c is always 0 or 1, because we assumed that $f_t^c \geq 0$.

5.4 Heuristic: Generating feasible solutions

When all columns are added, the solution we obtain is the solution to the LP-relaxation. We then need to generate a feasible solution to the problem. We do that using 3 different methods. The easiest way can be found in Section 5.4.1. Another way can be found in Section 5.4.2. A method inspired by relax and fix & fix and optimise method can be found in Section 5.4.3.

5.4.1 Feasible setup-costs

After the LP-relaxation has been solved, we are left with columns (and setup costs) that are only taken partially into account. We do not guarantee that those setup costs sum up to 1. Therefore, we add the missing setup-costs of the last iteration to the problem to get a feasible solution. If lots of columns are only added for a small amount, this could give bad optimality gaps.

5.4.2 Solve MBLP with specific columns

Another way to generate feasible solutions, is to solve a MBLP formulation, but only include the columns that are used in the last iteration. Since the solution space is limited, a commercial solver, such as Cplex, might be able to find a solution within reasonable time. This solution does not have to be the optimal solution to the original problem. Denote set $\mathcal{Q}$ as the set of production plans that are used in the last iteration to solve the problem described by equations (24)-(30). We change this formulation by removing the setup costs from the production plan variables and re-adding the setup costs variables y_t^c. A MBLP formulation, which includes production plans and binary setup costs is:

$$\min \quad \sum_{t\in\mathcal{T}}\sum_{k\in\mathcal{K}} h_t^k s_t^k + \sum_{t\in\mathcal{T}}\sum_{c\in\mathcal{C}} f_t^c y_t^c + \sum_{t\in\mathcal{T}}\sum_{q\in\mathcal{Q}}\sum_{c\in\mathcal{C}} \left(c_t^c \gamma_{tq}^c\right) v_{tq}^c \tag{45}$$

$$\text{s.t.} \quad \sum_{q\in\mathcal{Q}} v_{tq}^c \leq 1 \qquad \forall t\in\mathcal{T},\ \forall c\in\mathcal{C} \tag{46}$$

$$\sum_{q\in\mathcal{Q}}\sum_{c\in\mathcal{C}} \left(\alpha^{c,k}\gamma_{tq}^c\right) v_{tq}^c + s_{t-1}^k = d_t^k + s_t^k \qquad \forall t\in\mathcal{T},\ \forall k\in\mathcal{K} \tag{47}$$

$$\sum_{k\in\mathcal{K}} s_t^k \leq B \qquad \forall t\in\mathcal{T} \tag{48}$$

$$s_0^k = 0 \qquad \forall k\in\mathcal{K} \tag{49}$$

$$M y_t^c \geq \sum_{q\in\mathcal{Q}} \left(\alpha^{c,k}\gamma_{tq}^c\right) v_{tq}^c \qquad \forall t\in\mathcal{T},\ \forall c\in\mathcal{C} \tag{50}$$

$$v_{tq}^c \geq 0 \qquad \forall t\in\mathcal{T},\ \forall q\in\mathcal{Q},\ \forall c\in\mathcal{C} \tag{51}$$

$$s_t^k \geq 0 \qquad \forall t\in\mathcal{T},\ \forall k\in\mathcal{K} \tag{52}$$

$$y_t^c \in \mathbb{B} \qquad \forall t\in\mathcal{T},\ \forall c\in\mathcal{C} \tag{53}$$

The objective (Equation (45)) is to minimze the total costs, using setup costs, holding costs and production plans. The production plans, do not contain the setup costs. Equations (46), (47), (48), (51) and (52) are the same as Equations (25), (26), (27), (29) and (30). Equations (50) are to ensure that setup costs are taken into account when producing the corresponding CPU. Equations (53) are to ensure that setup costs are binary variables.

5.4.3 Heuristic inspired by Relax and Fix & Fix and Optimise

Relax and fix & Fix and optimise is way to generate feasible solutions. This is done by cutting the problem into smaller problems. Then, we solve the first small problem (and relaxing the rest) and then using that (fixed) solutions to solve the next small problem. After treating all small problems, we obtain a feasible solution. Then, we again solve the small problems using this obtained solution, but we optimise it.

For the Relax and Fix part, we cut down the total number of periods to p to obtain smaller problems to solve. The last small problem exists of p or less periods. Next, we only take the columns into account that are part of the solution of the last iteration. Then, we iteratively add the column which has the highest value of v_{tp}^c in these p periods. We add columns until the solution becomes feasible up to this period. When the solution becomes feasible, we move up to the next period. If all columns are added in a small problem and the solution remains infeasible, we go back a period and add more columns until the solution becomes feasible. Then we go to the next period, until all periods are treated. We set $p = 5$.

When all periods are treated, we go to the Fix and Optimise part. In this part, we use this feasible solution obtained by the relax and fix part and remove columns. We start removing columns which were added the last. If those columns can be removed without creating an infeasible solution and it results in lowering the total costs, the column stays removed and we try removing the next column, until all columns are potentially removed. This gives a feasible solution to the problem, using the generated columns.

5.5 Branch-and-Price

Branch-and-Price is a combination of column generation and branch-and-bound. In this case, column generation is the Dantzig-Wolfe Decomposition. A branch-and-bound procedure is needed since setup costs variables are binary. This is not considered in the restricted master problem. Recall that the setup costs variables are calculated as: $y_t^c = \sum_{p \in \mathcal{P}} \beta_{tp}^c v_{tp}^c$ (Equation (32)). Therefore, we apply branch-and-price to the problem to ensure that the setup costs are calculated in a correct way. Branching rules are: branch on the most fractional variable y_t^c. Denote subsets $\tilde{P}$ as the production plans that are currently in the Restricted Master Problem. Next, add the following two nodes: node $\sum_{p \in \tilde{P}} \beta_{tp}^c v_{tp}^c = 0$ and node $\sum_{p \in \tilde{P}} \beta_{tp}^c v_{tp}^c = 1$ to the branching tree. These are called the branching constraints.

When the branching constraints are added, we modify the problem to the following. In case that $y_t^c = 0$, the PP corresponding to this variable is never being solved. This ensures that this restriction is handled correctly. In case that $y_t^c = 1$, we add the setup costs corresponding to this variable to the objective of the RMP. Next, we set the setup costs to 0 for this variable in both the RMP as well as in the corresponding PP. This implicates that when solving this PP, only the variable costs are taken into account. This means for the RMP that the fixed costs of the column do not include setup costs:

$$\beta_{tp}^c = 0 \tag{54}$$

Using the same argument as in Section 4.3, it is very unlikely that the optimal solution is found using branch-and-price because in the worst case 2^{CT} nodes need to be treated. Therefore, we use a breadth-first searching method to decrease the optimality gap in only solving a few additional problems. The upperbounds are calculated by rounding up all setup costs, similar to the approach given in Section 5.4.1.

5.6 Column Management

After solving the PPs, we add columns with negative reduced costs to the RMP. It is possible to add multiple columns with negative reduced costs in each iteration. We consider different options, but we always add the column with the most negative reduced costs: We start by only adding the most negative column found. Then, we increase that number by also adding the second highest in the same iteration. We consider different options in total columns added in every iteration. Besides, we also add columns based on the total number of pricing problems solved. It is also possible to remove added columns if those columns are not part of the optimal solution of the RMP for a few iterations or to add the first few columns found with negative reduced costs. We did not implement these options.

6 Decomposition without predetermined production

In the previous section (Section 5), we came up with a Dantzig-Wolfe formulation to solve the problem. Now, we try a similar approach, but instead of adding columns with predetermined production in every iteration, we do not predetermine the production. We go into more detail about this in Section 6.1. After that, we propose how to solve this new Restricted Master Problem in Section 6.2 and how to solve the corresponding pricing problems in Section 6.3. The generated solution is already feasible, but we try to lower the objective value in an easy way in Section 6.4. This decomposition does not have a mathematical background, therefore we predict the obtained objective values after every iteration, in Section 6.5. Since column management affects the solution, we explain why it is important to take it into consideration in Section 6.6.

6.1 Decomposing MBLP formulation

We do not use a LP relaxation, instead in each iteration we allow CPU(s) to produce and fixing the corresponding setup-costs variable to 1. Then, if a solution appears without the need of artifical variables, it is feasible. Artificial variables ensure that the solution obtained by solving the RMP becomes feasible. Every artificial variable ensures that the demand of one product at a time is fully convered. For every product and every period, a new artificial variable is made, resulting in KT artificial variables. The costs of the artificial variables are calculated in the same way as in equation (33):

$$\sum_{t\in\mathcal{T}}\left(\sum_{c\in\mathcal{C}}\left(f_t^c + c_t^c\frac{B}{\sum_{k\in\mathcal{K}}\alpha^{c,k}} + \sum_{k\in\mathcal{K}} h_t^k\frac{B}{\alpha^{c,k}}\right)\right) \tag{55}$$

Next, we split constraints into two different sets. Those sets are the same as introduced in Section 5. Equations (16)-(23). Constraints (17), (18), (20) are the linking constraints and (19) are the independent constraints.

6.2 Restricted Master Problem

We solve this RMP in a different way than as introduced in Section 5.2. Instead of adding columns in which we predetermine how many co-products are added to the problem. We now add a column which makes it possible to co-produce a CPU c in period t. The amount is determined when solving this problem. When adding such a column, the objective of the restricted master problem is increased by the setup-costs incurred by adding these costs. Every time a column is added, the setup costs are added to the objective of the Restricted Master Problem (f_t^c). This means that y_t^c is no longer a variable, but a parameter in this part.

After that, we need to include production plans, which are similar to the production plans as described in Section 5.1. A production plan is a plan which contains the quantity to co-produce at a specific time. These production plans are obtained via solving the pricing problems. Denote $\mathcal{R}$ as the set of all production plans. For production plans, we use the following parameters in combination with production plans:

$$\gamma_{tr}^c = \begin{cases} 1, & \text{if production plan } r \text{ contains CPU } c \text{ at time } t, \\ 0, & \text{otherwise} \end{cases}$$

This parameter is always used in combination with $\alpha^{c,k}$ to ensure that the right ratios hold of the CPUs hold. In the reformulated model, we include new decision variables than introduced before. The decision variables are v_{tr}^c: the weight of production plan r at time t for CPU c. Since we do not predetermine how much to co-produce of every product, we do not need to determine a subset anymore.

A formulation of the restricted master problem using production plans and y_t^c parameters can be found below.

$$\min \quad \sum_{t\in\mathcal{T}}\sum_{k\in\mathcal{K}} h_t^k s_t^k + \sum_{t\in\mathcal{T}}\sum_{c\in\mathcal{C}} f_t^c y_t^c + \sum_{r\in\mathcal{R}}\sum_{t\in\mathcal{T}}\sum_{c\in\mathcal{C}} \left(c_t^c \gamma_{tr}^c\right) v_{tr}^c \tag{56}$$

$$\text{s.t.} \quad \sum_{r\in\mathcal{R}}\sum_{c\in\mathcal{C}} \left(\alpha^{c,k}\gamma_{tr}^c\right) v_{tr}^c + s_{t-1}^k = d_t^k + s_t^k \qquad \forall t\in\mathcal{T},\ \forall k\in\mathcal{K}\ \left(\mu_t^k\right) \tag{57}$$

$$\sum_{k\in\mathcal{K}} s_t^k \leq B \qquad \forall t\in\mathcal{T}(\xi_t) \tag{58}$$

$$s_0^k = 0 \qquad \forall k\in\mathcal{K} \tag{59}$$

$$v_{tr}^c \geq 0 \qquad \forall t\in\mathcal{T},\ \forall r\in\mathcal{R},\ \forall c\in\mathcal{C} \tag{60}$$

$$s_t^k \geq 0 \qquad \forall t\in\mathcal{T},\ \forall k\in\mathcal{K} \tag{61}$$

This restricted master problem is similar to the restricted master problem in Section 5.2. Since we do not include how much to co-produce, we do not need convexity constraints. Constraints (57)-(61) are almost the same as constraints (26)-(30). The difference is the set of the production plans. Since the final solution is feasible (y_t^c is fixed), we do not need to create feasible solutions afterwards. The x_t^c-variable can be obtained using the following equations:

$$x_t^c = \sum_{r\in\mathcal{R}} \gamma_{tp}^c v_{tr}^c \tag{62}$$

6.3 Pricing Problems

The Restricted Master Problem has CT pricing problems. The pricing problems (PPs) contain the remaining constraints. The reduced costs of a variable v_{tr}^c are subtracted from the objective of these pricing problems. These reduced costs are similar to those described in Section 5.3. We therefore only include the final equations of all constraints. The reduced costs of a variable v_{tr}^c of constraint (57) is:

$$\mu_t^k \alpha^{c,k} x_t^c \tag{63}$$

The reduced costs of constraints (58) are obtained by rewriting the constraints in a similar way as what has been done in Equation (37). After rewriting the constraints, we obtain the following reduced costs:

$$\xi_t \left(\sum_{k\in\mathcal{K}}\sum_{c\in\mathcal{C}} \alpha^{c,k} x_t^c\right) \tag{64}$$

These reduced costs are subtracted from the pricing problems. Now, we propose a model to solve the pricing problems, which can be found below:

$$\min \quad f_t^c y_t^c + c_t^c x_t^c - \left(\sum_{k\in\mathcal{K}} \alpha^{c,k} x_t^c\right) \mu_t^k - \left(\sum_{k\in\mathcal{K}} \alpha^{c,k} x_t^c\right) \xi_t \tag{65}$$

$$\text{s.t.} \quad \min\left\{\frac{B + \sum_{k\in\mathcal{K}} d_t^k}{\sum_{k\in\mathcal{K}} \alpha^{c,k}}, \max_{k\in\mathcal{K}} \widetilde{d_{t,T}^{c,k}}\right\} y_t^k \geq x_t^c \tag{66}$$

$$x_t^c \geq 0 \tag{67}$$

$$y_t^c \in \mathbb{B} \tag{68}$$

These pricing problems are similar as the pricing problems obtained in Equations (40)-(43). To solve these pricing problems, we rewrite the objective to:

$$\min \quad f_t^c + x_t^c \left(c_t^c - \left(\sum_{k\in\mathcal{K}} \alpha^{c,k} \mu_t^k\right) - \left(\sum_{k\in\mathcal{K}} \alpha^{c,k}\right) \xi_t\right) \tag{69}$$

A problem with this objective and constraints (66)-(68) can easily be solved. By inspection, we observe that two solutions are possible: $y_t^c = 0$ and $y_t^c = 1$. In case that $y_t^c = 0$, then $x_t^c = 0$, because of constraint (66). In case that $y_t^c = 1$, then $x_t^c = \min\left\{\frac{M+\sum_{k\in\mathcal{K}} d_t^k}{\sum_{k\in\mathcal{K}} \alpha^{c,k}}, \max_{k\in\mathcal{K}} \widetilde{d_{t,T}^{c,k}}\right\}$.

6.4 Constructing feasible solution

The solution we obtain is feasible if no artificial variables are needed in the solution obtained by the RMP. The fixed costs of those unused variables are mistakenly added to the objective value. Those setup costs need to be subtracted from the objective value to get the right objective value. Since we do not have a mathematical proof of convergence, it does not automatically mean that the optimal solution is obtained in the last iteration. That also means that we can not guarantee that a feasible solution is found.

