Erasmus University Rotterdam

Erasmus School of Economics

Master Thesis Operations Research & Quantitative Logistics

Comparing exact and heuristic algorithms to optimise the production planning and scheduling of circular online grocer Pieter Pot

Date final version: October 1, 2021

Name student Simone Hilhorst
Student ID number 489074

Company supervisor:
Sjoerd van Bekhoven

Supervisor: Dr. Wilco van den Heuvel
Second assessor: Prof. dr. Albert Wagelmans

Abstract

In this thesis we focus on the optimisation of the production planning of circular online grocer Pieter Pot. Pieter Pot delivers their groceries in glass jars. Products arrive at Pieter Pot in bulk. The bulk needs to be distributed over the jars, to which we refer as production. The production planning consists of two stages: determining the optimal production quantities per product and a production schedule. Determining optimal production quantities is a problem known in literature as lot-sizing. We study two different approaches. The first is finding a good feasible solution while solving the stages simultaneously. We do this in two ways: exact by using an Mixed Integer Linear Program (MILP) formulation and heuristically via a Relax & Fix algorithm. The second approach, uses a good feasible solution obtained by solving the lot-sizing problem as input for the scheduling problem. We study multiple approaches for both lot-sizing and scheduling. We introduce an MILP and Variable Neighbourhood Descent algorithm to find a solution to the lot-sizing. The results are the input for a Column Generation heuristic and Greedy Randomised Adaptive Search algorithm (GRASP) that aim to find good feasible schedules. We conclude that for the instance size of Pieter Pot's case the sequential approach using a MILP for lot-sizing and Column Generation heuristic for scheduling perform best.

Contents

Abbreviations

Abbreviation	Meaning
GRASP	Greedy Randomised Adaptive Search Procedure
ILP	Integer Linear Program
LP	Linear Program
MCLSP	Multi item Capacitated Lot-Sizing Problem
MILP	Mixed Integer Linear Porgram
MOQ	Minimum Order Quantities
RMP	Restricted Master Problem
SEC	Sub-tour Elimination Constraint
SKU	Stock Keeping Unit
VND	Variable Neighbourhood Descent
WOI	Weeks Of Inventory

Table 1: List of abbreviations in alphabetical order

Chapter 1

Introduction

The convenience and efficiency of having your groceries delivered at your door is what made the market for online grocers grow the past years. Most customers expect to receive their orders on a chosen time and location. While it is convenient for each individual customer, it is not beneficial for the environment. Almost all grocers wrap their products in plastic resulting in extremely high amount of plastic waste. This is not only a problem caused by online grocers, but a problem that all supermarkets are facing. To tackle the plastic problem while maintaining the ease of having your groceries delivered at a chosen time and location, Pieter Pot was founded a little over two years ago. Pieter Pot is the first online supermarket that delivers their products waste free. They do this by delivering the products in glass jars that are cleaned and re-filled. Collecting, cleaning and refilling the jars highly increases the complexity of the supply chain of Pieter Pot compared to other online grocers. In this thesis, we report on research done on the optimisation of the production planning of Pieter Pot. This research is of interest for Pieter Pot as well as it serves academic purposes. This chapter provides an introduction to the problem and the goals of this thesis.

1.1 About Pieter Pot

Pieter Pot currently handles approximately 2500 orders per week. Each week this number increases with approximately 100 orders. This growth is realised by allowing people from the waiting list to create an account and place their first order. With many people on the waiting list the growth curve could be steeper. However, to control the growth and minimise the increasing pressure on the operations only a limited amount of people is allowed to create an account each week.

When customers place their first order they pay a deposit of two euros per jar. For all successive orders the customers have the opportunity to return used jars upon their next delivery. If jars are returned without any damage, the money is added to their Pieter Pot credit account to be used for a new order. Pieter Pot procures the products in bulk at many different suppliers, after which the bulk is distributed over the jars by people working in social workshops. The filled jars are stored in the warehouse and are ready to be picked for an incoming order.

With the aim of not losing any customers because products are not in stock, Pieter Pot needs to determine how many jars to fill (referred to as produce from here on) and how many jars to have in inventory while maintaining as little out of stocks as possible. Currently, production is planned based on the number of incoming orders, past inventory levels and a sales forecast per product. With a minimal lead time of 2 weeks, Pieter Pot estimates when and how much production must take place. First the inventory of filled jars per week is determined by taking the last known start inventory. From this value they subtract the cumulative sales forecast and add the cumulative planned production. This returns an estimated inventory per product per week. Based on these new start inventories the time until the next stock-out is estimated. This period is called the weeks of inventory (WOI). For each product an ideal WOI is determined based on bulk volume and cost price. Products are scheduled to be produced whenever the amount of inventory is getting close to the buffer period. The production amounts are determined so that the inventory after production matches the WOI. When the production amounts are determined, the products are manually scheduled to be produced in one of the social workplaces.

Figure 1.1 shows the supply chain of Pieter Pot. In which all outbound transport is planned and executed by PostNL, inbound transport is done by Pieter Pot. We follow the cycle of one jar starting clean and empty at one of the social workplaces where it is filled. Then, the jar is stored in the warehouse, where it waits to be picked for an incoming order. The order is picked-up by PostNL by whom it is transported to the customer. Whenever the customer receives a new order, empty jars can be returned to PostNL. PostNL transports the jars to the cleaning location after which the jars are returned to the social workplaces. One additional part of the production cycle is the inbound of bulk from suppliers. They either bring the bulk to the workplaces so that jars can be filled or to the warehouse if it is a product not sold in jars.

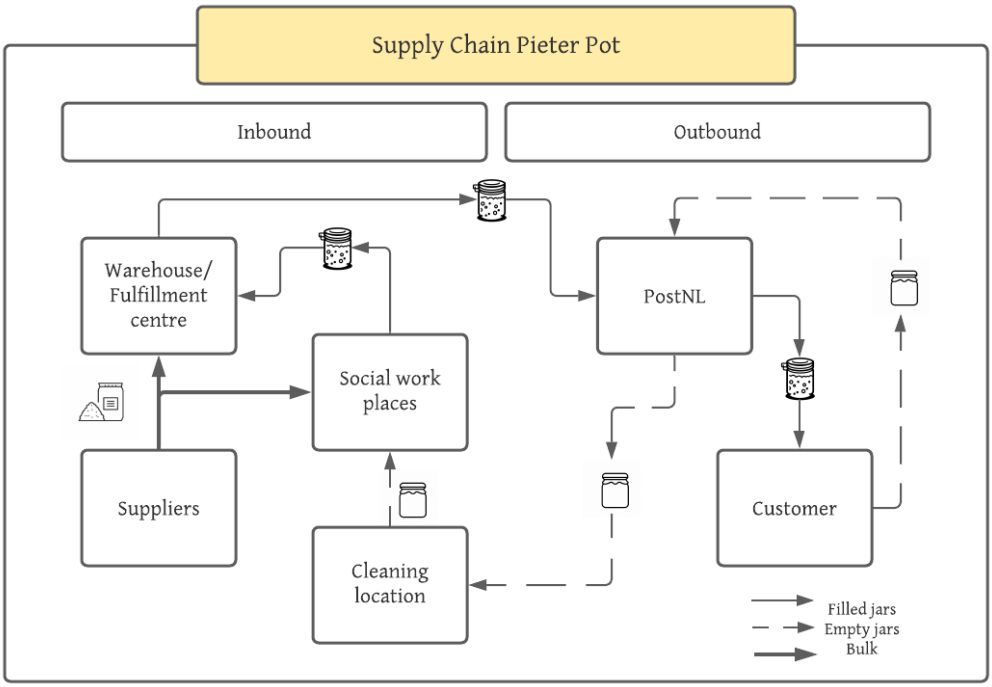


Figure 1.1: Pieter Pot supply chain

1.2 Pieter Pot's case

As described above, the production process consists of two steps. The first is determining the number of jars that need to be produced per product, we refer to this as a job. We refer to a specific product as Stock Keeping Unit (SKU). Second, the jobs need to be scheduled. Currently, the necessary production amounts are determined using an heuristic approach and scheduling is done manually. Therefore, improving these processes with new models is relevant. We first elaborate on determining the production amounts in Section 1.2.1. Second, we explain the scheduling problem in Section 1.2.2.