6.5 Obtained objective values

In every iteration, we add columns to the RMP. In this RMP, we add the setup costs of those added columns to the objective value. After that, we solve the LP and determine how many CPUs we produce. When only a few columns are added to the RMP, we have low setup costs, but we start producing CPUs that cover all products. When only producing a few different CPUs, we have a chance to overproduce multiple products. However, the demand is satisfied among all products and hence the solution is feasible. Nevertheless, the procedure does not stop since other CPUs can be added to this problem to ensure less products are overproduced, leading to higher setup costs. Despite the fact of unused setup costs are subtracted from the objective value, multiple different CPUs are getting produced, in small quantities. Therefore, we expect the objective value to decrease in the first iterations, and start increasing after that.

6.6 Column Management

Adding multiple columns at once, affect the objective value since those setup-costs are subtracted immediately. Even if no production occurs. Therefore, we try to find a good way of maximum added columns every iteration. Instead of tuning this parameter, we make use of the tuned parameter of the Dantzig-Wolfe decomposition as described in Section 5.

7 Dantzig-Wolfe: Period Decomposition

This decomposition is based on the period decomposition as given in Pimentel et al. (2010). In Section 7.1, we decompose the formulation of Section 3.1. This is similar as previously done in Section 5.1. The difference is that this decomposition adds production plans to the RMP, which contain multiple CPUs. Therefore, less pricing problems need to be solved. After that, we show a Restricted Master Problem corresponding with this decomposition in Section 7.3. This decomposition, however, is not implemented.

7.1 Decomposing MBLP formulation

To solve this Dantzig-Wolfe decomposition, we again solve the LP formulation. This LP formulation is the same as in Section 5.1 (Equations (16)-(23)).

In this decomposition, we need to split the constraints into: linking and independent constraints. We propose constraints (17), (18) and (20) to be the linking constraints. Then, constraints (19) are the independent constraints. In this way, the constraint matrix is a block-angular constraint matrix. The corresponding pricing problems are again bounded problems. Then, each period t corresponds with a pricing problem.

To solve this problem, we reformulate the problem using production plans. Production plans contain the quantity to co-produce at every period. These plans contain a mix of different CPUs. Denote set $\mathcal{D}$ as the set of all production plans obtained. Consider the following parameters:

$$\beta_{td}^{c} = \begin{cases} 1, & \text{if setup costs needs to be payed at time } t \text{ in product plan } d \text{ for CPU } c, \\ 0, & \text{otherwise} \end{cases}$$

γ_{td}^{c} = amount of co-produced products at time t in product plan d for CPU c.

The decision variables are v_{td}: the weight of production plan o at time t and s_t^k, which is the same variable as used in other formulations. A formulation of the restricted master problem of the period decomposition is given below. The pricing problems are bounded problems, and therefore, we only consider production plans that are extreme points.

7.2 Restricted Master Problem

A formulation of the restricted master problem of the period decomposition is given below. This formulation is a linear relaxation.

$$\min \quad \sum_{t\in\mathcal{T}}\sum_{k\in\mathcal{K}} h_t^k s_t^k + \sum_{d\in\mathcal{D}}\sum_{t\in\mathcal{T}}\sum_{c\in\mathcal{C}} \left(f_t^c \beta_{td}^c + c_t^c \gamma_{td}^c\right) v_{td} \tag{70}$$

$$\text{s.t.} \quad \sum_{d\in\mathcal{D}} v_{td} \leq 1 \qquad \forall t \in \mathcal{T}\,(\pi_t) \tag{71}$$

$$\sum_{d\in\mathcal{D}}\sum_{c\in\mathcal{C}} \left(\alpha^{c,k}\gamma_{td}^c\right) v_{td} + s_{t-1}^k = d_t^k + s_t^k \qquad \forall t \in \mathcal{T},\ \forall k \in \mathcal{K}\,\left(\mu_t^k\right) \tag{72}$$

$$\sum_{k\in\mathcal{K}} s_t^k \leq B \qquad \forall t \in \mathcal{T}(\xi_t) \tag{73}$$

$$s_0^k = 0 \qquad \forall k \in \mathcal{K} \tag{74}$$

$$v_{td} \geq 0 \qquad \forall t \in \mathcal{T},\ \forall d \in \mathcal{D} \tag{75}$$

$$s_t^k \geq 0 \qquad \forall t \in \mathcal{T},\ \forall k \in \mathcal{K} \tag{76}$$

The objective (equation (70)) is to minimize the total costs. Constraints (71) are the convexity constraints to ensure only a selection of production plans at each time is chosen. The dual variables corresponding to these constraints are π_t. Constraints (72) ensure demand is met and

correct inventory is stored, with corresponding dual variables μ_t^k. Constraints (73) ensure that the inventory bound is never exceeded. The corresponding dual variables are ξ_t. Constraints (74) are to ensure that we have no products at the start. Constraints (75) are to ensure a production plan is (partially) chosen. Constraints (76) ensures no negative inventory exists. Since the pricing problems (as shown later) are bounded sets, the restricted master problems only contains extreme points and therefore does not need to take extreme directions into account.

The original variables can be obtained using the following equations:

$$x_t^c = \sum_{d \in \mathcal{D}} \gamma_{td}^c v_{td} \tag{77}$$

$$y_t^c = \sum_{d \in \mathcal{D}} \beta_{td}^c v_{td} \tag{78}$$

In the original constraints, we have $y_t^c \in \mathbb{B}$ ($\forall c \in \mathcal{C},\ \forall t \in \mathcal{T}$). Therefore, the solutions obtained using this decomposition are not feasible. They need to be changed to become feasible to the original formulation, preferably using a heuristic.

7.3 Pricing Problems

The Restricted Master Problem has T pricing problems, with the remaining constraints in it. The reduced costs of a variable v_{td} are subtracted from the objective of these pricing problems. The reduced costs of a variable v_{td} of constraint (72) is:

$$\mu_t^k \left(\sum_{d \in \mathcal{D}} \sum_{c \in \mathcal{C}} \alpha^{c,k} \gamma_{td}^c v_{td} \right) \tag{79}$$

Using equation (77), it becomes:

$$\mu_t^k \alpha^{c,k} x_t^c \tag{80}$$

Similarly, the reduced costs of constraints (73) are:

$$\xi_t \left(\sum_{d \in \mathcal{D}} \sum_{k \in \mathcal{K}} \sum_{c \in \mathcal{C}} \alpha^{c,k} \gamma_{td}^c v_{td} \right) \tag{81}$$

$$\Leftrightarrow \xi_t \left(\sum_{k \in \mathcal{K}} \sum_{c \in \mathcal{C}} \alpha^{c,k} x_t^c \right) \tag{82}$$

The reduced costs of a variable v_{td} of constraints (71) are trivial: π_t. These reduced costs are subtracted from the pricing problems. A formulation of the pricing problems can be found below:

$$\min \quad \sum_{c \in \mathcal{C}} f_t^c y_t^c + \sum_{c \in \mathcal{C}} c_t^c x_t^c - \left(\sum_{k \in \mathcal{K}} \sum_{c \in \mathcal{C}} \alpha^{c,k} x_t^c \mu_t^k \right) - \left(\sum_{k \in \mathcal{K}} \sum_{c \in \mathcal{C}} \alpha^{c,k} x_t^c \right) \xi_t - \pi_t \tag{83}$$

$$\text{s.t.} \quad \min \left\{ \frac{B + \sum_{k \in \mathcal{K}} d_t^k}{\sum_{k \in \mathcal{K}} \alpha^{c,k}}, \max_{k \in \mathcal{K}} \widetilde{d_{t,T}^{c,k}} \right\} y_t^k \geq x_t^c \qquad \forall c \in \mathcal{C} \tag{84}$$

$$\sum_{k \in \mathcal{K}} \sum_{c \in \mathcal{C}} \alpha^{c,k} x_t^c \leq M + \sum_{k \in \mathcal{K}} d_t^k \tag{85}$$

$$x_t^c \geq 0 \qquad \forall c \in \mathcal{C} \tag{86}$$

$$y_t^c \in \mathbb{B} \qquad \forall c \in \mathcal{C} \tag{87}$$

$$\tag{88}$$

The objective (Equation (83)) is to minimize the costs taking the dual variables of the restricted master problem into account. Constraints (84) ensure that the setup costs are taken into account correctly. Furthermore, it bounds this problem. Constraint (85) ensures that the obtained solution does not exceed the inventory limit corresponding at the current period t. The objective can be rewritten:

$$\min \sum_{c \in \mathcal{C}} f_t^c y_t^c + \sum_{c \in \mathcal{C}} x_t^c \left(c_t^c - \sum_{k \in \mathbb{K}} \left(\alpha^{c,k} \mu_t^k + \alpha^{c,k} \xi_t \right) \right) - \pi_t \tag{89}$$

$$\max - \sum_{c \in \mathbb{C}} f_t^c y_t^c + \sum_{c \in \mathbb{C}} x_t^c \left(\sum_{k \in \mathbb{K}} \left(\alpha^{c,k} \mu_t^k + \alpha^{c,k} \xi_t \right) - c_t^c \right) + \pi_t \tag{90}$$

This means that the pricing problems are continous knapsack problems with setup costs, containing C products. Michel et al. (2009) proposed a branch-and-bound method to solve this problem.

8 Results

In this section, we present the results of the proposed methods. We start this section by providing some information about the data we were given (Section 8.1). After that, we tune parameters which indicates how many columns are added to the RMP in every iteration in Section 8.2. Then, we provide the optimality gaps of the methods (Section 8.3). To give more information about the found solutions, we also provide information about the costs distribution in Section 8.4. In the heuristic as described in Section 5.4.3, we described an optimisation part. In Section 8.5, we provide results about the performance of the optimisation part. In Section 6, we changed the way we add columns to the RMP. With this change, the objectives values do not necessarily converge. Therefore, we present the objective values which are obtained in every iteration in Section 8.6. All computations are made using a computer with a CPU speed of 2.93GHz and 2 processors. The methods are implemented in Java, using CPLEX version 18.0.0.

8.1 Data

In this section, we provide information about the data. We are provided data containing multiple instances. We divided the data into three smaller data sets. We refer to these sets as: small, medium and large. Then, we choose 10 instances to ensure a mix of different periods, products and CPUs are used. In this data set, we are given all CPUs, all products, co-production factor, demand and all costs. We set the inventory bounds in Section 8.1.4.

Table 1 contains information about the total number of periods, products and CPU in every data set. The total number of periods, products and CPUs is much lower at the small data set. The other data sets contain more periods, products and CPUs. The medium data set is limited to contain either 36 periods or 200 CPUs. This implies that the medium and large data set contain instances overlapping in the total number of periods, products and CPUs. The large data set, in contrast to the medium data set, is allowed to have 36 periods and 200 CPUs.

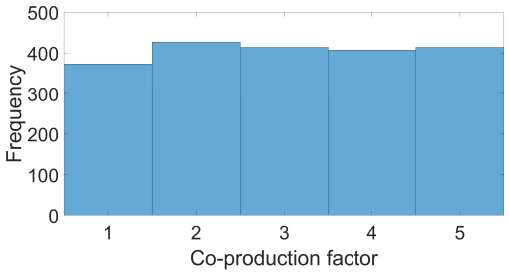
Table 1: Information about total number of periods, products and CPUs in every data set

	Small	Medium	Large
Periods	12-24	24-36	36
Products	10-40	40	40
CPUs	20-120	120-200	120-200

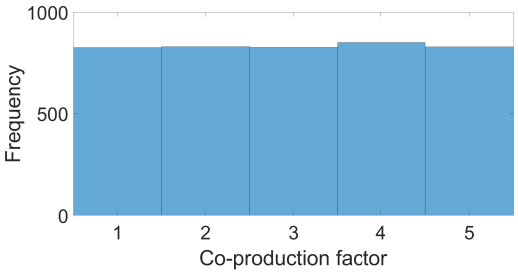
In the next subsubsections, we present some information about the co-production factor, demand, variable costs, setup costs, holding costs and inventory limit.

8.1.1 Co-production factor

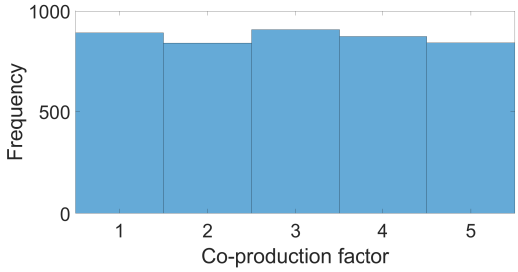
In Figure 1, histograms can be found of the co-production factor, in which all values of zero are excluded. These are excluded, considering that the co-production factor is 93.13% zero in the small, 93.80% in the medium and 93.81% in the large data set. Figures including zero can be found in Appendix B.1, Figure 11. Since the co-production factors are mainly zero, multiple CPUs are required to be able to produce every product. This could lead towards a production scheme in which not all products are produced in every period, but are instead intensively produced in specific periods to satisfy demand.



(a) Histogram of the co-production factor of the small data set excluding zero



(b) Histogram of the co-production factor the medium data set excluding zero

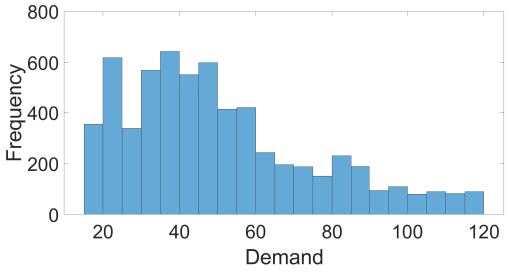


(c) Histogram of the co-production factor of the large data set excluding zero

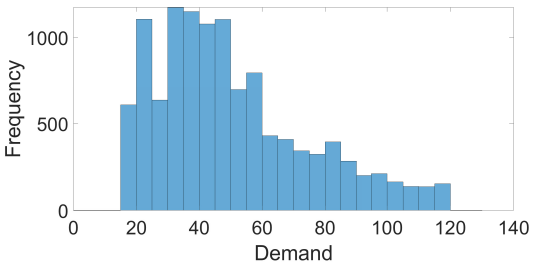
Figure 1: Histograms of the co-production factor of all three data sets excluding zero

8.1.2 Demand

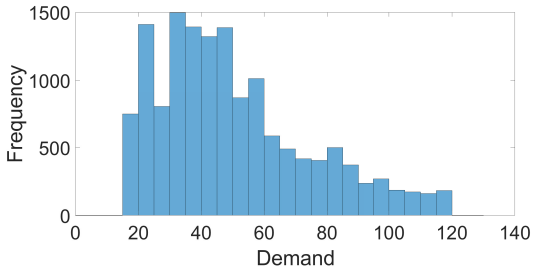
In Figure 5, we show histograms of the demand for every product k in every period t in one graph. This means that all demands of all products are put in one graph. The demand is always greater than zero for every product. Moreover, the demand vary a lot, meaning that we deal with dynamic lot-sizing problems.



(a) Histogram of Demand of the small data set



(b) Histogram of Demand of the medium data set



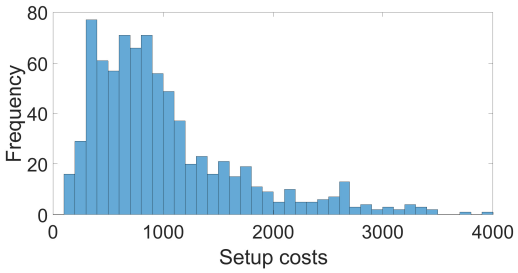
(c) Histogram of Demand of the large data set

Figure 2: Histograms of Demand of all three data sets

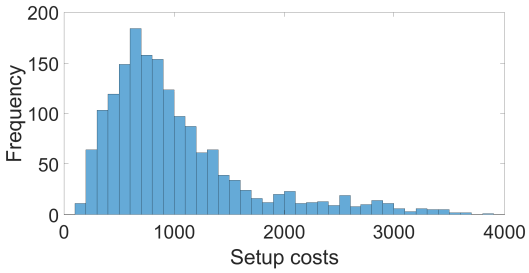
8.1.3 Costs

In this section, we provide histograms and scatterplots of the different costs aspects.

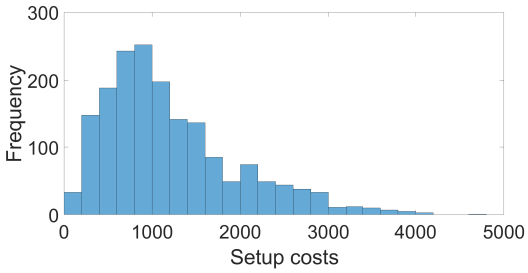
In almost every case, the setup costs are constant over time for a given CPU. Therefore, we show histograms in which only the first period is considered, instead of the setup costs of all periods. That indicates the more different setup costs. In all data sets, the range between low and high setup costs is substantial.



(a) Histogram of Setup costs of the small data set



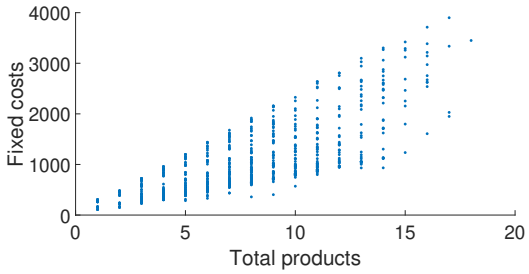
(b) Histogram of Setup costs of the medium data set



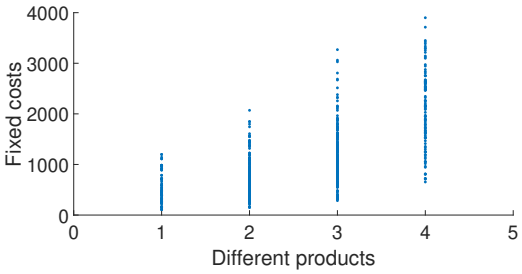
(c) Histogram of Variable costs of the large data set

Figure 3: Histograms of Setup costs of all three data sets

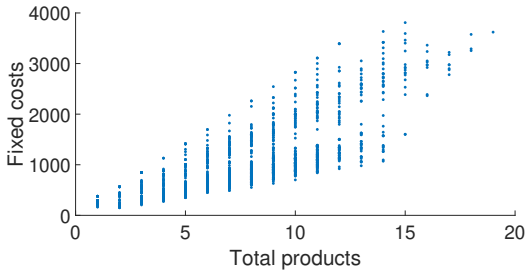
To determine if high setup costs indicate a CPU which produces multiple different products, we construct two different scatterplots. In the first scatterplot, the total number of products a CPU produces are counted. In the last scatterplot, the total number of unique products a CPU produces are counted. These scatterplots can be found below (Figure 4). In all figures, producing more products leads to higher setup costs. The same applies to unique products: the more unique products, the higher the setup costs. Hence, the setup costs and size of CPUs are correlated.



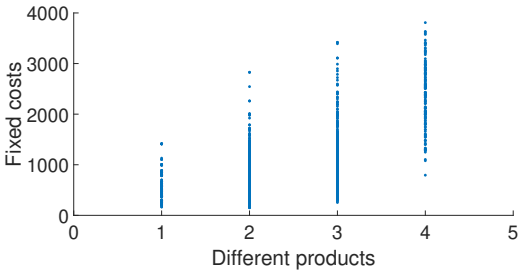
(a) Scatterplot of Setup costs and total products a CPU produces of the small data set



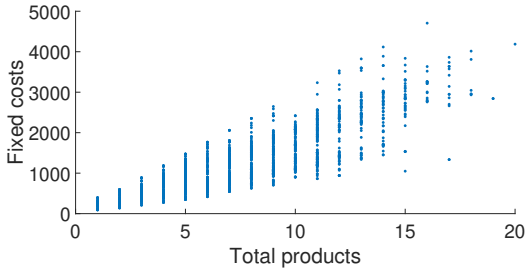
(b) Scatterplot of Setup costs and total unique products a CPU produces of the small data set



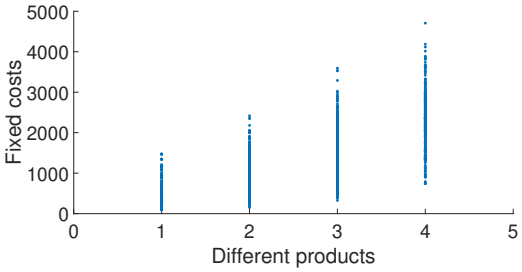
(c) Scatterplot of Setup costs and total products a CPU produces of the medium data set



(d) Scatterplot of Setup costs and total unique products a CPU produces of the medium data set



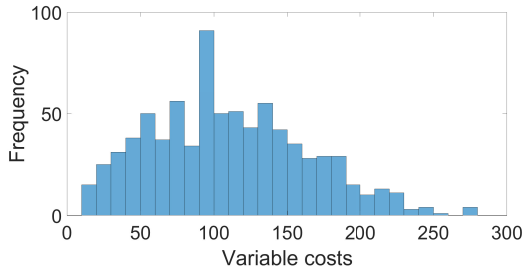
(e) Scatterplot of Setup costs and total products a CPU produces of the large data set



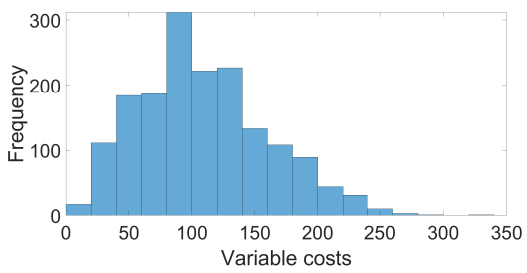
(f) Scatterplot of Setup costs and total unique products a CPU produces of the large data set

Figure 4: Scatterplot of Setup costs and CPUs of all three data sets

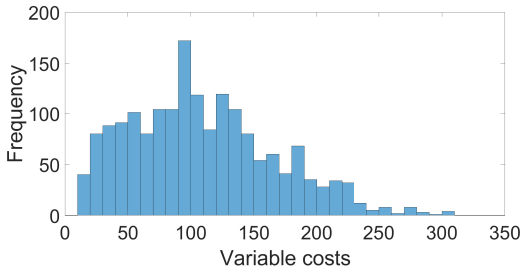
The variable costs are, in addition to the setup costs, constant over time. Therefore, we again only show histograms of the variable costs in the first period. The variable costs of the CPU cover a wide range, meaning that variable costs with low and high costs exists. High variable costs might indicate a CPU which produces multiple different products.



(a) Histogram of Variable costs of the small dataset



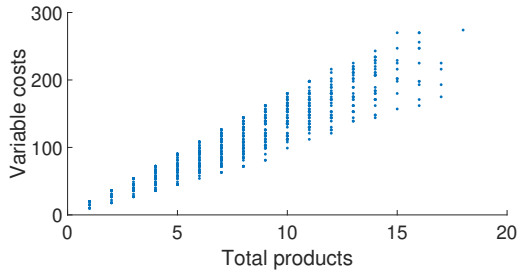
(b) Histogram of Variable costs of the medium dataset



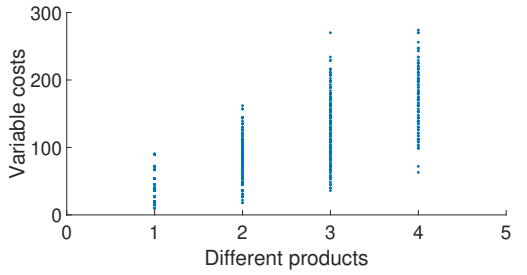
(c) Histogram of Variable costs for large data set

Figure 5: Histograms of Variable costs of all three data sets

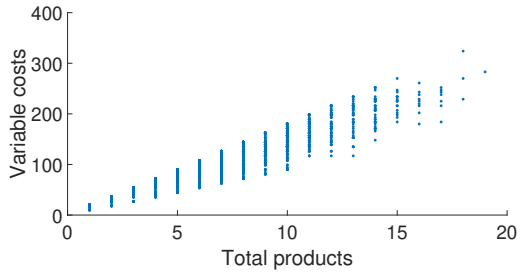
For the setup costs, we determined that setup costs and the size of CPUs are correlated. Now, we determine if this also holds for the variable costs. In the first scatterplot, the total number of products a CPU produces are counted. In the last scatterplot, the total number of unique products a CPU produces are counted. These scatterplots can be found below (Figure 6). In all figures, producing more products leads to higher variable costs. The same applies to unique products: the more unique products, the higher the variable costs. Hence, the higher the costs the larger the CPU.



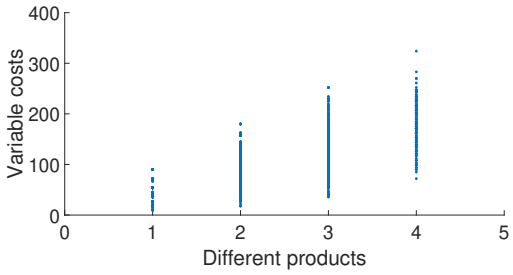
(a) Scatterplot of Variable costs and total products a CPU produces of the small data set



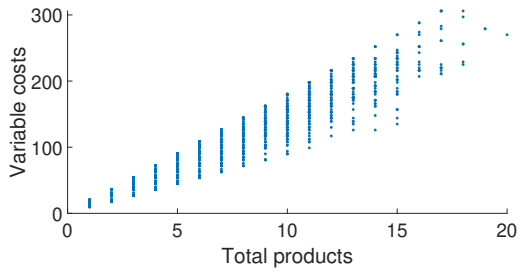
(b) Scatterplot of Variable costs and total unique products a CPU produces of the small data set



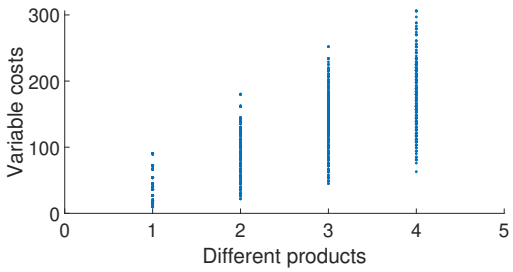
(c) Scatterplot of Variable costs and total products a CPU produces of the medium data set



(d) Scatterplot of Variable costs and total unique products a CPU produces of the medium data set



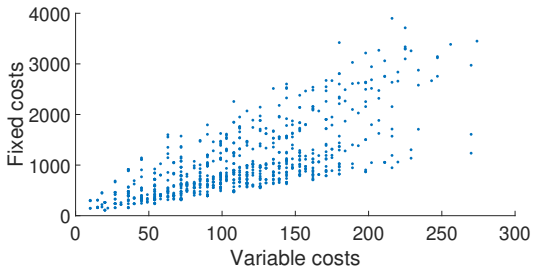
(e) Scatterplot of Variable costs and total products a CPU produces of the large data set



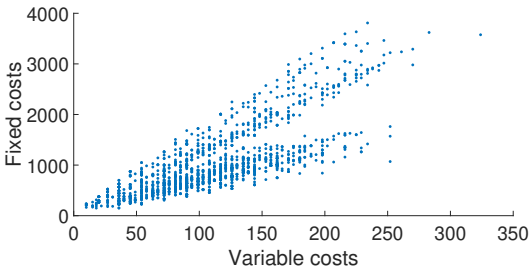
(f) Scatterplot of Variable costs and total unique products a CPU produces of the large data set

Figure 6: Scatterplot of Variable costs and CPUs of all three data sets

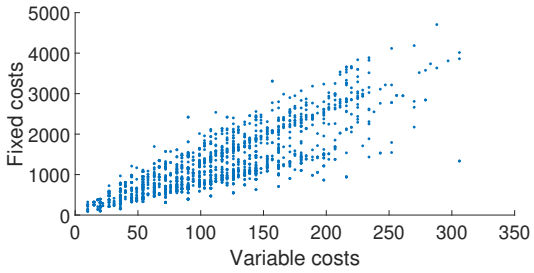
Next, we show scatterplots (Figure 7) of variable costs and setup costs. These scatterplots are made to verify if a CPU with high setup costs also has high variable costs. For all data sets, in general, high variable costs mean high setup costs. Hence, the variable and setup costs are correlated.



(a) Scatterplot of variable costs and fixed costs of small data set



(b) Scatterplot of variable costs and fixed costs of medium data set



(c) Scatterplot of variable costs and fixed costs of large data set

Figure 7: Scatterplots of variable costs and fixed costs of all three data sets

The holding costs are also constant over time. Therefore, we again only provide the holding costs of the first period. The holding costs of a product is always 1, 1.5 or 1.75, depending on the product. The histograms of the holding costs can be found in Appendix B.3, Figure 13.

8.1.4 Inventory limit

The inventory limit is not set. Therefore, we need to set the limit for every instance. If we set a low inventory limit, we have to produce (almost) every product in every period, leading to high setup costs and low holding costs. On the other side, if the limit is high, then the optimal solution has low setup costs and high holding costs. We try different capacity limits. Most solutions are expected feasible when the inventory limit is at least the maximum demand in one period. Therefore, we set the capacity limit as multiplication of a factor and the maximum demand in one period.

We tune the inventory limit using the Dantzig-Wolfe decomposition as given in Section 5. The lowerbounds are the objective values given by the LP relaxation. The feasible upperbounds are calculated using the method as given in Section 5.4.1. The maximum added columns in each iteration is set at $\frac{CT}{10}$. This means that only the pricing problems with the 10% most negative reduced costs can be added to the RMP in every iteration. We set the inventory limit to make this problem more difficult. Setting the inventory limit too low could result in producing in every period or could cause infeasibility. If the inventory limit is too high, it could result in producing once to cover demand for all periods.

We adjusted the means and standard deviations in the tables below. If an instance caused infeasibilty, due to the inventory limit, then all means and standard deviations of that instance are not taken into account. A graph of the the means, standard deviations and the adjusted values can be found in Appendix B.2, Figure 12. The computational results can be found in Appendix B.2, tables 14-19.