1.2.1 Production amounts

Even though the current method for determining production quantities works quite well, the increasing amount of orders makes it beneficial to optimise the production planning. The current solution method does not optimise the production quantities and has a few disadvantages. Firstly, there is little differentiation between products. Most products have the same buffer period (minimum weeks of inventory), lead time and weeks of inventory. This is inconvenient since for some fast selling products, i.e. fast movers, it might be better if these variables are different compared to slower selling products, i.e. slow movers. For example, it might be better to have a lower number of weeks of inventory for fast movers, otherwise the amount of jars might get too large. Additionally, the fast growth of Pieter Pot results in more outliers in orders which results in outliers in production. Outliers in production can also occur due to a certain periodicity. For example, if SKUs are always produced with WOI ≤ 1, it could occur that one week 40000 jars are produced and the next week only 10000. This is undesirable because the production location might not be able to handle these outliers due to capacity restrictions. Second, it is not possible to produce 40000 jars in one week and 10000 jars the next week unless the production team is made aware of this well in advance. It is better to average these numbers and produce 25000 every week. Determining the best time and quantity of production per product needs to be optimised. This is why Pieter Pot wants a less greedy approach to determine where, when and how much to produce while having as little inventory and stock outs as possible. Finally, the growing amount of customers leads to increasing production quantities. Consequently, more capacity at the filling locations is necessary. This needs to be known well in advance since not all locations can easily scale up their capacity and other social workplaces must be found.

The optimisation of production quantities as described above resembles the well-known lot-sizing problem. Lot-sizing problems focus on determining optimal batch sizes. In this problem we address the optimal lot-sizes for production quantities. Thus, how much jars of each SKU should be produced to fulfil the demand each week. Solving the lot-sizing problem returns the amount of jars that needs to be produced per SKU. Additional to the products, lot-sizes of jars, crates, seals and labels need to be determined. A part of the jars is returned clean after usage of customers. If more jars are needed they are ordered at the supplier. There are some characteristics that specify what a lot-sizing problem looks like (Karimi et al. (2003)). The problem Pieter Pot is facing is capacitated since there is a maximum number of jars that can be produced at each location and suppliers often have minimum order quantities (MOQ). Furthermore, the problem is single

level since the only production step is filling the jars. The aim of lot-sizing is to determine how many jars of each product need to be produced in every time interval of one week. During these time intervals, lot-sizes of multiple products are determined, making the problem a so called big bucket problem. Thus, determining the optimal production quantities is done by solving a discrete, Multi item, Capacitated Lot-Sizing Problem (MCLSP).

1.2.2 Job scheduling

The second part of the production planning is determining where and in what order production needs to take place. When the amount of jars that need to be produced is known, a job schedule must be made. A job is filling the predefined amount of jars of one product. The aim of this scheduling problem is to find an optimal production schedule. A schedule is optimal if the total completion time of all locations combined is as small as possible. The scheduling problem Pieter Pot is facing, is sequence dependent. In between consecutive jobs the production team needs to clean the area. These cleaning times (referred to as change-over times from here on) differ for combinations of jobs. This is because some products are allergy sensitive. Before production of allergy sensitive products, the complete area must be cleaned more thorough, leading to higher change-over times. However, if the previous product has the same allergy sensitivity, extra cleaning is not necessary. Sequence dependency is an important characteristic to take into account because the change-over times often determine a large part of the completion time of each lot (White & Wilson (1977)).

In the case of Pieter Pot's scheduling problem, each filling location is seen as one machine. Currently, Pieter Pot uses five different locations each with their own capacity based on the space and amount of people that work there. Schedules are made on a weekly basis. Due to the sequence dependent change-over times, batch scheduling is preferred. Batch scheduling means that all jars of one SKU are produced at once, without interruptions of production of other SKUs. The machines work in parallel on the production of batches. The production is single level. This means when a batch of a product is produced, the product is ready to be sold. Additionally, the machine scheduling problem is unrelated. Meaning, a team of people that produce the jars for a certain product are independent of another team that produces a different product. The scheduling problem can also be seen as a big bucket model with multiple products being produced in one period. Thus, the problem to be solved is an unrelated parallel machine scheduling problem (UPMSP).

1.3 Model requirements

The goal of this thesis is to improve the production process of Pieter Pot. The scope of this is to consider one week at a time. This is because the social work places schedule on a weeks basis and at the end of the week the produced products are shipped to the warehouse. The focus of Pieter Pot is currently only to improve the determination of the production quantities. However, for this thesis determining when and where production needs to take place is also included. Other than creating a fast and flexible model, requirements for the production planning are the following:

- **Little outliers in production quantities**
  As mentioned before cumulative production quantities must not differ too much in consecutive weeks since it is undesirable for scheduling the jobs. Additionally, much difference in the work load is difficult for the social work places.
- **Minimise production of the same SKU in consecutive weeks**
  Because of the need to clean the production locations when switching from one SKU to the other and because of additional costs when ordering bulk at suppliers, it is inconvenient to produce the same product in successive weeks.
- **Minimise out of stocks**
  Out of stocks are evidently undesirable and result in lost sales. Therefore, the aim is to keep the out of stocks at a minimum while maintaining the lowest inventory value as possible.

So next to minimising the cost function of the problem, these requirements are checked with metrics. Additionally, the performance of the lot-sizing model can be checked by KPIs Pieter Pot currently uses. These are: inventory value, production capacity and out of stocks. Requirements for job scheduling are to create optimal schedules that minimise the total production time.

1.4 Research question

The main goal of this thesis is to find an answer to the following question:

What algorithm is the best for optimising the production planning and scheduling of Pieter Pot?

The following sub-questions are considered:

- What algorithm is the best for solving the lot-sizing problem separately?
- What algorithm is the best for solving the scheduling problem separately?
- Can we design an algorithm that solves the lot-sizing and scheduling problem simultaneously while satisfying the model requirements?

1.5 Outline of the report

The remainder of this thesis is structured as follows. In Chapter 2 literature about lot-sizing and scheduling problems is studied. Chapter 3 presents two solution methods for finding good feasible solutions for both lot-sizing and scheduling simultaneously. Chapter 4 provides the algorithms studied to solve the lot-sizing and scheduling problems separately. Chapter 5 considers the assumptions and parameter values. Chapter 6 shows the results of the different solution methods. Chapters 7 and 8 draw conclusions and recommendations for future research.