Table 2: Objective values of various capacity limits of small data set

\Capacity	1.0	1.5	2.0	2.5	3.0	3.5	4.0	4.5	5.0
Mean LB	461,336	459,666	458,657	458,045	457,215	456,598	456,180	456,143	456,140
Std LB	350,027	349,195	348,500	348,173	347,562	347,189	346,950	346,954	346,952
Mean UB	937,222	931,097	929,659	928,771	926,642	918,482	912,157	911,981	910,813
Std. UB	801,259	794,668	793,698	793,826	790,063	781,081	775,910	776,395	776,278

The costs of the lower- and upperbound start converging at 4 times the maximum demand. This means that the inventory limit does not have the correct purpose. Capacity limits at 1 and 1.5 times the maximum demand caused infeasible solutions for a few instances. Therefore, we set the inventory limit at 3 times the maximum demand for every instance for the small data set.

Table 3: Objective values of various capacity limits of medium data set

\Capacity	1.0	2.0	3.0	4.0	5.0	6.0	7.0	8.0	9.0	10.0
Mean LB	851,692	849,966	849,402	848,713	847,252	846,737	846,642	846,545	846,485	846,482
Std LB	179,607	178,444	178,453	178,314	178,312	178,402	178,357	178,264	178,219	178,216
Mean UB	1,746,980	1,741,443	1,730,716	1,730,992	1,727,153	1,715,405	1,708,292	1,708,210	1,704,987	1,704,882
Std. UB	407,636	408,499	401,024	402,902	401,121	391,904	387,941	389,556	391,357	391,544

The medium data set caused infeasibily for 2 different instances at 1 time the maximum demand. The objective values start converging at 6 times the maximum demand. The upperbounds decrease substantially after 5 times the maximum demand. Therefore, we set the inventory bound at 5 times the maximum demand. An inventory bound of 4 times the maximum demand is also viable, since the objective value of the lowerbound decreased in comparison with lower inventory limits.

Table 4: Objective values of various capacity limits of large data set

\Capacity	1.0	3.0	5.0	7.0	9.0	11.0	13.0	15.0	17.0	19.0
Mean LB	1,080,406	1,076,631	1,073,040	1,069,763	1,069,292	1,067,343	1,067,245	1,067,207	1,067,193	1,067,170
Std LB	18,128	16,359	14,674	11,325	11,027	9,983	9,960	9,949	9,948	9,946
Mean UB	2,641,333	2,627,759	2,625,732	2,589,178	2,590,689	2,564,770	2,563,371	2,564,825	2,566,305	2,565,561
Std. UB	482,596	478,202	474,142	448,488	450,808	432,195	431,639	433,015	434,351	433,931

The standard deviations of the lowerbound of the large data sets are low. These are low since the total number of products and periods are the same for all instances at the large data set. The standard deviations of the upperbounds, however, remain high. Two instances are infeasible for a capacity limit set at 1 time the maximum demand. The lowerbounds start converging at 11 times the maximum demand. At 7 and 9 times the maximum demand, the lower- and upperbounds are almost equal. At lower inventory bounds, the objective value decreases. Therefore, we decided to set the capacity limit at 5 times the maximum demand.

8.2 Parameter tuning

We tune the maximum added columns in every iterations for the Dantzig-Wolfe decomposition of Section 5. Since we have 3 data sets (small, medium and large), we tune it for every data set individually. This parameter is then used in all decompositions. All computational results can be found in Appendix A.2, tables 20-28.

When tuning the maximum added columns, we want the Dantzig-Wolfe to perform some iterations. If it only performed a few iterations, the RMP contains too many columns since most pricing problems have a solution with negative reduced costs in the first iterations. Then, some columns could become redundant in the last performed iterations. Since we want solutions fast, we set a time limit at 15 minutes. If the Dantzig-Wolfe does not find the optimal solution in 15 minutes, then too little columns are added in every iteration. We try on fixed numbers of columns and columns that are based of the total number of pricing problems that have to be solved. In these tables, we indicate values with an asterisk if at least one instance was not able to find the optimal solution within 15 minutes or if the solution time exceeded 15 minutes.

Table 5: Average solution time and number of iteration needed to solve DW for small data set

	1	10	50	100	200	500	25%	50%	75%	100%
Time (sec)	543.04*	335.32	70.2	37.79	22.86	14.64	12.75	12.45	11.85	3.76
Iterations	431.8*	139.5	29.7	16.3	9.8	5.7	6.4	4.8	4.7	2.00

For the small data set, we decided to use 50% of maximum pricing problems. This has been done,

since the number of iterations is slightly more than adding at most 75% of the maximum pricing problems. That means that 50% does not have a lot of redundant columns in the RMP. It is also slightly faster than 25%. The fixed number of columns added every iteration is not able to be faster than a percentage.

Table 6: Average solution time and number of iteration needed to solve DW for medium data set

	1	10	100	250	500	1000	25%	50%	75%	100%
Time (sec)	901.47*	903.73*	242.81	128.38	83.64	66.13	60.89	59.35	54.51	21.40
Iterations	335.20*	194.30*	34.90	16.00	9.70	7.20	6.40	5.60	4.60	2.00

In the medium data set, we need to add at least 100 columns in every iteration to find the optimal solution value within reasonable time. Adding all columns give the optimal solution always in 2 iterations, with fast solving times. Adding 25% or 50% of the total pricing problems results in almost the same solution time. Adding less columns in every iteration, increases the solving time. Therefore, we set the maximum added columns in every iteration for the medium data set at 50%.

Table 7: Average solution time and number of iteration needed to solve DW for large data set

	1	10	100	250	500	1000	25%	50%	75%	100%
Time (sec)	902.77*	905.18*	625.73*	264.10	183.43	141.08	134.93	128.00	132.87	852.97*
Iterations	221.00*	147.00*	45.00*	20.20	12.20	8.00	6.80	5.80	5.10	2.00*

In the large data set, in contrast to the other data sets, adding all columns does not always find the optimal solution within 15 minutes. Adding 50% of the maximum pricing problems give the optimal solution value in the shortest amount of time. The optimal solution value is not obtained within 15 minutes for all instances if 100 columns or less are added in every iteration. Therefore, we set the maximum added columns in every iteration for the large data set at 50%.

8.3 Optimality gaps

In the tables that can be found below, we include the average and standard deviation of the most important results. In all tables we use the same abbreviations: LB means the lowerbound found in the Dantzig-Wolfe decomposition of Section 5, UB means the corresponding upperbound as calculated in Section 5.4.1. DW+Cplex means the found upperbound after performing this decomposition and then solving the problem using CPLEX (Section 5.4.2) . UBheur means the found objective value using the proposed heuristic after processing Dantzig-Wolfe (Section 5.4.3). LB cplex means the best found lowerbound using the commercial solver CPLEX, UB cplex the best found upperbound. Dec2 means the best found objective value of the decomposition as described in Section 6. Next, we also calculate the average gap between the found lowerbound using the Dantzig-Wolfe LP relaxation and the other upperbounds. In addition to this, we also calculate the gap between the best found lowerbound and the upperbounds. The maximum running time is 3600 seconds (1 hour). If the optimal solution is not found at that time, then the best solution is obtained. All computational results can be found in Appendix A.3, tables 29-40.

Table 8: Optimality gaps of small data set

	LB	UB	DW+Cplex	UBheur	LB cplex	UB cplex	Dec2
Mean	489,196.9	925,460.4	605,343.19	645,520.71	580,653.15	605,016.83	902,686.27
Std.	347,369.01	715,866.9	430,782.7	458,069.4	410,733.7	431,755	692,078
Gap LB (%)	-	78.40	23.08	32.34	-	23.08	74.61
Std. gap LB	-	40.52	5.73	7.08	-	5.72	36.71
Gap best LB (%)	-	49.23	11.10	11.17	-	3.30	46.11
Std. gap best LB	-	30.61	2.07	4.67	-	2.07	27.49
Time (sec)	12.45	12.45	2151.54	144.55	2190.39	2190.39	3.62

In Table 8, the objective value of the LP relaxation are much lower than the objective values found using CPLEX. This means that the optimality gaps, in comparison with the found lowerbound, are high. The objective values found using Dec2 are slightly lower than the objective values found using UB. On average, the lowerbound found using cplex is 18.70% higher. When comparing optimality gaps found using the best found lowerbound of CPLEX, the optimality gaps are reasonable. The heuristic (UBheur) managed to find a better solution than cplex in one out of ten instances. Next, the heuristic is fast, with only 144.55 seconds needed to solve. The decomposition based on fixing setup costs in every iteration (Dec2), was not able to find a solution close to the optimum as the others. The objective value found using Dantzig-Wolfe and CPLEX (DW+Cplex) is approximately equal to the objective value found using cplex only (UB cplex). The standard deviations, however, are large for UB and Dec2, but are low for DW+Cplex, UBheur and UB cplex. This means that the found optimality gaps are confidence for DW+Cplex, UBheur and UB cplex.

Table 9: Optimality gaps of medium data set

	LB	UB	DW+Cplex	UBheur	LB cplex	UB cplex	Dec2
Mean	889,334.9	1,769,848	1,106,732	1,229,284	1,049,267	1,101,894	1,776,519
Std.	236,185.9	366,602.3	267,302.9	351,963	264,022.7	278,468	347,668.9
Gap LB (%)	-	105.424	25.06	37.56	-	24.19	106.69
Std. gap LB	-	49.54	4.95	8.41	-	4.34	49.49
Gap best LB (%)	-	72.59	5.68	16.28	-	4.97	73.66
Std. gap best LB	-	35.52	1.20	6.40	-	1.67	35.31
Time (sec)	60.89	60.89	3610.68	486.42	3601.01	3601.01	8.23

For the medium data set (Table 9), the objective value of UB is much higher than the corresponding LB, resulting in a large optimality gap. The best objective value found using Dec2 are almost the same as the objective values found using UB, and those are therefore not satisfactory. The found objective values of the heuristic (UBheur) are acceptable, especially when compared with the best found lowerbound. These objective values are obtained fast. The standard deviation of the heuristic is again low, meaning that the found optimality gaps are low. The objective values found using a commercial solver are again lower than the objective values of the heuristic. The commercial solver, however, was not able to find the optimal solution in an instance.

Table 10: Optimality gaps of large data set

	LB	UB	DW+Cplex	UBheur	LB cplex	UB cplex	Dec2
Mean	1,098,873	2,511,908	1,417,042	1,678,653	1,325,645	1,402,569	2,526,609
Std.	64,370.67	483,222	76,490.99	214,568.5	82,336.84	74,941.17	428,813.4
Gap LB (%)	-	129.66	29.06	53.26	-	27.72	129.81
Std. gap LB	-	47.80	5.10	21.74	-	4.03	36.22
Gap best LB (%)	-	128.59	6.98	26.91	-	5.86	90.41
Std. gap best LB	-	89.49	3.10	16.55	-	1.68	28.68
Time (sec)	134.92	134.92	3608.41	1141.9	3603.20	3606.67	11.54

The optimality gaps of the large instance can be found in the table above (Table 10). The optimality gaps of the found upperbound (UB) and the objective value of Dec2 deteriorate in comparison with the other data sets. These methods do not give satisfactory objective values in all instances. The objective value of the heuristic (UBheur) is on average higher than the small and medium data set. The optimality gaps remain acceptable along with fast solving times. The standard deviation, however, deteriorate. At the end, it does perform well in all instances. The commercial solver does manage to find the lowest objective valus, with a low optimaltiy gap. However, those optimality gaps are found after 1 hour running time.

8.3.1 Branch-and-price

In Section 5.5, we proposed a branch-and-price algorithm to increase the lowerbound and decrease the upperbound. No instance was able to find the optimal solution within 1 hour. The objective value decreased by $9.28\%, 5.61\%$ and 1.80% for the small, medium and large data set respectively. Recall that the upperbounds are calculated by rounding up all setup costs. We do not have results about the objective value when performing the heuristic after branch-and-price. The small data set performed, on average, 4860 iterations, while the medium and large data set performed 64.9 and 36.5 iterations respectively. The optimality gaps of the medium data set decreased the most per performed iteration of branch-and-price.

Table 11: Improved Gaps using branch-and-price

	New Gap	Std. New Gap	LB increase	UB decrease	Av. Improv.	Std. Improv.	Av. Iterations
Small	71.92	1.34	2.41	39.97	9.28	8.21	4860
Medium	100.46	0.93	1.65	50.08	5.61	3.81	64.90
Large	127.14	0.19	0.82	46.39	1.80	1.55	36.50

8.4 Costs distribution

We also compare the influence of every costs aspect: setup, variable and holding costs. This is done since it provides more insight in the obtained solutions. Another thing is that it indicates limitations of different methods. Next, the costs distribution give an overview of all costs. It also shows how to decrease the total costs overall. In the table below, we present the different distributions of all data sets. It is possible that the percentages do not add up to 100%, this is due to rounding.

Table 12: Costs distribution of all data set

		UB	DW+Cplex	UBheur	UB cplex	Dec2
Small	Mean Setup (%)	44.52	14.36	14.01	14.40	42.23
	Std. Setup	10.89	3.60	5.16	2.89	9.21
	Mean Variable (%)	52.69	73.85	68.96	74.35	54.05
	Std. Variable	9.48	3.96	4.99	3.47	8.42
	Mean Holding (%)	2.79	11.79	17.04	11.24	3.72
	Std. Holding	2.04	1.72	4.62	1.65	1.99
Medium	Mean Setup (%)	49.50	11.98	12.37	12.17	47.67
	Std. Setup	12.0	0.98	5.73	1.18	13.63
	Mean Variable (%)	48.63	78.03	70.38	78.51	46.69
	Std. Variable	9.86	3.40	5.11	3.43	11.24
	Mean Holding (%)	1.86	9.99	17.24	9.32	5.64
	Std. Holding	3.60	3.35	2.44	3.38	10.45
Large	Mean Setup (%)	55.27	13.83	22.24	13.87	56.63
	Std. Setup	10.58	1.84	12.46	1.29	7.59
	Mean Variable (%)	43.47	74.43	63.43	75.58	42.56
	Std. Variable	9.10	3.49	9.84	2.69	7.38
	Mean Holding (%)	1.26	11.75	14.33	10.56	0.82
	Std. Holding	2.24	2.24	3.78	2.04	0.89

The costs distribution of a method are approximately constant for different data sets. The costs distribution when compared with other methods are different. When comparing every method, the solutions of cplex have around equal setup costs. The setup costs of the heuristic (UBheur) is relatively the same in comparison with cplex, except for the large data set. The setup costs of UB and Dec2 are much higher. That implies the high objective values. The variable costs of the solutions of cplex are also around the same, but the average variable costs of the heuristic is signifcantly lower. That could be the case, because we are limited to only produce a few CPUs. This implies that we overproduce some products, leading to higher holding costs. Increased holding costs also leads to lower variable costs. The variable costs and holding costs of the second decomposition are much lower, due to the fact that we have much higher setup costs. The holding costs of the two cplex are again almost equal. The standard deviations are low, except for the second decomposition in the medium data set and the heuristic in the large data set. These low standard deviations indicate that the instances have a stable costs distribution among these methods.

8.5 Costs decrease in optimise part heuristic

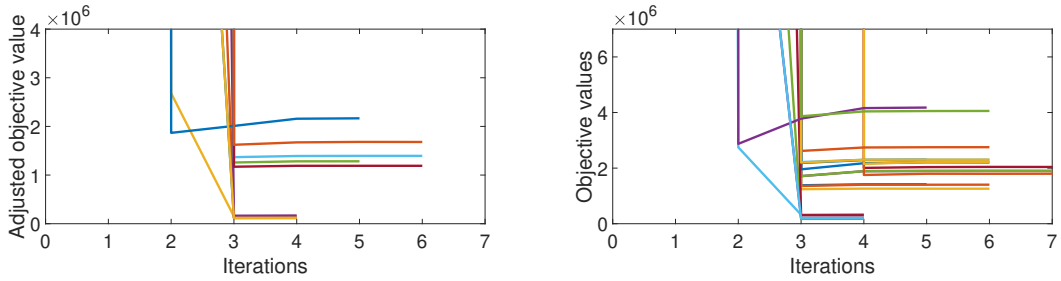
In Section 5.4.3, we proposed a heuristic inspired by relax & fix and fix & optimise. After a feasible solution has been found, we start optimising the found solution. The results of the optimisation part can be found in the table below, Table 13. The optimising part proved to be successful for all different data sets. On average, in more than half of the instances, the obtained objective value decreased. When the objective value decreases due to optimising, the objective value is decreased by over 10% for all data sets. Therefore, this optimising part is effective.

Table 13: Optimisation part heuristic

	Small	Medium	Large
Instances	6/10	8/10	4/10
Mean costs decrease (%)	13.00	11.86	11.10
Std. costs decrease	8.73	6.96	3.72

8.6 Iterations of Decomposition without predetermined production

The objective values of the decomposition without predetermined production (Section 6) do not converge to a certain value. Instead, they decrease fast and after reaching a certain value, it starts increasing. Therefore, we do not take the last objective value, but keep track of the best objective value. In the figures below, adjusted objective value means that unused setup costs are subtracted from the objective value. Objective value means the objective value attained by solving the RMP as described in Section 6.

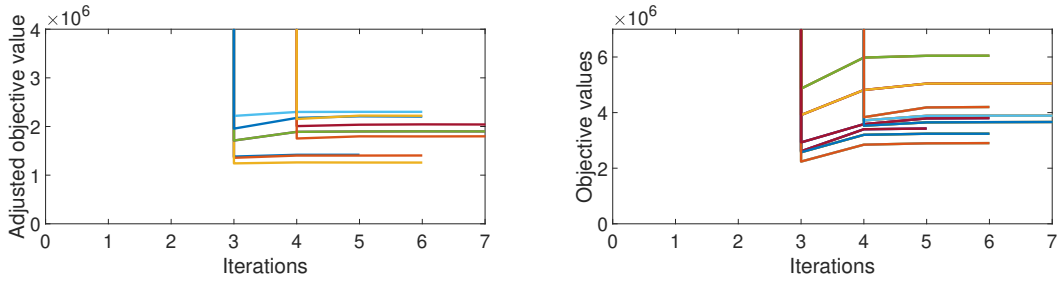


(a) Objective values found in every iteration with subtracting setup costs

(b) Objective values found in every iteration without subtracting setup costs

Figure 8: Objective values found in every iteration for the second decomposition, with and without, subtracting setup costs of unused CPUs of small data set

If a setup costs of a CPU has been payed, but that CPU has not been used, we subtract those setup costs of the objective value in Figure 8a. In Figure 8b, we do not subtract those setup costs. The interesting part is when we obtain the lowest objective values. That is almost always in the third iteration, even if unused setup costs are not subtracted from the objective value. This could occur, because we add at most 50% of all subproblems in the small data set. If we add two times at most 50%, after the third, we added at most 100% of all subproblems. This is clearly not the case, since we have at least 4 iterations, before the procedure terminates.

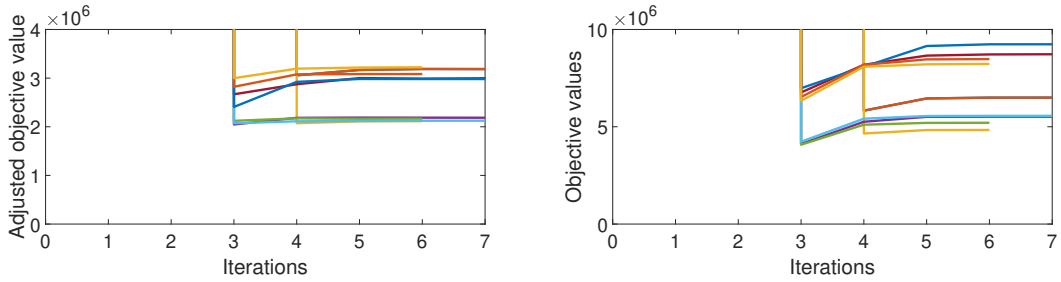


(a) Objective values found in every iteration with subtracting setup costs

(b) Objective values found in every iteration without subtracting setup costs

Figure 9: Objective values found in every iteration for the second decomposition, with and without, subtracting setup costs of unused CPUs of medium data set

The figure above (Figure 9) presents the objective values of the medium data set. In the medium data set, we add at most 25% of the total number of PPs to the RMP. In the third or fourth iteration, the lowest objective values are obtained. That indicates that at most 75% or 100% of the total number of PPs are added to the RMP. Since the procedue does not terminate after the fourth iteration, the maximum added columns is not attained in every iteration.



(a) Objective values found in every iteration with subtracting setup costs

(b) Objective values found in every iteration without subtracting setup costs

Figure 10: Objective values found in every iteration for the second decomposition, with and without, subtracting setup costs of unused CPUs of large data set

The figure above (Figure 10) presents the objective values of the large data set. In the large data set, we also add 25% of the total number of PPs to the RMP. For the large data set, the lowest objective values are also obtained at the thrid or fourth iteration. Most of the instances terminate after the sixth or seventh iteration. This again suggest that the maximum of 25% added columns is not attained in every iteration.

9 Conclusion

We are interested in solving the uncapacitated lot-sizing problem with inventory bounds and CPUs. This problem is hard to solve to optimality, we proved that this problem is strongly $\mathcal{NP}$-hard. This means that this problem is one of the hardest problems to solve. In addition to this, we also proved that this problem does, in general, not satisfy the (generalised) zero-inventory property. These two elements indicate the complexity of this problem. Therefore, we have valid reasons to use heuristics instead.

A first approach is a math programming heuristic (Section 5). We apply a Dantzig-Wolfe decomposition to our problem and after that we perform a heuristic, based on the solutions of this decomposition. In this way, we obtain a lowerbound to our problem, which we can use to compare with any other found feasible solutions. After finding this lowerbound, we try to obtain a feasible solution using different methods. A feasible solution is a solution which satisfy all constraints. To be more precise, the binary setup costs variable is a hard constraint. The first approach to find a solution, satisfying all constraints, is simple, round every setup costs up. This leads to high costs overall, since some setup costs were used for a small amount. A second approach is to take the selected columns of the last iteration of the RMP and solve the problem using a commercial solver, to find the best solution among these columns within reasonable time. That leads to good solutions, with a low optimality gap. However, we prefer a heuristic without the need of a commercial solver instead. Therefore, we come up with another solution method. The last heuristic part is inspired by relax and fix & fix and optimise. It is not strictly this heuristic, but it ressembles it. The obtained solutions have low optimality gaps and are obtained fast. In one of the cases, the obtained solution is even better than those obtained by a commercial solver.

After that, we changed the way in which columns are added to the RMP (Section 6). In this approach, we add the setup costs to the problem and ensure that the corresponding CPU is allowed to produce. We do not predetermine how much it could produce. Since this approach does not have a mathematical background, we do not guarantee convergence. Therefore, we are interested in the obtained objective values in every iteration. This resulted in obtained objective values that are not converging.

Next, we proposed a Dantzig-Wolfe decomposition in which multiple CPUs can be produced in one added column. This leads to less pricing problems that need to be solved in every iteration, but those pricing problems are harder to solve. These pricing problems are knapsack problems with setup costs. Those problems can be solved using a branch-and-bound algorithm. This decomposition, however, is not implemented in this thesis. Therefore, we can not provide any results. This method, however, could be further explored.

Concluding, the proposed math programming heuristic with a heuristic inspired by relax and fix & fix and optimise provided solid results, comparing with the best found lowerbound using a commercial solver. The feasible solution is in some cases even better than those found using a commercial solver, while finding these solutions in a much shorter time period. The lowerbound found using a Dantzig-Wolfe decomposition remains low, even when branch-and-price is applied. The optimality gap only improved by around 9% for small instances and 2% for large instances. Therfore, high optimality gaps remain.

9.1 Further Research

For further research, other problems with CPUs can be solved, for instance solving the capacitated lot sizing problem with CPUs. This problem is similar to the problem solved in this thesis, but then with capacities on producing. The LP-relaxation of the problem can be solved by a Dantzig-Wolfe decomposition, in particular the decomposition which is similar to the one given in Section 7. Other research can be done by improving the math programming heuristic. For instance, choosing specific CPUs that can be produced in every period, and via a genetic algorithm, improving the solution values. Research can also be done about improving the branch-and-price algorithm. An improvement could be the way that upperbounds are calculated. Instead of finding the upperbound by rounding all setup costs up, use a fast heuristic to find an upperbound (for instance, the heuristic

as proposed in Section 5.4.3). Branch-and-price can also be changed to a Branch-and-price-and-cut algorithm, adding cuts in every iteration. Before solving the problem using branch-and-price-and-cut, feasible cuts need to be constructed. Therefore, it is a good suggestion to start by solving the problem using branch-and-cut. After that has been done, the algorithm could be expanded towards a branch-and-price-and-cut algorithm.

References

S. Ağralı. A dynamic uncapacitated lot-sizing problem with co-production. *Optimization Letters*, 6(6):1051–1061, 2012.

A. Akbalik, B. Penz, and C. Rapine. Multi-item uncapacitated lot sizing problem with inventory bounds. *Optimization Letters*, 9(1):143–154, 2015.

I. Barany, T. Van Roy, and L. A. Wolsey. Uncapacitated lot-sizing: The convex hull of solutions. In *Mathematical programming at Oberwolfach II*, pages 32–43. Springer, 1984.

N. Brahimi, N. Absi, S. Dauzère-Pérès, and A. Nordli. Single-item dynamic lot-sizing problems: An updated survey. *European Journal of Operational Research*, 263(3):838–863, 2017.

M. Desrochers. *An algorithm for the shortest path problem with resource constraints*. École des hautes études commerciales, Groupe d'études et de recherche en . . . , 1988.

G. D. Eppen and R. K. Martin. Solving multi-item capacitated lot-sizing problems using variable redefinition. *Operations Research*, 35(6):832–848, 1987.

J. Gutiérrez, A. Sedeño-Noda, M. Colebrook, and J. Sicilia. A new characterization for the dynamic lot size problem with bounded inventory. *Computers & Operations Research*, 30(3):383–395, 2003.

D. S. Johnson. The np-completeness column: an ongoing guide. *Journal of Algorithms*, 6(3): 434–451, 1985.

J.-C. Lange. *" Design and management of networks with fixed transportation costs*. PhD thesis, UCL-Université Catholique de Louvain, 2010.

M. Loparic, H. Marchand, and L. A. Wolsey. Dynamic knapsack sets and capacitated lot-sizing. *Mathematical Programming*, 95(1):53–69, 2003.

V. Lotfi and Y.-S. Yoon. An algorithm for the single-item capacitated lot-sizing problem with concave production and holding costs. *Journal of the Operational Research Society*, 45(8):934–941, 1994.

S. F. Love. Bounded production and inventory models with piecewise concave costs. *Management Science*, 20(3):313–318, 1973.

L. Lu and X. Qi. Dynamic lot sizing for multiple products with a new joint replenishment model. *European Journal of Operational Research*, 212(1):74–80, 2011.

R. A. Melo and C. C. Ribeiro. Formulations and heuristics for the multi-item uncapacitated lot-sizing problem with inventory bounds. *International Journal of Production Research*, 55(2): 576–592, 2017.

S. Michel, N. Perrot, and F. Vanderbeck. Knapsack problems with setups. *European Journal of Operational Research*, 196(3):909–918, 2009.

S. Öner and T. Bilgiç. Economic lot scheduling with uncontrolled co-production. *European Journal of Operational Research*, 188(3):793–810, 2008.

B. Pamuk. A lot sizing problem in deliberated and controlled co-production systems. Master's thesis, Boğaziçi University, 2016.

C. M. O. Pimentel, F. P. e. Alvelos, and J. M. Valerio de Carvalho. Comparing dantzig–wolfe decompositions and branch-and-price algorithms for the multi-item capacitated lot sizing problem. *Optimization Methods & Software*, 25(2):299–319, 2010.

Y. Pochet and L. A. Wolsey. Polyhedra for lot-sizing with wagner—whitin costs. *Mathematical Programming*, 67(1-3):297–323, 1994.

Y. Pochet and L. A. Wolsey. *Production planning by mixed integer programming*. Springer Science & Business Media, 2006.

E. Toczylowski. An o (t/sup 2/) algorithm for the lot-sizing problem with limited inventory levels. In *Proceedings 1995 INRIA/IEEE Symposium on Emerging Technologies and Factory Automation. ETFA'95*, volume 3, pages 77–85. IEEE, 1995.

A. Wagelmans, S. Van Hoesel, and A. Kolen. Economic lot sizing: an o (n log n) algorithm that runs in linear time in the wagner-whitin case. *Operations Research*, 40(1-supplement-1): S145–S156, 1992.

H. M. Wagner and T. M. Whitin. Dynamic version of the economic lot size model. *Management science*, 5(1):89–96, 1958.

A Additional tables

A.1 Inventory limits

On the next pages, the obtained objective values can be found. LB means the found objective value of the LP relaxation. UB means the found value when implementing the heuristic as explained in Section 5.4.1.