Chapter 2

Literature review

Optimisation of production flows is a widely reviewed topic in literature. The production chain consists of several different activities such as material procurement, production planning and inventory management. Problems that often occurred in papers that study the production chain are lot-sizing and scheduling problems. Solving lot-sizing and scheduling problems was often done sequentially by first solving the lot-sizing problem. The scheduling problem can then be solved using the output of the lot-sizing problem as input values. The sequential approach is most often used in literature as well as in practical applications. However, in recent years more and more research has been done on simultaneous approaches. We first studied lot-sizing and scheduling separately in Sections 2.1 and 2.2. In this review we focused on Multi-item Capacitated Lot-Sizing problem with a dynamic demand. The demand is dynamic since the forecast is made every week. We viewed approach to solve sequence dependent scheduling problems with and without identical machines. In Section 2.3 we review literature on solving the problems simultaneously. For both approaches we studied exact methods as well as heuristic methods to find good feasible solutions.

2.1 The Lot-sizing problem

Many reviews on production planning are focused solely on lot-sizing. Karimi et al. (2003) reported that determining optimal lot-sizes is a problem that can have many different characteristics. Dynamic demand is on characteristic that increases the complexity of determining optimal lot-sizes. Karimi et al. (2003) concluded that despite the fact that many research has been done on lot-sizing problems, it remains a challenging subject to find more efficient approaches. Buschkühl et al. (2010) focused on solving just the lot-sizing problem. Buschkühl et al. (2010) studied different heuristics to solve the Multi-level Capacitated Lot-Sizing problem (MLCLSP) and found that mathematical programming based heuristics and metaheuristics are actively used in the field. The drawback of these methods was that the relative performance is hard to measure. This is mainly because of all the different versions of lot-sizing problems. One of these mathematical programming based heuristics is the Fix-And-Optimise approach (Helber & Sahling (2010)). The Fix-And-Optimise algorithm iteratively re-optimises the integer variables. The flexibility of the proposed solution method made it easy to adjust the model to changes in

the lot-sizing problem such as adding parallel machines or inventory capacities.

In addition to metaheuristics and mathematical programming approaches for solving the ML-CLSP, we found more solution approaches, namely: Local Search (Kuik et al. (1993)), Lagrangian relaxation (Toledo & Armentano (2006)), decomposition and aggregation (Tempelmeier & Helber (1994)). Another example of a decomposition approach is studied in Stadtler (2003). The method provided a decomposition based on time intervals. Lot-sizing sub-problems are solved sequentially with a rolling window while taking capacity constraints for the entire time period into consideration. The heuristic resulted in high quality solutions. However, it could not deal with positive lead-times and computation times increase substantially with increasing instance sizes.

2.2 The Scheduling problem

In literature scheduling of jobs is most frequently studied for single or sequential machines. However, the interest in studying parallel machine scheduling problems increased over the past years. Additionally, research on unrelated machine scheduling problems has been done. Rabadi et al. (2006) studied multiple heuristics to solve the unrelated Parallel Machine Scheduling Problem (PMSP) with machine-dependent and sequence-dependent setup times. They concluded that a metaheuristic provides better solutions compared to the partitioning heuristic studied in Al-Salem (2004).

In more recent research from Arnaout et al. (2010), an Ant Colony Optimisation (ACO) proved to be superior over all methods compared in Rabadi et al. (2006) and Al-Salem (2004). They partitioned the problem into an assignment and sequencing problem, proving that Ant Colony Optimisation can solve hard combinatorial optimisation problems. However, ACO only outperformed Rabadi et al. (2006) metaheuristic for a large jobs/machines ratio. That is why Arnaout et al. (2014) continued research on ACO . With some updates the ACO now outperforms the metaheuristic for all job/machine ratios.

More research on the NP-hard problem of scheduling jobs on unrelated parallel machines is done in recent years. Fleszar et al. (2012) studied a Variable Neighbourhood Descent approach and found that it outperforms a lot of previous research based on computation time. Diana et al. (2015) proposed an immune inspired algorithm for the problem that performs really well. The down side is the dependence on operators constructing the initial population. Many research has been done on heuristics for this problem. However, very recently Fanjul-Peyro et al. (2019) introduced a new MILP formulation to solve the problem using commercial solvers. This method works really well with instance sizes up to 1000 jobs and 8 machines which is comparable to the instance sizes of Pieter Pot.

Another approach studied in literature is solving job scheduling problems using Column Generation. Van den Akker et al. (1999) studied the problem of minimising the total completion time on parallel machines. The problem was formulated as a Set Covering problem. The algorithm solved problems within reasonable computation time. Additionally, many solutions turned out to be integer. However, some needed to be fixed with a Branch and Bound algorithm. Van den

Akker et al. (2005) studied the same problem only with different Branch and Bound procedures. Chen & Powell (2003) included sequence dependent set-up times and divided jobs into families so that jobs in one family were executed consecutive. Both studies concluded that Column Generation algorithms are able to solve problems of medium size within reasonable time. Problems with nonidentical machines are not as frequently studied as problems with identical machines. However, Lopes & de Carvalho (2007) studied a complex version of the job scheduling problem with nonidentical machines with availability for the machines, release and due dates for jobs and sequence dependent set-up times. Formulating this as a Set Partitioning problem and solving it with Column Generation resulted in an algorithm that solves problems with 50 machines and 150 jobs in a reasonable computational time. Most studies on solving job scheduling problems using Column Generation use a dynamic programming algorithm to solve the pricing problem. This is due to the speed of dynamic programming.

2.3 Simultaneous lot-sizing and scheduling

In the past lot-sizing and scheduling problems were solved individually. But, in recent years the need for a precise production planning increased. Thus, researchers started to investigate solving lot-sizing and scheduling problems simultaneously. Garey et al. (1976) stated that the a classic job shop problem with n jobs and m machines is NP-hard. Since the simultaneous approach adds assignment to machines and sequencing of the jobs to the problem, this problem has an even higher level of complexity. Fleischmann & Meyr (1997) studied heuristics for the General Lot-sizing and Scheduling Problem (GLSP). The problem with these heuristics was that performance of the models could only be measured relatively to each other. Additionally, solving the model resulted in rather high running times.

Meyr (2000) provided two solution methods to solve the GLSP with sequence dependent set up times. He found that Dual Re-optimisation is an effective tool to solve these types of problems. In Dual Re-optimisation the current solution is updated by a new candidate based on the dual of the problem. Whenever the dual is smaller than a certain bound, the new solution is accepted. However, the approach does not include parallel production lines. Meyr (2002) presented an approach for the GLSP with parallel production lines. He combined Local Search heuristics such as Simulated Annealing with Dual Re-optimisation. Even tough this method worked well and can easily be adapted to different forms of the GLSP problem, it resulted in high computation times. Fandel & Stammen-Hegene (2006) also concluded that the simultaneous approach can not find the optimal solution for large instance sizes. One way to reduce running times and find an optimal solution is by solving Parallel Machine Lot-sizing and Scheduling problem with identical machines instead of solving it with different machine types (Beraldi et al. (2008)). This feature makes the problem solvable for large instance sizes without increasing the running time too much. Another way to reduce the problem is to decompose the original problem. By pre-assigning jobs to machines, some binary variables are fixed and the problem can be solved by commercial solvers (Xiao et al. (2013)).