Table 14: Obtained lowerbounds for various capacity limits of the small data set

Small	**LB**								
Instance\Cap	**1.0**	**1.5**	**2.0**	**2.5**	**3.0**	**3.5**	**4.0**	**4.5**	**5.0**
1	91,955.74	91,554.57	91,110.77	90,795.06	90,732.94	90,215.02	90,081.82	90,081.82	90,081.82
2	83,914.57	83,477.13	83,390.13	83,268.63	83,017.84	83,009.22	82,939.50	82,918.95	82,918.95
3	86,266.86	84,772.17	84,515.96	84,185.74	83,906.09	837,18.42	83,435.14	83,332.44	83,332.44
4	infeas	95,370.35	94,346.64	94,035.71	91,019.69	91,002.43	90,744.68	90,662.98	90,604.42
5	infeas	infeas	835,320.50	832,733.06	831,951.85	829,273.78	826,718.29	825,621.08	824,953.92
6	726,586.54	722,453.32	719,606.60	719,601.69	718,566.88	717,765.57	717,120.10	717,022.16	717,006.71
7	736,695.66	736,359.52	735,332.61	734,699.24	733,756.90	733,616.33	732,443.37	732,443.37	732,443.37
8	737,192.18	735,343.26	734,866.73	731,990.40	731,467.37	731,061.15	731,019.49	731,019.49	731,019.49
9	766,737.58	763,706.40	761,776.99	761,772.95	759,055.05	756,797.18	756,218.17	756,179.63	756,179.63
10	infeas	778,058.48	776,897.37	772,921.19	768,493.90	767,661.89	766,396.81	766,282.84	766,282.84

Table 15: Obtained upperbounds for various capacity limits of the small data set

Small	**UB**								
Instance\Cap	**1.0**	**1.5**	**2.0**	**2.5**	**3.0**	**3.5**	**4.0**	**4.5**	**5.0**
1	122,750.61	122,182.38	121,616.40	120,515.88	120,609.48	120,298.76	116,208.82	1162,08.82	116,208.82
2	116,002.96	115,902.96	115,906.62	115,998.51	115,984.13	115,997.41	115,767.91	115,567.91	113,286.31
3	154,190.36	150,659.59	151,056.39	149,509.47	147,735.17	149,571.78	147,136.56	147,536.56	147,536.56
4	infeas	169,636.36	167,856.70	166,717.45	163,768.94	161,706.06	161,595.72	161,100.02	161,081.40
5	infeas	infeas	1,274,597.21	1,268,244.39	1,263,492.72	1,268,234.91	1,261,368.96	1,263,153.92	1,258,840.86
6	1,328,380.27	1,324,057.65	1,325,196.69	1,324,001.70	1,325,702.99	1,321,952.71	1,327,955.85	1,315,017.81	1,309,123.47
7	1,170,940.93	1,167,803.98	1,168,121.06	1,166,001.51	1,166,061.85	1,155,620.00	1,145,412.38	1,145,412.38	1,145,412.38
8	2,039,838.67	2,012,797.23	2,019,011.56	2,018,537.11	1,994,066.59	1,976,011.19	1,957,024.69	1,957,024.69	1,957,024.69
9	1,628,450.45	1,624,275.88	1,616,703.00	1,616,830.85	1,616,335.17	1,589,919.69	1,575,592.73	1,587,099.43	1,587,099.43
10	infeas	1,384,817.81	1,381,099.75	1,332,565.21	1,340,846.81	1,282,228.40	1,274,079.43	1,275,911.31	1,275,911.31

Table 16: Obtained lowerbounds for various capacity limits of the medium data set

Medium	**LB**									
Instance\Cap	**1.0**	**2.0**	**3.0**	**4.0**	**5.0**	**6.0**	**7.0**	**8.0**	**9.0**	**10.0**
1	716,361.96	715,054.69	714,631.64	714,050.98	713,818.91	713,612.79	713,563.87	713,508.75	713,504.71	713,504.71
2	717,077.20	716,946.64	716,550.19	715,319.06	714,821.41	714,634.24	7145,54.09	714,545.49	714,441.05	714,441.05
3	716,675.66	715,363.84	714,750.56	714,512.53	713,598.36	713,388.64	713,320.79	713,308.80	713,291.08	713,291.08
4	723,663.85	722,800.59	722,336.65	722,033.09	719,464.57	719,152.87	71,9054.76	719,005.97	719,005.97	719,005.97
5	infeas	738,233.46	737,057.13	737,053.83	734,674.74	733,976.47	733,967.24	732,865.12	732,865.12	732,865.12
6	734,346.16	733,545.61	732,571.47	731,989.59	728,800.75	726,758.43	726,744.05	726,724.17	726,714.69	726,710.10
7	infeas	1,404,811.27	1,395,784.40	1,389,489.02	1,380,660.67	1,377,627.63	1,377,545.96	1,377,481.27	1,377,183.47	1,377,104.65
8	1,067,428.53	1,064,609.15	1,063,925.03	1,062,925.23	1,062,761.82	1,062,312.69	1,062,132.62	1,062,032.62	1,062,032.62	1,062,032.62
9	1,070,373.43	1,065,570.12	1,065,320.54	1,064,273.64	1,062,102.45	1,061,507.01	1,061,382.72	1,060,882.82	1,060,538.34	1,060,520.52
10	1,067,609.26	1,065,838.82	1,065,130.09	1,064,603.10	1,062,645.26	1,062,526.39	1,062,382.60	1,062,350.28	1,062,350.28	1,062,350.28

Table 17: Obtained upperbounds for various capacity limits of the medium data set

Medium	**UB**									
Instance\Cap	**1.0**	**2.0**	**3.0**	**4.0**	**5.0**	**6.0**	**7.0**	**8.0**	**9.0**	**10.0**
1	1,3258,44.99	1,321,426.74	1,320,574.45	1,319,843.13	1,319,450.14	1,319,399.09	1,318,217.80	1,314,119.61	1,316,114.97	1,316,114.97
2	1,297,288.53	1,295,941.92	1,292,479.94	1,294,103.73	1,294,013.62	1,291,206.29	1,288,039.81	1,287,330.81	1,287,330.81	1,287,330.81
3	1,239,656.12	1,236,511.97	1,221,497.96	1,221,915.14	1,214,136.14	1,212,495.29	1,212,023.60	1,210,668.83	1,197,648.75	1,197,648.75
4	1,864,634.71	1,852,557.84	1,856,522.61	1,843,120.53	1,843,520.13	1,809,404.62	1,789,077.74	1,790,858.50	1,790,858.50	1,790,858.50
5	infeas	1,885,783.82	1,868,888.65	1,870,746.47	1,868,608.08	1,852,120.27	1,827,661.74	1,798,252.23	1,803,511.52	1,799,153.76
6	2,261,712.96	2,269,445.94	2,211,458.79	2,228,055.78	2,209,051.76	2,161,225.36	2,148,024.35	2,154,044.94	2,153,364.82	2,154,652.82
7	infeas	2,057,634.90	2,0348,10.48	2,006,787.98	2,012,645.65	1,991,997.24	1,996,320.20	2,005,764.42	2,009,487.16	2,011,940.84
8	2,093,568.55	2,083,709.37	2,080,529.93	2,079,259.18	2,079,410.60	2,075,783.61	2,064,311.46	2,064,311.46	2,064,311.46	2,064,311.46
9	1,792,246.51	1,771,209.19	1,763,452.30	1,762,831.16	1,758,412.60	1,755,770.75	1,751,095.01	1,751,423.66	1,737,344.14	1,735,209.89
10	2,100,889.33	2,100,740.63	2,099,210.69	2,098,804.14	2,099,228.15	2,097,954.87	2,095,542.83	2,092,925.24	2,092,925.24	2,092,925.24

Table 18: Obtained lowerbounds for various capacity limits of the large data set

Large	**LB**									
Instance\Cap	**1.0**	**3.0**	**5.0**	**7.0**	**9.0**	**11.0**	**13.0**	**15.0**	**17.0**	**19.0**
1	infeas	1,290,423.47	1,269,624.52	1,265,482.32	1,263,361.04	1,263,078.43	1,263,078.43	1,263,078.43	1,263,078.43	1,263,078.43
2	1,094,109.58	1,088,925.40	1,086,055.32	1,079,347.32	1,079,001.48	1,068,882.84	1,068,241.54	1,068,001.32	1,067,890.61	1,067,705.59
3	infeas	1,160,917.07	1,134,789.52	1,130,152.63	1,129,480.78	1,129,304.13	1,129,304.13	1,129,304.13	1,129,304.13	1,129,304.13
4	1,060,644.54	1,059,263.76	1,058,038.36	1,057,789.74	1,057,753.46	1,057,742.46	1057740.30	1,057,740.30	1,057,740.30	1,057,740.30
5	1,058,121.26	1,056,308.58	1,053,729.81	1,053,476.35	1,053,476.35	1,053,476.35	1053476.35	1,053,476.35	1,053,476.35	1,053,476.35
6	1,061,871.28	1,059,943.23	1,058,163.47	1,058,109.22	1,057,998.92	1,057,998.92	1057998.92	1,057,998.92	1,057,998.92	1,057,998.92
7	1,107,915.83	1,101,796.92	1,093,988.31	1,081,543.13	1,080,574.98	1,079,911.48	1079911.48	1,079,911.48	1,079,911.48	1,079,911.48
8	1,087,022.93	1,081,897.12	1,077,376.80	1,076,292.90	1,075,565.34	1,075,002.95	1074955.43	1,074,902.92	1,074,902.92	1,074,902.92
9	1,086,833.06	1,082,667.61	1,079,429.52	1,077,749.88	1,077,788.44	1,077,650.08	1077616.74	1,077,602.22	1,077,602.22	1,077,602.22
10	1,086,727.98	1,082,245.31	1,077,536.53	1,073,799.02	1,072,276.76	1,068,076.62	1,068,020.82	1,068,020.71	1,068,020.71	1,068,020.71

Table 19: Obtained upperbounds for various capacity limits of the large data set

Large	**UB**									
Instance\Cap	**1.0**	**3.0**	**5.0**	**7.0**	**9.0**	**11.0**	**13.0**	**15.0**	**17.0**	**19.0**
1	infeas	2,122,701.97	2,125,958.11	2,117,300.53	2,115,882.53	2,118,727.33	2,118,727.33	2,118,727.33	2,118,727.33	2,118,727.33
2	3,081,542.52	3,077,605.35	3,063,958.04	2,832,997.50	2,952,362.18	2,833,557.14	2,829,092.37	2,831,976.89	2,844,820.97	2,839,308.59
3	infeas	2,001,963.47	1,987,264.11	1,943,850.97	1,942,587.36	1,941,403.71	1,941,403.71	1,941,403.71	1,941,403.71	1,941,403.71
4	2,069,712.61	2,061,891.46	2,067,002.03	2,066,745.81	2,065,890.91	2,060,612.39	2,058,214.23	2,058,214.23	2,058,214.23	2,058,214.23
5	2,077,562.92	2,076,193.02	2,079,396.35	2,068,681.90	2,068,681.90	2,068,681.90	2,068,681.90	2,068,681.90	2,068,681.90	2,068,681.90
6	2,038,483.74	2,023,950.35	2,022,683.52	2,014,477.44	2,014,895.09	2,014,895.09	2,014,895.09	2,014,339.69	2,013,339.69	2,012,895.09
7	2,893,439.33	2,883,777.93	2,889,135.08	2,888,837.52	2,801,238.64	2,761,837.46	2,761,837.46	2,761,837.46	2,761,837.46	2,761,837.46
8	2,997,055.67	2,976,264.82	2,956,221.71	2,944,830.11	2,933,188.93	2,931,506.24	2,930,836.08	2,930,389.26	2,930,389.26	2,930,389.26
9	2,976,205.01	2,960,606.48	2,953,349.68	2,950,260.52	2,953,706.32	2,954,390.32	2,950,213.37	2,950,185.01	2,950,185.01	2,950,185.01
10	2,996,663.14	2,961,782.43	2,974,108.89	2,946,596.91	2,935,548.84	2,892,682.70	2,893,199.29	2,902,973.97	2,902,973.97	2,902,973.97

A.2 Tuning maximum added columns

In the tables below, the running times (of the RMP and PP) and total number of iterations can be found of all data sets.

Table 20: Seconds solving RMP for various maximum added columns of the small data set

Small	Seconds RMP									
Instance\Columns	**1.0**	**10.0**	**50.0**	**100.0**	**200.0**	**500.0**	**25.0%**	**50.0%**	**75.0%**	**100.0%**
1	4.95	0.45	0.13	0.11	0.09	0.06	0.14	0.12	0.09	0.06
2	6.50	0.61	0.13	0.12	0.10	0.06	0.17	0.11	0.09	0.05
3	6.09	0.55	0.29	0.16	0.18	0.05	0.13	0.13	0.23	0.06
4	4.92	0.47	0.17	0.15	0.09	0.05	0.17	0.12	0.12	0.05
5	900.28	561.79	117.94	61.69	37.72	27.76	17.93	20.89	17.94	5.97
6	900.10	548.81	115.23	62.17	38.31	23.43	21.62	20.41	18.00	5.93
7	900.22	557.35	115.86	62.64	33.94	23.26	21.22	16.58	22.09	5.97
8	899.45	589.89	124.47	70.78	42.45	27.94	22.11	25.25	18.16	6.14
9	899.07	555.18	119.19	61.76	41.50	23.48	21.31	20.15	18.42	6.00
10	899.72	534.22	107.13	57.20	33.22	19.36	21.68	19.66	21.77	6.01

Table 21: Seconds solving PP for various maximum added columns of the small data set

Small	Seconds PP									
Instance\Columns	**1.0**	**10.0**	**50.0**	**100.0**	**200.0**	**500.0**	**25.0%**	**50.0%**	**75.0%**	**100.0%**
1	0.05	0.01	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
2	0.05	0.01	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
3	0.01	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
4	0.04	0.01	0.01	0.01	0.01	0.00	0.00	0.00	0.00	0.00
5	1.52	0.63	0.23	0.17	0.15	0.15	0.19	0.17	0.16	0.17
6	1.49	0.63	0.24	0.19	0.15	0.14	0.15	0.17	0.17	0.25
7	1.49	0.64	0.23	0.17	0.14	0.14	0.13	0.15	0.30	0.38
8	1.43	0.64	0.27	0.18	0.16	0.16	0.17	0.18	0.39	0.17
9	1.46	0.68	0.27	0.22	0.20	0.18	0.19	0.20	0.21	0.22
10	1.52	0.62	0.21	0.17	0.14	0.13	0.14	0.20	0.37	0.17

Table 22: Total number of performed iterations for various maximum added columns of the small data set

Small	Iterations									
Instance\Columns	**1.0**	**10.0**	**50.0**	**100.0**	**200.0**	**500.0**	**25.0%**	**50.0%**	**75.0%**	**100.0%**
1	169	18	5	4	3	2	5	4	3	2
2	196	22	5	4	3	2	6	4	3	2
3	200	21	9	5	5	2	6	4	5	2
4	167	19	6	5	3	2	6	4	4	2
5	599	221	46	24	14	9	6	6	5	2
6	600	217	45	24	14	8	7	6	5	2
7	600	219	45	24	13	8	7	5	6	2
8	594	226	47	26	15	9	7	5	5	2
9	596	219	46	24	15	8	7	5	5	2
10	597	213	43	23	13	7	7	5	6	2

Table 23: Seconds solving RMP for various maximum added columns of the medium data set

Medium	Seconds RMP									
Instance\Columns	**1.0**	**10.0**	**100.0**	**250.0**	**500.0**	**1000.0**	**25.0%**	**50.0%**	**75.0%**	**100.0%**
1	897.81	902.24	215.66	100.09	63.94	47.47	52.51	49.42	42.41	18.57
2	899.37	904.35	211.83	109.55	74.45	67.91	48.34	57.17	42.14	18.44
3	899.32	902.99	242.81	105.40	61.65	58.90	63.63	47.85	43.48	18.77
4	899.33	899.31	203.46	105.67	71.04	66.61	49.61	60.76	57.93	19.29
5	898.79	903.41	196.32	95.28	70.42	67.57	59.56	56.81	71.93	20.53
6	898.98	898.96	215.30	108.48	83.60	58.24	47.23	46.52	43.14	19.01
7	900.27	900.09	260.13	129.61	74.27	69.38	55.64	53.64	58.56	21.96
8	900.89	904.54	283.45	129.96	76.40	56.87	59.31	69.82	49.77	21.58
9	900.54	903.14	309.56	139.39	90.77	89.48	91.71	72.00	62.65	21.43
10	898.93	904.54	282.38	145.36	89.85	70.11	73.52	71.82	63.81	21.90