Ferreira et al. (2009) studied a problem comparable to Pieter Pot's case. They studied a parallel machine GLSP that consisted of two production levels. The research concluded that two

heuristics, the relaxation approach and the Relax & Fix heuristic, provide better solutions than previous models. The relaxation approach relaxed the constraints on bulk material, so bulk was always available. Gomez Urrutia et al. (2014) focused on the multi resource system and found a heuristic to solve these models that outperforms a standard solver. Rohaninejad et al. (2015) overcame the complexity of solving the problem with multiple levels and multiple items by developing a new hybrid algorithm. Most research is based on single-resource systems. Many approaches to solve Simultaneous Lot-sizing and Scheduling problems are studied. It can be concluded that commercial solvers are almost always outperformed by heuristics based on computation time. However, there is a wide variation of heuristics and which one solves a specific problem best, can not be concluded.

Chapter 3

Solution methods: simultaneous approach

Based on the literature study we chose two ways to improve the production process of Pieter Pot. The first one is solving lot-sizing and scheduling simultaneously. This chapter explains how the simultaneous approach can be executed. In Section 3.1 we formulate the problem as a mixed integer linear program (MILP). Next, Section 3.2 shows a Relax & Fix heuristic that can be used to speed up the computation times.

3.1 Simultaneous lot-sizing and scheduling as MILP

As commented in the literature review, solving capacitated lot-sizing problems is difficult and often results in high computation times. Combining it with optimal scheduling makes the problem even harder. In literature many of these type of problems could not be solved to optimality by a commercial solver. However, because Pieter Pot's problem is single level computation times might not be too high. Therefore, it is interesting to see what computational results a commercial solver provides. These values provide a benchmark to compare other results to. This integrated lot-sizing and scheduling problem can be formulated as a Mixed Integer Linear Program (MILP) and can therefore be solved using MILP solving techniques.

When formulating the problem as a mixed integer program, the goal is to minimise the objective function while staying within the feasible region. The feasible region is defined by a set of constraints. The objective of the integrated approach is minimising the sum of costs assigned to product inventory, machine change-overs and out-of-stocks. The product inventory consists of filled jars. Some SKUs are stored in bulk but for now bulk storage is neglected. A penalty is assigned to products that are out of stock during a time period. Since it is a big bucket problem, the planning horizon is divided into T macro periods with the length of one week. Each period $t \in T$ is divided into micro periods $s \in N$. The number of micro periods is based on the trade-off between having a precise solution and high running times. A high number of micro periods gives solutions that are more precise, but the model might be too large to solve. Other relevant sets are the following: the number of machines M and the SKUs J, with indices $m \in M$ and $i, j \in J$

respectively. The following data sets and variables are known:

$$
\begin{aligned}
B &= \text{A large number} \\
S_t &= \text{Set of micro periods in macro period } t \\
G_j &= \text{Set of machines that can produce jar } j \\
A_m &= \text{Set of products that can be produced on machine } m \\
d_{jt} &= \text{Demand in jars of item } j \text{ in period } t \\
h_j &= \text{Holding costs per jar of product } j \\
s_{ij} &= \text{Change-over costs from } i \text{ to } j \\
b_{ij} &= \text{Change-over time from } i \text{ to } j \\
c_j &= \text{Out of stock costs in jars for product } j \\
f_{mj} &= \text{Production time of one jar of item } j \text{ on machine } m \\
k_{mt} &= \text{Total time capacity on machine } m \text{ in period } t \\
q_{jm} &= \text{Minimum production amount in jars of product } j \text{ on machine } m
\end{aligned}
$$

The following decision variables are necessary to solve this problem:

$$
\begin{aligned}
I_{jt} &= \text{The number of jars in inventory of product } j \text{ at period } t \\
u_{jt} &= \text{The amount of stock-out for product } j \text{ in period } t \text{ in jars} \\
x_{mjs} &= \text{Production quantity in jars on machine } m \text{ of item } j \text{ in period } s \\
y_{mjs} &= \begin{cases} 1, & \text{if machine } m \text{ produces } j \text{ in } s \\ 0, & \text{otherwise} \end{cases} \\
z_{mijs} &= \begin{cases} 1, & \text{if there is a change-over on machine } m \text{ from product } i \text{ to } j \text{ in } s \\ 0, & \text{otherwise} \end{cases}
\end{aligned}
$$

The objective to be minimised is the following:

$$
\min \sum_{j \in J} \sum_{t \in T} h_j I_{jt} + \sum_{s \in N} \sum_{m \in M} \sum_{i \in A_m} \sum_{j \in A_m} s_{ij} z_{mijs} + \sum_{t \in T} \sum_{j \in J} u_{jt} c_j \tag{3.1}
$$

Subject to the following constraints:

$$x_{mjs} \geq q_{jm} y_{mjs} \qquad \forall j \in A_m, m \in M, s \in N; \qquad (3.2)$$

$$I_{j(t-1)} + \sum_{m \in G_j} \sum_{s \in S_t} x_{mjs} = I_{jt} + d_{jt} - u_{jt} \qquad \forall j \in J, t \in T; \qquad (3.3)$$

$$\sum_{j \in A_m} \sum_{s \in S_t} f_{mj} x_{mjs} + \sum_{i \in A_m} \sum_{j \in A_m} \sum_{s \in S_t} b_{ij} z_{mijs} \leq k_{mt} \qquad \forall m \in M, t \in T; \qquad (3.4)$$

$$\sum_{j \in A_m} y_{mjs} \leq 1 \qquad \forall m \in M, s \in N; \qquad (3.5)$$

$$x_{mjs} \leq y_{mjs} B \qquad \forall m \in M, j \in A_m, s \in S_t; \qquad (3.6)$$

$$I_{jt} + u_{jt} \geq d_{j(t+1)} \qquad \forall j \in J, t \in T; \qquad (3.7)$$

$$z_{mijs} \geq y_{mi(s-1)} + y_{mjs} - 1 \qquad \forall m \in M, i, j \in A_m, s \in N; \qquad (3.8)$$

$$\sum_{i \in A_m} \sum_{j \in A_m} z_{mijs} \leq 1 \qquad \forall m \in M, s \in N; \qquad (3.9)$$

$$I_{jt}, u_{jt} \geq 0 \qquad \forall j \in J, t \in T; \qquad (3.10)$$

$$x_{mjs} \geq 0 \qquad \forall m \in M, j \in A_m, s \in S_t; \qquad (3.11)$$

$$y_{mjs}, z_{mijs} \in \mathbb{B} \qquad \forall m \in M, i, j \in J, s \in S_t; \qquad (3.12)$$

The objective in (3.1) minimises the costs assigned to keeping inventory, change-over times and out of stocks. Constraints (3.2) make sure that production batch sizes meet the minimum production requirements. In constraints (3.3) the inventory is balanced for every macro-period based on the demand for a given period t and constraints (3.4) restrict the total production time to the time capacity of the machines. Since every machine can only produce one item in every micro period s, the sum of all items produced on machine m in period s must be less then or equal to one (constraints (3.5)). Constraints (3.6) ensure that in micro period s, production of item j on machine m only occurs if the set-up variable y_{mjs} is equal to one. A buffer period of one week is guaranteed in constraints (3.7). Constraints (3.8) and (3.9) guarantee that only one change-over of production on a machine occurs every micro period. Finally, constraints (4.9) and (4.10) ensure the domain of the decision variables.