Table 24: Seconds solving PP for various maximum added columns of the medium data set

Medium	Seconds PP									
Instance\Columns	**1.0**	**10.0**	**100.0**	**250.0**	**500.0**	**1000.0**	**25.0%**	**50.0%**	**75.0%**	**100.0%**
1	2.35	1.51	0.78	0.59	0.74	0.64	0.60	0.67	1.23	1.27
2	2.35	1.49	0.71	109.55	74.45	0.90	1.59	0.60	1.06	1.33
3	2.28	1.47	0.75	0.63	0.64	1.74	0.77	0.87	1.05	1.30
4	2.24	1.50	0.69	0.60	0.67	1.76	0.72	0.64	1.04	1.34
5	2.24	1.47	0.71	0.60	0.65	0.66	0.77	1.65	1.02	1.41
6	2.38	1.47	0.70	0.61	0.57	0.64	0.80	0.68	1.05	1.10
7	1.64	1.18	0.67	0.57	0.52	0.53	0.64	0.66	0.81	1.14
8	1.69	1.19	0.69	0.58	0.59	0.60	0.61	0.61	0.64	1.35
9	1.64	1.20	0.73	0.63	0.60	0.65	0.67	0.62	0.74	1.01
10	1.66	1.20	0.72	0.62	0.60	0.64	0.66	0.65	0.66	1.27

Table 25: Total number of performed iterations for various maximum added columns of the medium data set

Medium	Iterations									
Instance\Columns	**1.0**	**10.0**	**100.0**	**250.0**	**500.0**	**1000.0**	**25.0%**	**50.0%**	**75.0%**	**100.0%**
1	373	205	34	15	9	6	6	5	4	2
2	377	207	34	16	10	8	6	6	4	2
3	373	206	37	16	9	7	7	5	4	2
4	372	206	34	16	10	8	6	6	5	2
5	379	209	33	15	10	8	6	6	6	2
6	372	205	34	16	11	7	6	5	4	2
7	276	175	34	16	9	7	6	5	5	2
8	272	174	35	16	9	6	6	6	4	2
9	281	179	38	17	10	8	8	6	5	2
10	277	177	36	17	10	7	7	6	5	2

Table 26: Seconds solving RMP for various maximum added columns of the large data set

Large	Seconds RMP									
Instance\Columns	**1.0**	**10.0**	**100.0**	**250.0**	**500.0**	**1000.0**	**25.0%**	**50.0%**	**75.0%**	**100.0%**
1	901.39	900.88	236.35	104.03	80.15	49.63	52.38	56.09	53.71	19.60
2	900.31	903.19	275.38	18.00	84.59	62.22	65.45	59.51	53.08	21.03
3	901.21	901.88	235.45	107.33	79.72	48.71	51.87	56.51	52.17	20.30
4	900.99	907.25	785.28	380.93	259.77	190.14	201.65	136.30	158.04	1,098.75
5	901.06	906.12	679.74	324.11	225.38	182.54	160.97	161.78	117.30	1,118.04
6	900.30	904.12	738.33	336.51	228.48	156.08	173.89	165.40	119.04	1,154.56
7	901.93	906.27	779.05	304.99	227.90	146.69	170.11	128.31	236.93	1,054.64
8	900.68	901.62	819.90	359.32	239.92	184.76	166.43	129.10	193.05	1,086.03
9	901.57	902.03	952.55	345.24	164.72	187.07	128.07	130.38	158.15	1,091.84
10	901.15	903.49	736.81	346.95	229.62	188.12	162.92	237.20	165.09	1,816.00

Table 27: Total number of performed iterations for various maximum added columns of the large data set

Large	Seconds PP									
Instance\Columns	**1.0**	**10.0**	**100.0**	**250.0**	**500.0**	**1000.0**	**25.0%**	**50.0%**	**75.0%**	**100.0%**
1	1.53	1.32	0.99	0.63	0.94	0.65	1.09	0.82	1.25	1.81
2	1.94	1.17	1.01	0.68	0.96	0.60	1.14	0.61	1.52	1.91
3	1.62	1.18	0.97	0.56	0.95	0.63	1.10	0.86	1.16	1.22
4	1.99	1.45	2.15	1.45	2.00	1.54	2.41	2.04	2.92	6.12
5	1.98	1.44	1.91	1.31	1.93	1.28	2.17	2.14	2.78	6.86
6	1.55	1.83	1.61	1.96	1.38	2.31	1.72	2.85	2.31	6.40
7	1.95	1.65	1.61	1.95	1.31	1.92	1.57	2.50	2.43	6.03
8	1.64	1.63	1.64	2.03	1.50	2.14	1.51	2.85	2.30	5.99
9	1.85	1.45	4.45	1.59	1.72	2.19	1.32	2.67	2.44	5.79
10	1.05	1.88	2.11	1.47	1.36	1.62	1.49	2.07	3.02	6.76

Table 28: Seconds solving PP for various maximum added columns of the large data set

Large	Iterations									
Instance\Columns	**1.0**	**10.0**	**100.0**	**250.0**	**500.0**	**1000.0**	**25.0%**	**50.0%**	**75.0%**	**100.0%**
1	312	198	34	15	10	6	6	6	5	2
2	265	172	37	18	10	7	7	6	5	2
3	279	177	34	15	10	6	6	6	5	2
4	197	132	53	23	14	9	8	5	5	2
5	195	130	49	21	13	9	7	6	4	2
6	185	133	50	22	13	8	7	6	4	2
7	190	134	50	21	13	8	7	5	7	2
8	191	131	51	23	13	9	7	5	6	2
9	197	131	41	22	13	9	6	5	5	2
10	199	132	51	22	13	9	7	8	5	2

A.3 Final results

The results obtained after parameter tuning can be found in the tables below. For every data set, 4 tables are used. DW means Dantzig-Wolfe. The upperbound is calculated using the method in Section 5.4.1. UB means the results of this upperbound. DW+Cplex means Dantzig-Wolfe and after that solving it using solver CPLEX (Section 5.4.2). UBHeur means the heuristic as explained in Section 5.4.3. Cplex means solving the problem using a commercial solver. UB Cplex means the found upperbound using a commercial solver. Dec2 means the decomposition as explained in Section 6. B&P means the obtained results using branch-and-price as explained in Section 5.5.

Table 29: Final results small data set, part 1

Small	DW			UB			DW+Cplex		
Instance\	LB	UB	Gap	Setup	Production	Holding	Time	LB	UB
1	90,732.94	120,609.48	32.93	36,158.00	77,033.86	7,417.62	0.63	103,950.52	103,950.52
2	83,017.84	115,984.13	39.71	36,775.00	75,115.01	4,094.12	0.83	97,918.64	97,918.64
3	83,906.09	147,735.17	76.07	71,388.00	72,387.07	3,960.10	1.35	108,566.20	108,566.20
4	91,019.69	163,768.94	79.93	79,940.00	77,734.17	6,094.77	1.51	113,285.05	113,285.05
5	831,951.85	1,263,492.72	51.87	448,563.00	742,039.56	72,890.16	3,576.26	956,098.04	969,766.26
6	718,566.88	1,325,702.99	84.49	628,408.00	692,705.19	4,589.80	3,576.88	873,382.65	895,905.27
7	733,756.90	1,166,061.85	58.92	452,264.00	696,479.13	17,318.72	3,593.95	871,854.13	889,338.44
8	731,467.37	1,994,066.59	172.61	1,297,116.00	691,504.84	5,445.75	3,581.34	929,128.59	962,716.01
9	759,055.05	1,616,335.17	112.94	892,149.00	701,048.17	23,138.00	3,592.89	954,948.21	976,865.66
10	768,493.90	1,340,846.81	74.48	595,603.00	711,763.02	33,480.79	3,589.79	916,428.85	935,119.80

Table 30: Final results small data set, part 2

Small	DW+Cplex			UBHeur			UBHeur		
Instance\	Setup	Production	Holding	Time	Pre-Opt	Post-Opt	Setup	Production	Holding
1	12,869.00	77,087.69	13,993.82	0.50	106,188.33	106,188.33	13,725.00	71,097.43	21,365.90
2	13,300.00	75,117.37	9,501.27	0.91	109,999.22	109,999.22	18,261.00	70,039.31	21,698.92
3	24,486.00	72,242.56	11,837.64	0.61	120,222.61	120,222.61	27,543.00	72,562.00	20,117.61
4	19,711.00	77,844.06	15,729.99	0.63	126,742.64	126,742.64	21,789.00	80,224.26	24,729.38
5	95,139.00	742,690.91	131,936.35	415.90	1,263,492.72	1,113,641.35	228,129.00	831,951.85	53,560.50
6	112,310.00	692,925.60	90,669.67	267.65	1,065,527.44	974,104.91	127,328.00	698,067.73	148,709.17
7	109,510.00	696,930.51	82,897.93	169.92	1,001,480.17	953,852.92	96,481.00	699,018.17	158,353.75
8	153,609.00	692,775.28	116,331.74	273.03	1,330,564.53	979,224.28	104,862.00	694,143.17	180,219.11
9	145,714.00	702,202.98	128,948.68	187.92	1,200,004.60	964,925.03	68,024.00	700,614.77	196,286.26
10	113,771.00	712,541.42	108,807.38	128.44	1,079,517.35	1,006,305.83	90,722.00	726,146.85	189,436.98

Table 31: Final results small data set, part 3

textbfSmall	Cplex			UB Cplex			Dec2		
Instance	**Time**	**LB**	**UB**	**Setup**	**Production**	**Holding**	**Time**	**Correct obj**	**Obj val**
1	38.44	103,075.18	103,075.18	14,532.00	77,117.51	11,425.67	0.42	122,127.59	199,563.59
2	17.44	96,927.15	96,927.15	13,633.00	75,119.23	8,174.91	0.46	115,878.13	172,197.13
3	193.01	105,264.70	105,264.70	20,977.00	72,457.38	11,830.32	0.44	149,108.25	323,149.25
4	48.47	111,942.88	111,942.88	18,200.00	77,832.29	15,910.60	0.47	153,316.19	291,810.19
5	3,600.72	941,738.98	966,448.50	94,499.00	744,559.97	127,389.54	5.53	1,245,115.64	2,121,925.64
6	3,600.26	855,874.88	904,106.02	122,132.00	693,683.65	88,290.37	5.57	1,327,071.40	2,427,860.40
7	3,600.54	866,410.47	890,298.63	101,930.84	698,420.69	89,947.10	6.84	1,125,824.61	2,070,299.61
8	3,601.47	897,659.48	968,885.98	161,192.00	693,149.51	114,544.46	4.72	1,867,597.81	2,873,118.81
9	3,600.64	925,767.84	969,076.10	152,740.00	704,169.74	112,166.36	6.05	1,622,665.33	3,865,218.33
10	3,602.94	901,869.89	934,143.14	117,374.00	713,781.06	102,988.08	5.72	1,298,157.75	2,787,276.75

Table 32: Final results small data set, part 4

Small	**Dec2**			**B&P**				
Instance	**Setup**	**Production**	**Holding**	**new LB**	**new UB**	**Gap**	**Iterations**	**Improv**
1	37,732.00	77,116.17	7,279.43	92,009.51	117,334.06	27.52	2,812	16.41
2	36,544.00	75,115.01	4,219.12	83,959.13	114,699.72	36.61	14,520	7.80
3	72,916.00	72,106.30	4,085.95	85,978.24	146,270.59	70.13	13,967	7.82
4	64,260.00	79,230.41	9,825.78	97,140.40	151,816.36	56.29	16,470	29.58
5	424,195.00	742,483.89	78,436.75	835,390.88	1,237,364.16	48.12	162	7.24
6	625,458.00	692,882.46	8,730.94	721,129.08	1,318,302.83	82.81	131	1.99
7	400,955.00	698,229.08	26,640.53	736,052.34	1,144,679.45	55.52	136	5.77
8	1,035,775.00	755,755.96	76,066.85	732,451.42	1,932,453.95	163.83	133	5.09
9	894,882.00	701,198.86	26,584.47	760,327.10	1,601,288.82	110.61	123	2.07
10	540857.00	712,759.66	44,541.09	770,949.00	1,293,466.47	67.78	146	9.00

Table 33: Final results medium data set, part 1

Medium	DW			UB			DW+Cplex		
Instance\	LB	UB	Gap	Setup	Production	Holding	Time	LB	UB
1	713,818.91	1,319,450.14	84.84	620,037.00	697,889.57	1,523.57	3,556.83	859,633.81	879,602.18
2	714,821.41	1,294,013.62	81.03	593,610.00	698,737.33	1,666.29	3,557.83	852,376.95	877,656.35
3	713,598.36	1,214,136.14	70.14	515,322.00	696,101.14	2,713.00	3,542.99	839,508.08	858,889.84
4	719,464.57	1,843,520.13	156.24	1,148,704.00	690,214.00	4,602.13	3,556.46	899,660.04	931,495.37
5	734,674.74	1,868,608.08	154.34	1,163,188.00	697,509.47	7,910.61	3,555.27	915,876.47	972,187.06
6	728,800.75	2,209,051.76	203.11	1,512,341.00	691,701.94	5,008.82	3,552.91	907,148.34	962,708.90
7	1,380,660.67	2,012,645.65	45.77	660,320.00	1,143,567.65	208,758.00	3,551.30	1,601,372.66	1,627,360.70
8	1,062,761.82	2,079,410.60	95.66	1,041,207.00	1,035,431.72	2,771.88	3,554.33	1,283,753.94	1,324,709.09
9	1,062,102.45	1,758,412.60	65.56	585,895.00	1,056,445.60	116,072.00	3,528.08	1,255,289.21	1,288,377.20
10	1,062,645.26	2,099,228.15	97.55	1,060,109.00	1,035,518.96	3,600.19	3,541.89	1,293,858.13	1,344,328.36

Table 34: Final results medium data set, part 2

Medium	DW+Cplex			UBHeur			UBHeur		
Instance\	Setup	Production	Holding	Time	Pre-Opt	Post-Opt	Setup	Production	Holding
1	101,765.00	698,511.53	79,325.66	493.37	1,035,992.77	924,888.57	78,761.00	698,277.40	147,850.17
2	103,060.00	698,817.48	75,778.87	401.30	985,203.45	909,276.75	73,369.00	698,510.95	137,396.80
3	98,009.00	695,846.86	65,033.97	402.98	990,130.66	934,391.72	83,889.00	702,037.46	148,465.27
4	130,847.00	690,320.37	110,328.00	457.48	1,240,742.30	934,647.63	61,554.00	690,812.24	182,281.39
5	118,750.00	772,695.52	80,741.54	251.98	1,221,956.85	1,031,551.58	140,258.00	705,690.96	185,602.62
6	110,487.00	773,789.29	78,432.61	267.84	1,298,491.63	1,065,107.60	185,617.00	708,137.17	171,353.43
7	175,419.00	1,144,718.18	307,223.53	589.89	1,828,864.80	1,828,864.80	341,479.00	1,139,770.10	347,615.69
8	165,595.00	1,035,503.44	123,610.65	737.20	1,649,694.71	1,521,541.45	147,084.00	1,080,146.85	294,310.59
9	142,973.00	1,040,213.82	105,190.38	522.09	1,625,805.91	1,547,074.34	132,001.00	1,095,443.32	319,630.02
10	174,834.00	1,035,445.31	134,049.06	740.10	1,595,499.58	1,595,499.58	378,130.00	1,012,199.06	205,170.52

Table 35: Final results medium data set, part 3

Medium	Cplex			UB Cplex			Dec2		
Instance\	Time	LB	UB	Setup	Production	Holding	Time	Correct obj	Obj val
1	3,600.13	824,367.02	867,455.02	103,875.00	697,542.78	66,037.25	5.09	1,382,134.67	2,609,241.67
2	3,600.17	827,075.82	875,530.93	107,852.00	698,209.99	69,468.94	5.62	1,356,603.39	2,572,800.39
3	3,600.27	810,585.86	849,326.50	93,450.00	695,597.67	60,278.83	6.45	1,242,037.59	2,238,471.59
4	3,600.23	869,870.63	895,699.33	118,348.00	693,605.33	83,746.00	6.24	1,711,677.57	3,912,219.57
5	3,600.31	920,804.17	954,427.01	110,671.00	768,341.74	75,414.27	6.01	1,983,607.44	4,508,731.02
6	3,600.28	910,091.45	948,287.84	105,739.00	769,587.33	72,961.51	5.59	2,217,725.52	4,861,321.52
7	3,600.70	1,579,202.34	1,628,128.83	178,621.00	1,145,711.03	303,796.80	13.32	2,007,879.29	3,718,919.29
8	3,607.59	1,257,478.40	1,334,131.85	176,864.00	1,035,369.90	121,897.95	10.28	1,956,313.83	2,927,225.83
9	3,600.26	1,233,635.98	1,298,371.37	151,354.00	1,040,795.45	106,221.91	13.58	1,753,603.83	3,528,316.83
10	3,600.18	1,259,560.40	1,367,579.83	199,009.00	1,036,260.12	132,310.71	10.11	2,153,603.40	3,834,407.40

Table 36: Final results medium data set, part 4

Medium	Dec2			B&P				
Instance\	Setup	Production	Holding	new LB	new UB	Gap	Iterations	Improv
1	680,608.00	698,155.88	698,155.88	721,280.37	1,281,378.35	77.65	84	8.48
2	654,045.00	699,132.23	3,426.16	721,768.90	1,288,558.62	78.53	80	3.08
3	536,577.00	696,722.25	8,738.33	717311.88	1,162,205.42	62.02	63	11.58
4	949,368.00	719,831.05	42,478.52	723,696.70	1,790,548.85	147.42	64	5.64
5	1,336,192.00	640,611.12	6,804.32	737,019.39	1,856,135.50	151.84	66	1.62
6	1,522,736.00	691,308.72	3,680.80	729,928.27	2,198,832.09	201.24	75	0.92
7	654,827.00	1,142,803.57	210,248.72	1,398,914.22	2,002,983.79	43.18	56	5.66
8	906,692.00	1,035,969.39	13,652.44	1,068,827.78	2,078,643.68	94.48	53	1.24
9	534,410.00	1,089,935.71	129,258.11	1,097,322.20	1,757,560.67	60.17	55	8.22
10	1,111,685.00	1,035,825.45	6,092.94	1,067,973.52	2,008,702.09	88.09	53	9.70

Table 37: Final results large data set, part 1

Large	DW			UB			DW+Cplex		
Instance\	LB	UB	Gap	Setup	Production	Holding	Time	LB	UB
1	1,269,624.52	2,125,958.11	67.45	893,134.00	1,103,082.37	129,741.74	3,550.13	1,514,091.04	1,542,622.78
2	1,086,055.32	3,063,958.04	182.12	2,014,057.00	1,043,272.93	6,628.11	3,550.03	1,386,905.68	1,456,846.21
3	1,134,789.52	1,987,264.11	75.12	763,134.00	1,127,551.37	96,578.74	3,537.14	1,405,687.33	1,437,088.41
4	1,058,038.36	2,067,002.03	95.36	1,030,921.00	1,033,147.56	2,933.47	3,598.94	1,284,621.51	1,326,067.61
5	1,053,729.81	2,079,396.35	97.34	1,044,537.00	1,032,949.13	1,910.22	3,540.86	1,267,456.49	1,317,684.01
6	1,058,163.47	2,022,683.52	91.15	983,768.00	1,036,609.40	2,306.12	3,552.99	1,272,795.15	1,311,135.08
7	1,093,988.31	2,889,135.08	164.09	1,834,651.00	1,042,748.02	11,736.06	3,545.42	1,426,296.04	1,484,539.03
8	1,077,376.80	2,956,221.71	174.39	1,914,655.00	1,034,947.39	6,619.32	3,544.43	1,365,933.32	1,424,484.78
9	1,079,429.52	2,953,349.68	173.60	1,908,694.00	1,038,046.68	6,609.00	3,510.04	1,369,209.20	1,423,100.14
10	1,077,536.53	2,974,108.89	176.01	1,930,972.00	1,036,210.57	6,926.32	3,527.91	1,364,712.49	1,446,849.80

Table 38: Final results large data set, part 2

Large	DW+Cplex			UBHeur			UBHeur		
Instance\	Setup	Production	Holding	Time	Pre-Opt	Post-Opt	Setup	Production	Holding
1	190,801.00	1,102,062.94	249,758.84	458.37	1720,558.49	1,579,480.84	115,939.00	1,113,361.96	350,179.88
2	221,686.00	1,043,769.57	191,390.64	689.49	1,864,519.27	1,864,519.27	671,598.00	1,016,930.98	175,990.29
3	178,561.00	1,122,653.08	135,874.33	450.09	1,654,514.51	1,495,158.48	107,854.00	1,186,852.50	200,451.97
4	160,296.00	1,032,717.73	133,053.88	1,197.38	1,579,047.05	1,579,047.05	372,994.00	1,004,466.32	201,586.73
5	155,150.00	1,033,595.73	128,938.29	947.46	1,542,519.50	1,542,519.50	327,563.00	1,005,042.19	209,914.31
6	157,946.00	1,036,639.86	116,549.22	1,087.98	1,552,048.92	1,396,476.85	118,054.00	1,042,209.74	236,213.11
7	242,740.00	1,043,537.79	198,261.24	1,342.99	1,996,895.83	1,996,895.83	779,148.00	1,014,653.77	203,094.06
8	218,283.00	1,034,693.01	171,508.77	1,427.53	1,982,303.05	1,982,303.05	686,918.00	1,031,428.17	263,956.87
9	208,696.00	1,040,509.95	173,894.19	765.92	1,806,822.50	1,806,822.50	550,779.00	1,013,597.55	242,445.94
10	231,361.00	1,036,717.06	178,771.74	3,051.79	1,849,405.81	1,543,305.63	221,738.00	1,042,977.57	278,590.06

Table 39: Final results large data set, part 3

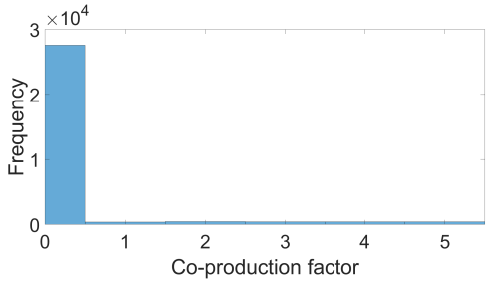
Large	Cplex			UB Cplex			Dec2		
Instance\	Time	LB	UB	Setup	Production	Holding	Time	Correct obj	Obj val
1	3,600.00	1476,962.27	1,532,930.34	199,619.00	1,101,695.74	231,615.59	11.13	2,998,960.07	6,333,800.07
2	3,600.00	1,341,767.86	1,448,450.57	230,893.00	1,048,589.88	168,967.68	11.67	3,057,340.01	5,827,897.01
3	3,600.00	1,439,107.70	1,480,639.31	178,450.00	1,152,374.60	149,814.71	11.28	2,072,593.97	4,654,471.97
4	3,600.00	1,241,281.68	1,316,383.20	170,650.00	1,032,615.68	113,117.53	12.23	2,047,137.19	4,116,740.19
5	3,600.00	1,231,217.44	1,316,394.78	172,872.00	1,031,774.46	111,748.32	10.77	2,122,113.26	4,076,435.26
6	3,600.00	1,236,597.07	1,319,035.86	169,688.00	1,036,496.59	112,851.28	12.43	2,072,454.80	4,231,435.80
7	3,600.00	1,348,606.62	1,453,449.48	224,035.00	1,051,970.56	177,443.93	12.28	2,667,577.58	6,764,174.58
8	3,600.00	1,315,920.45	1,384,624.40	203,142.00	1,039,214.96	142,267.44	11.84	2,407,335.51	6,980,122.51
9	3,600.00	1,317,616.42	1,379,233.68	187,952.00	1,047,902.06	143,379.62	10.61	2,821,613.36	6,528,044.36
10	3,600.00	1,307,368.82	1,394,550.34	208,702.00	1,044,191.89	141,656.45	11.13	2,998,960.07	6,333,800.07

Table 40: Final results large data set, part 4

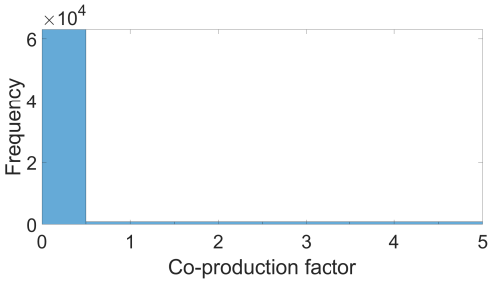
Large	Dec2			B&P				
Instance\	Setup	Production	Holding	new LB	new UB	Gap	Iterations	Improv
1	1,954,788.00	1,038,146.31	6,025.76	1,271,528.54	2,118,213.25	66.59	67	1.27
2	2,004,388.00	1,044,167.23	8,784.78	1,087,123.49	3,001,470.06	176.09	52	3.31
3	971,407.00	1,081,474.54	19,712.42	1,140,078.21	1,980,207.07	73.69	67	1.91
4	998,740.00	1,034,975.54	13,421.66	1,059,916.59	2,057,776.77	94.15	29	1.28
5	1,081,725.00	1,034,184.55	6,203.71	1,059,226.55	2,072,791.83	95.69	24	1.69
6	1,025,575.00	1,038,046.08	8,833.72	1,058,773.09	2,014,570.48	90.27	32	0.96
7	1,560,337.00	1,062,469.90	44,770.68	1,094,132.62	2,888,862.52	164.03	23	0.04
8	1,276,903.00	1,058,343.47	72,089.04	1,078,758.32	2,939,469.45	172.49	23	1.09
9	1,768,930.00	1,039,299.65	13,383.72	1,082,142.34	2,857,657.76	164.07	24	5.49
10	1,954,788.00	1,038,146.31	6,025.76	1,078,410.48	2,958,529.28	174.34	24	0.95

B Additional figures

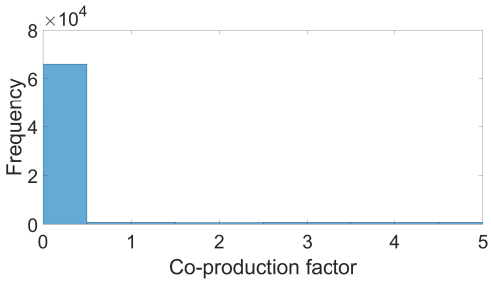
B.1 Histograms co-production factor



(a) Histogram of the co-production factor of the small data set



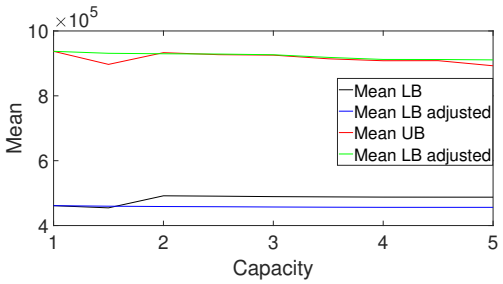
(b) Histogram of the co-production factor of the medium data set



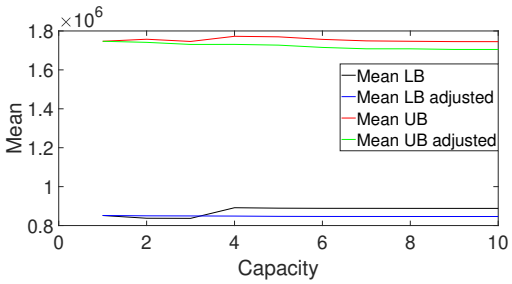
(c) Histogram of the co-production factor of the large data set

Figure 11: Histograms of the co-production factor of all three data sets including zero

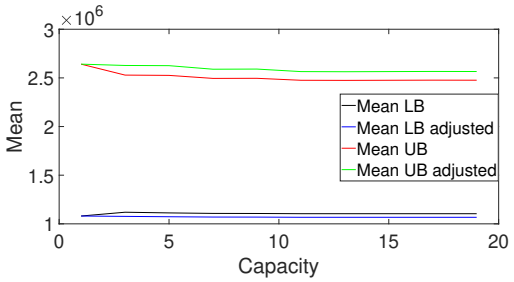
B.2 Inventory Limits



(a) Mean of solution values for various inventory limit of small data set



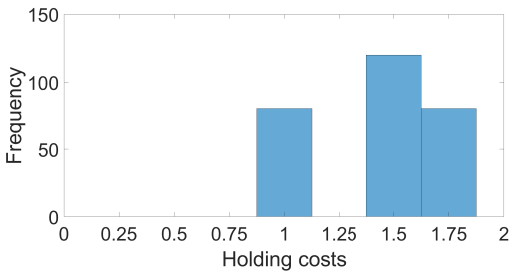
(b) Mean of solution values for various inventory limit of medium data set



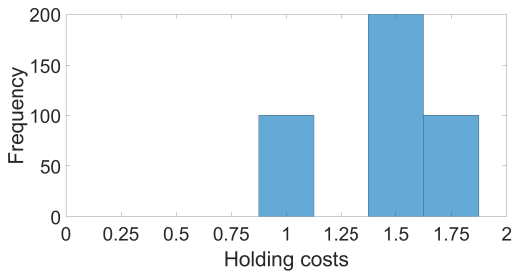
(c) Mean of solution values for various inventory limit of large data set

Figure 12: Mean of solution values for various inventory limits

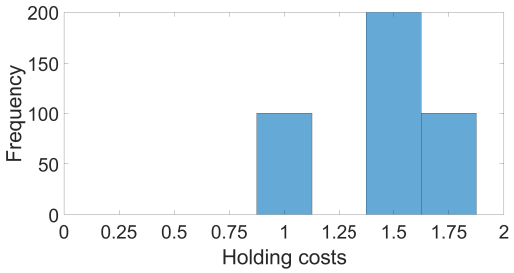
B.3 Holding costs



(a) Histogram of holding costs of small data set



(b) Histogram of holding costs of medium data set



(c) Histogram of holding costs of large data set

Figure 13: Histogram of holding costs of all data sets