A solution for mixed integer linear programming problems can be found using commercial solvers. These solvers use the Branch and Bound algorithm. The Branch and Bound algorithm searches for the optimal solution that, in this case, minimises the objective function within the solution space defined by the constraints. The goal of the algorithm is to do this relatively quickly by recursively splitting the problem into smaller sub-problems. Splitting is called branching. By keeping track of bounds on the minimum, the algorithm is able to prune the search space. Meaning, it eliminates parts of the solution space that will not contain the optimal solution.

3.2 Relax & Fix

Since high computation times are not preferable, we chose to focus on a heuristic that is able to find a good feasible solution for the integrated approach. The Relax & Fix heuristic based on

the research of Ferreira et al. (2009), is used to find a good feasible solution for the problem. In these heuristics the integer variable set is divided into subsets. In the proposed algorithm the variable y_{mjs} is relaxed and fixed for every macro period. The variable z_{mijs} is not relaxed since this variable value follows from equation (3.8). In each iteration, the variables in one of these sets are binary. The variables in all successive sets are relaxed. This leads to a sub-problem P_k in the k^{th} iteration. The sub-problem is solved and the results of the binary variables are fixed in the next iteration, in which the next set of variables becomes binary. This procedure continues for all sets. The advantage of this method is that the sub-problems are much easier to solve (Ferreira et al. (2010)). In figure (3.1) the algorithm is graphically explained in which we define the following sets: I_k and F_k.

I_k = the subset of integer variables that are optimised in iteration k

F_k = the set of integer variables that will be fixed

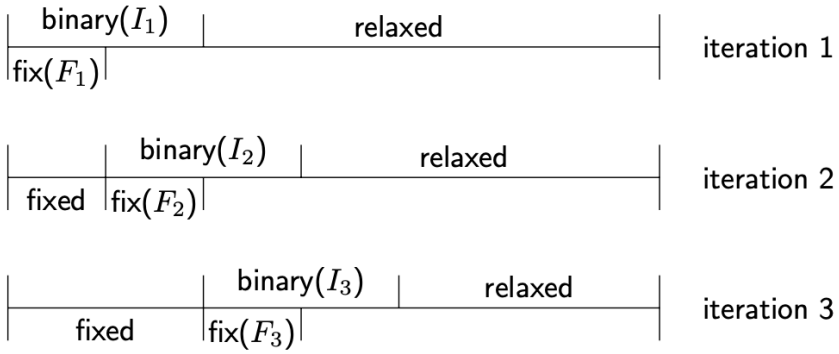


Figure 3.1: Relax & Fix per macro period (Van den Heuvel (2020))

We viewed different integrated approaches. The advantage of using an integrated approach is that lot-sizes are automatically connected to the job scheduling so that there are no infeasible solutions regarding lot-sizes and production capabilities. However, the disadvantage of solving the problem simultaneously is that it increases the complexity and thus running time. Therefore, we investigate the sequential approach in the next chapter.

Chapter 4

Solution methods: sequential approach

This chapter focuses the second solution approach: solving the lot-sizing and scheduling problem sequentially. First, we solve the lot-sizing model and we use its output as input for the creation of a good feasible schedule. Section 4.1 shows how the lot-sizes can be determined by use of an MILP or a Variable Neighbourhood Descent heuristic. Section 4.2 explains how to find a good feasible schedule with Column Generation or a heuristic using the results of lot-sizing as input.

4.1 Lot-sizing

Just as in the integrated approach, the lot-sizing problem can be formulated as an MILP. Thus, solution methods such as Branch & Bound and metaheuristics are applicable to find good feasible solutions. When formulating the lot-sizing problem there is a choice between fixing the model requirements in the constraints or varying costs assigned to situations that are undesirable. In our problem the choice is relevant for all three model requirements as described in Section 1.3: balancing the production quantities, no production of the same SKU in consecutive weeks and minimising stock-outs. We first explain how the lot-sizing problem can be solved using an MILP.

4.1.1 MILP

The objective of the lot-sizing problem is similar to the objective of the integrated approach, namely the minimisation of set-up, holding and out-of-stock costs. We again define the sets J and T and variables that are partly the same as in the integrated approach. The set J with indices $i, j \in J$ refers to the products and T is the set of periods in weeks. We introduce the following decision variables:

x_{jt} = The production quantity in jars of item j in period t

u_{jt} = The amount of stock-out for product i in period j

I_{jt} = The amount of inventory for product i in period t

$$y_{jt} = \begin{cases} 1, & \text{if product } j \text{ is produced in period } t \\ 0, & \text{otherwise} \end{cases}$$

The aim of the lot-sizing problem is to minimise the cost function:

$$\min \sum_{j \in J} \sum_{t \in T} h_j I_{jt} + \sum_{j \in J} \sum_{t \in T} u_{jt} c_j + \sum_{j \in J} p_j y_{jt} \tag{4.1}$$

The holding cost h_j is a value that represents the costs of storing one jar for one week. We choose to limit the out of stocks by assigning a cost to every sale that is lost as a result from a SKU not being in stock. The costs c_j is the penalty assigned if a product j is out of stock. We refer to the penalty assigned to initialising production of a given SKU in week t as p_j. The set-up costs are assigned to prevent production of every product in every period. Additionally, assigning a cost to decision variable y_{jt} rules out the amount of consecutive production periods. The objective is subject to the following constraints:

$$I_{j(t-1)} + x_{jt} = I_{jt} + d_{jt} - u_{jt} \qquad \forall j \in J, \quad \forall t \in T; \tag{4.2}$$

$$x_{jt} \geq q_j y_{jt} \qquad \forall t \in T, \quad j \in J; \tag{4.3}$$

$$x_{jt} - B y_{jt} \leq 0 \qquad \forall j \in J, \quad t \in T; \tag{4.4}$$

$$u_{jt} \leq d_{jt} \qquad \forall j \in J, \quad t \in T; \tag{4.5}$$

$$\sum_{j \in J} x_{jt} \leq (1 + w) \sum_{j \in J} x_{jt+1} \qquad \forall t \in T; \tag{4.6}$$

$$\sum_{j \in J} x_{jt} \geq (1 - w) \sum_{j \in J} x_{jt+1} \qquad \forall t \in T; \tag{4.7}$$

$$I_{jt} + u_{jt} \geq d_{jt} \qquad \forall j \in J, \quad t \in T; \tag{4.8}$$

$$x_{jt}, I_{jt}, u_{jt} \geq 0 \qquad \forall j \in J, \quad t \in T; \tag{4.9}$$

$$y_{jt} \in \mathbb{B} \qquad \forall j \in J, \quad t \in T; \tag{4.10}$$

Constraints (4.2) balance the inventory between two successive weeks. Constraints (4.3) give a lower bound for the production for a SKU in one week. Constraints (4.4) match the decision variables x_{ij} and y_{jt} using a big M constraint. Constraints (4.5) bound the amount of lost sales if a product is out of stock. Constraints (4.6) and (4.7) make sure the requirement of having much deviation in production quantities in successive weeks is limited by a percentage w. A buffer period of one week is guaranteed by constraints (4.8). We refer to these constraints as the smoothing constraints. Constraints (4.9) and (4.10) secure the domain of the decision variables.

The advantage of solving an MILP, compared to a metaheuristic, is that it provides a lower bound. While searching the Branch & Bound tree, we find a smallest possible upper bound $\bar{z}$ and a lower bound $\underline{z}$ defined by the LP relaxation. With $\bar{z}$ and $\underline{z}$ we can determine the optimality gap which provides a good indication of the solution quality of the algorithm. We define the optimality gap as:

$$\text{optimality gap} = \frac{|\bar{z} - \underline{z}|}{|\bar{z}|} \tag{4.11}$$

4.1.2 Metaheuristic for lot-sizing

The computation times of solving any optimisation problem using an MILP can be high. Therefore, we study a heuristic solution method as well. A heuristic prioritises computation time over finding an optimal solution. In this research we focus on a specific type of heuristic, namely metaheuristics. Metaheuristics are problem-independent generic methods that aim to find a good feasible solution.

An example of a metaheuristic is Local Search. A Local Search algorithm relevant to our problem is Variable Neighbourhood Descent (VND). The VND algorithm takes a feasible solution as input. This initial solution is made by performing Branch & Bound and stopping it after a predefined number of feasible solutions is found. We perform Branch & Bound on the MILP for our lot-sizing problem but without the smoothing constraints. The aim of the VND algorithm is to find a improved solution x^* with respect to the current solution x based on a accepting criterium. We study the VND algorithm with two acceptance criteria: the objective value $z(x)$ and the cumulative difference in production quantities in consecutive weeks $f(x)$. The algorithm searches for a better a solution in specific neighbourhoods. A neighbourhood is a subset of solutions, i.e. points. Our solution space is denoted by X and the current solution by $x \in X$. The neighbourhood $N(x) \subseteq X$ consists of all solutions that are in some sense close to x and $x' \in N(x)$ is a neighbour of x. In one neighbourhood all points can be reached from any randomly chosen point in the neighbourhood.

The VND algorithm starts searching for a new solution x' in a relatively small neighbourhood N_1. If the new solution is improved with respect to the current solution, i.e. $z(x') < z(x)$ or $f(x') < f(x)$, the new solution is accepted and the algorithm continues by searching the solution space of the new current solution. If no improvement is found, a larger neighbourhood N_2 is used to find a better solution. A larger neighbourhood consists of more points than the previous neighbourhood. By doing so, the chances of escaping a local minimum are improved. Whenever no improvements are found, the algorithm continues the search in a larger neighbourhood N_k. If no better solution is found in the largest neighbourhood, the VND algorithm is finished and returns the current solution. If a better solution is found in a larger neighbourhood, the VND algorithm starts over from the smallest neighbourhood with the improved solution. The pseudo code for the VND algorithm can be found in algorithm 1.

Algorithm 1: Variable Neighbourhood Descent

Result: Feasible lot sizes

```
x ← initial solution;
k ← 1 ;
while k ≤ K do
    x' ← N*_k(x)
    if z(x') < z(x) then
        x ← x';
        k ← 1;
    else
        k ← k + 1 ;
    end
end
Return: x
```

We define n the total number of jobs and n_0 the number of jobs for which $x_{jt} > 0$. The neighbourhoods in our VND algorithm consist of re-allocating jobs in different ways. We define the following four neighbourhoods:

1. N_1: Merge two jobs. Merge the production quantity with the previous or successive week whenever $x_{j,t-1}$ or $x_{j,t+1}$ is greater than zero. This results in a neighbourhood of size $O(2n_0) = O(n_0)$.

2. N_2: Push a job one week forward or pull one week back. This results in a neighbourhood of size $O(2n) = O(n)$.

3. N_3: Merge two jobs. Merge the production of product j in week t with the production of j in week $t + i$ with $i \in \{-6, -5, -4, -2, 2, 3, 4, 5, 6\}$. This results in a neighbourhood of size $O(9n_0) = O(n_0)$.

4. N_4: Push a job forward or pull back. Reschedule the production of product j in week t to week $t + i$ with $i = \{-6, -5, -4, -2, 2, 3, 4, 5, 6\}$. This results in a neighbourhood of size $O(9n) = O(n)$.

The neighbourhoods all place production of a chosen product j in a different week. This influences the other decision variables in all weeks. To make sure the solutions are still feasible we check the inventory balance constraints (4.2). When the changed productions still result in a feasible solution, we accept the solution. In search of a better solution, we accept the first found improved feasible solution $x^* \in N_k(x^*)$ and continue by searching $N_1(x^*)$.

It is common to use the objective value as accepting criterium. However, because of the model requirement concerning minimisation of the differentiation in production quantities per week, we perform the VND algorithm with two acceptance criteria. Additionally to accepting an improved solution based on the objective value, we check if a solution is better based on the cumulative difference between week t and $t+1$. We refer to VND with the objective value and differentiation as acceptance as VND(obj) and VND(diff) respectively.

4.2 Scheduling

The second part of the sequential approach is finding a good feasible schedule. The aim is to find a schedule so that all jobs are produced in the desired week. A good schedule means that all jobs are scheduled exactly once, on one of the five machines while minimising the total computation time. We use the optimised production quantities as input to compose such a weekly schedule. We research two ways to find feasible schedules. The first is a method that decomposes the scheduling problem into a Set Partitioning problem which can be solved using Column Generation. The second method is a metaheuristic that aims to find a good feasible schedule.

4.2.1 Scheduling as Set Partitioning problem

Lopes & de Carvalho (2007) decompose a comparable job scheduling problem to a Set Partitioning problem. In this thesis the same approach is used, only for a simplified version of the problem. The problem is represented as a directed graph $G = (N, A)$ with N the set of nodes that represent the jobs that need to be performed. The set contains $n+1$ nodes, n corresponding to the jobs and one dummy node that indicates the start and end of each job sequence. A is the set of arcs. G is a complete graph, meaning all nodes are connected to each other. Each arc has a value, namely the sequence dependent change-over time between job $i, j \in J$. A machine schedule is defined as a sequence of jobs that is performed on one machine. Thus, a schedule is an acyclic route through the graph. In Figure 4.1 we show a directed graph G consisting of 4 jobs and one dummy job. Two job schedules are visualised by two acyclic routes through the graph. A feasible solution to the Set Partitioning problem must contain all jobs, i.e. must visit all nodes in the graph.

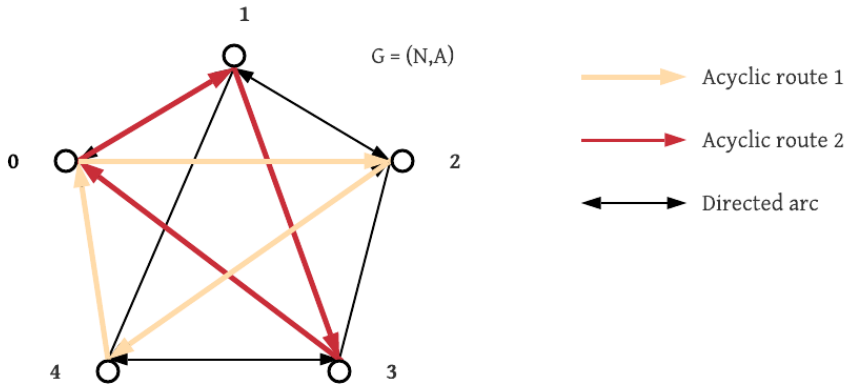


Figure 4.1: Acyclic router through directed graph G

Let S be the set of all possible machine schedules. We define the parameter a_{js} as the number of times job j is visited by path s. Additionally, we introduce the following decision variable:

$$y_s = \begin{cases} 1, & \text{if schedule } s \text{ is executed} \\ 0, & \text{otherwise} \end{cases} \tag{4.12}$$

We define the following formulation for the Set Partitioning problem:

$$\min \sum_{s \in S} c_s y_s \tag{4.13}$$

$$\text{s.t.} \sum_{s \in S} a_{js} y_s = 1 \qquad \forall j \in J \tag{4.14}$$

$$\sum_{s \in S} y_s \leq m_{max} \tag{4.15}$$

$$y_s \in \mathbb{B} \qquad \forall s \in S, \tag{4.16}$$

$$\tag{4.17}$$

The objective aims to minimise the costs of the selected schedules. Constraints (4.14) ensure that each job is exactly visited once. Constraints (4.15) ensure that at most m_{max} schedules are selected in which m_{max} represents the number of machines. The variables c_s are the completion time of sequence s equal to the costs of a path in the graph G. The costs of a schedule can be obtained by:

$$c_s = \sum_{i \in J_0} \sum_{j \in J_0} (b_{ij} + f_j) x_{ij} \tag{4.18}$$

In which f_j is the production time of item j and J_0 is the union of dummy job 0 and set J.

4.2.2 Column Generation algorithm

As a result of G being a complete graph, i.e. all jobs being able to succeed all jobs, many feasible schedules are available. Thus, the problem deals with a large number of variables. A large amount of variables may cause high computation times. Column Generation is a mathematical programming based algorithm that is often used for problems with a large number of variables. The goal of Column Generation is to only include variables that have potential to improve the objective value. New columns are added based on the simplex dual prices. Dual prices can be interpreted as the marginal costs of adding a job j to a schedule. The algorithm makes the problem easier to solve by only including a subset of the variables. This sub-problem with only a few variables included is called the restricted master problem (RMP). The algorithm starts with solving the linear relaxation of the restricted master problem with only a subset of the variables included. This subset is found by a heuristic. After the RMP is solved, the pricing problem is solved. The pricing problem finds the schedule with the maximum negative reduced costs, i.e.

the solution value is potentially decreased by adding that variable. That variable is added to the RMP. These iterations continue until no more negative cost variables can be found. The problem is then solved to optimality (Spliet (2020)). In this specific case the pricing problem is known as a Shortest Path Finding problem with resource constraints. Resource constraints ensure that schedules do not exceed the time capacity on machine m.

Let λ_j be the dual variables corresponding to the constraints (4.14) and μ to (4.15). Then, the reduced cost of a machine schedule is given by:

$$rc_s = c_s - \mu - \sum_{j \in J} \lambda_j a_{js} \tag{4.19}$$

In order to determine the schedules with negative reduced costs, this function must be minimised subject to some constraints. Two categories of constraints are necessary to generate feasible schedules. First, the schedules need to meet the maximum completion time requirement. Second, we need to eliminate sub-tours in the schedules. Many sub-tour elimination constraints (SEC) are studied in literature. Dantzig et al. (1954) proposed the first ILP SEC's. The constraints limit the number of selected arcs to the length of every possible subset of the nodes in N. With n nodes, excluding the dummy node, in the graph the number of subsets is 2^n. This results in a quickly increasing number of constrains, which blows up the computation times. This is undesirable, thus we chose to include the Gavish and Graves formulation (Bazrafshan et al. (2021)). The Gavish and Graves formulation does not increase the computation time too much. However, an optimal solution can not be guaranteed. We define the the decision variable g_{ij}:

$$g_{ij} = \text{the integer value that determines the location of arc } i \text{ to } j \text{ in the schedule}$$

Additionally, we introduce the decision variables x_{ij} and a_j:

$$x_{ij} = \begin{cases} 1, & \text{if job } j \text{ is the immediate successor of } i \text{ in the schedule} \\ 0, & \text{otherwise} \end{cases} \tag{4.20}$$

$$a_j = \begin{cases} 1, & \text{if job } j \text{ is included in the schedule} \\ 0, & \text{otherwise} \end{cases} \tag{4.21}$$

We redefine the reduced costs for each schedule as:

$$rc = \sum_{i \in J_0} \sum_{j \in J_0} c_{ij} x_{ij} + \sum_{j \in J} (f_j - \lambda_j) a_j \tag{4.22}$$

Finally, the pricing problem can be formulated as the following MILP:

$$\min \quad rc \tag{4.23}$$

$$s.t. \sum_{i \in J} \sum_{j \in J} c_{ij} x_{ij} + \sum_{j \in J} f_j a_j \leq k_{mt} \tag{4.24}$$

$$\sum_{i \in J} x_{ij} \leq a_j \qquad \forall j \in J, \tag{4.25}$$

$$\sum_{i \in J} x_{ij} \leq a_i \qquad \forall i \in J, \tag{4.26}$$

$$\sum_{j \in J} g_{ij} - \sum_{j \in J_0} g_{ji} = a_j \qquad \forall i \in J_0 \tag{4.27}$$

$$g_{ij} \leq (n-1) x_{ij} \qquad \forall i \in J_0, \quad \forall j \in J \tag{4.28}$$

$$x_{ij}, a_j \in \mathbf{B}, g_{ij} \in \mathbf{N} \qquad \forall i, j \in J \tag{4.29}$$

In which set J_0 is $J \cup 0$ and zero is the dummy job from which we start and end. The objective function (4.23) is a rewritten form of the pricing problem in (4.19). The dual variable for constraint (4.15) is neglected since it is the same for every machine schedule. Constraints (4.24) limit the duration of each schedule by the maximum workload per week per machine. Constraints (4.25) and (4.26) limit the number of predecessors of job j and successors of job i by the number of times a job is placed in the schedule. The Gavish and Graves constraints are shown in (4.27) and (4.28). Finally, constraints (4.29) give the domain of the decision variables. The MILP aims to find the schedule with the most negative reduced costs. However, we want to include all schedules with a negative reduced costs not just the most negative one. Therefore, we search the solution pool for all schedules with negative reduced costs.

The high number of binary and integer decision variables make the problem complex and may result in high computation times. High computation times could be due to the fact that the pricing problem needs to be solved in every iteration of the Column Generation algorithm. Therefore, the pricing problem is first solved using a greedy heuristic. Only when the greedy heuristic is no longer able to find a schedule with negative reduced cost, the pricing problem is solved using the MILP formulation.

The greedy heuristic is a method that aims to find a good feasible solution by making the best local choice. Algorithm 2 shows the pseudocode. We define the set J^- as the set of jobs that are not yet scheduled in the current schedule and S_{rc} as the set of schedules with negative reduced costs. The algorithm starts with scheduling job j as the start job of schedule_j. Then, the algorithm searches for the best job $i \in J^-$ to succeed job j and adds it to the schedule schedule_j. A job is the best successor if it improves the reduced costs the most. The algorithm finds the reduced costs rc' and the completion time c'_s of the temporary schedule using the *getreducedcosts* and *getcompletiontime* function respectively. In case the completion time of the schedule exceeds the maximum completion time or if $rc > 0$, no more jobs are added to schedule_j and schedule_j is added to S_{rc}. Additionally, all intermediate schedules are added to S_{rc}. We repeat this for every job $j \in J$. Thus, the output of the greedy heuristic consists of all sequences, including intermediate sequences, with a negative reduced cost. All columns with a

negative reduced cost are added to the RMP.

Algorithm 2: Pricing problem heuristic

Input: $S_{rc} \leftarrow \emptyset$

Result: Schedules with reduced costs

```
for all j ∈ J do
    J⁻ ← J \ j;
    start job ← j;
    schedule_j = j;
    rc ← getreducedcosts;
    c_s ← getcompletiontime;
    while c_s < k_mt and rc < 0 do
        for all i ∈ J⁻ do
            rc' ← getreducedcosts;
            c'_s ← getcompletiontime;
            if rc' < rc then
                best_i ← i;
                rc ← rc';
                c_s ← c'_s ;
            end
        end
        schedule_j ← best_i ∪ schedule_j;
        start job ← best_i;
        S_rc ← schedule_j;
        J⁻ ← J⁻ \ best_i;
    end
end
Return: S_rc
```

To start the Column Generation algorithm, a initial set of columns need to be determined. We find these columns by a greedy heuristic. The greedy heuristic randomly generates a number of acyclic schedules that do not exceed the time capacity. The initial column in the RMP influence the performance of the algorithm a lot. Despite the importance of a good initial solution, in this thesis we chose to randomly create job schedules j untill the maximum production time per week is reached.

When both the heuristic and the exact pricing problem are not able to generate any more schedules with negative reduced costs, the Column Generation is finished. The algorithm provides an optimal solution to the linear relaxation of the Set Partitioning problem. If the solution is integer, we find an optimal solution our original problem. Otherwise, some values are fractional. However, fractional solutions are not usable, thus we need to find an integer solution. The exact method to do this is performing Branch and Price, but we chose to use an heuristic approach, i.e. Column Generation heuristic. By initiating the original Set Partitioning problem with the solution found by Column Generation, we find a good feasible integer solution.

4.2.3 Metaheuristic for scheduling

Despite the heuristic that speeds up the Column Generation algorithm, computation times might still be large. This is due to the exact pricing problem that we need to solve to guarantee optimality. Therefore, we additionally study a metaheuristic for the scheduling problem. The chosen algorithm is Greedy Randomised Adaptive Search Procedure (GRASP). GRASP is a heuristic that applies local search on every solution constructed by the greedy randomised algorithm. A random solution is generated by a greedy heuristic, this yields a current solution x. We perform VND on the current solution which yields a new current solution x'. When the current solution is better than the best known solution, we update the best known solution. This continues until a stopping criteria is met. We define n as the number of jobs and m as the number of machines. The neighbourhoods in our VND algorithm for scheduling consist of swaps of one or multiple jobs on the same or different machines. We use the following neighbourhoods in the VND algorithm:

1. N_1: Add one job from one machine to another machine. This results in a neighbourhood of size $O(mn)$.
2. N_2: Find the schedules with the longest and shortest computation time. Add two jobs of longest machine schedule to the shortest machine schedule. This results in a neighbourhood of size $O(n^2)$.
3. N_3: Swap two jobs on one machine. This results in a neighbourhood of size $O(n^2)$.
4. N_4: Swap two jobs from two different machines. This results in a neighbourhood of size $O((mn)^2)$.
5. N_5: Swap three jobs from three different machines. This results in a neighbourhood of size $O((mn)^3)$.
6. N_6: Swap two jobs from one machine with two jobs from another machine. This results in a neighbourhood of size $O(m^2n^4)$.
7. N_7: Randomly shuffle one job sequence. This results in a neighbourhood of size $O(n!)$.

The greedy randomised heuristic randomly assigns each job to a machine, while taking the time capacity of each machine into consideration. The risk of a metaheuristic such as GRASP is that we find a local minimum instead of a global minimum. This happens when the neighbourhoods are not large enough to find better solution values. In contrary to the VND algorithm for lot-sizing, we use the steepest descent approach for neighbourhoods N_1 to N_6. In other words, we check all neighbours in $N(x)$ and accept the best solution found. Our search strategy for N_7 is different. Instead of checking all neighbours we check randomly selected neighbours and accept it if the objective value is improved. We check at most twenty random solutions.

Chapter 5

Parameter values & model assumptions

This chapter elaborates on the chosen parameters and assumptions for each algorithm. We first explain the data that is provided by Pieter Pot in Section 5.1. In Section 5.2 we specify the choices of the parameter values for the different algorithms. Additionally, we explain what assumptions are made to find good feasible solutions.

5.1 General data

The data used to test the algorithms consists of three parts: product specific information, inventory values and demand per product. The input data for Pieter Pot's models is constantly changing because of added SKUs, change in demand and so on. To validate the models equally, we use the data from the beginning of week 35. We plan the production for a period of thirty weeks. At the beginning of week 35 Pieter Pot offered 488 different products of which 397 are sold in jars. In our case we only plan for the SKUs that are sold in jars. The following known characteristics retrieved from the data are relevant for the algorithms:

- Jars per Order; the average amount of jars per order placed. We do not use these values specifically but it is used to characterise the SKUs.
- Category; product type A, B or C is assigned to each SKU based on the jars per order ratio. If the ratio is less than or equal to 0.5 a products belongs to category C, if it is larger than or equal to 0.15 it belongs to category A. All values in between result in category B. With these values we can easily see if a product is a fast, medium or slow runner.
- Demand; SKU specific forecasted demand in jars per week d_{jt}.
- Start inventory; inventory per SKU at the end of the previous week.
- Difficulty level; either level one, two or three is assigned to each SKU to declare the difficulty level of production. Products that are difficult to fill are for example honey and dates.
- Minimum production amount; based on the product category we assigned a minimum production amount to each SKU. The production amounts are 500, 200 and 100 for products of category A, B and C respectively.

- Change-over times b_{ij}; time needed to prepare for the production of product j with product i as predecessor. In consultation with one of the managers at a filling locations we set $b_{ij} = 0.25$ hours if job i and j have no conflicting allergies and $b_{ij} = 1.5$ hours otherwise.
- Time capacity per machine k_{mt}; the maximum amount of hours per week a machine is available to produce jars. We specify the exact time capacities per algorithm.
- Production time of one jar of item j on machine m f_{mj}; because production is currently done manually the time it takes to produce one jar of product j is not known. Therefore, we made an estimation of this value f_{mj}, varying per difficulty level.

5.2 Assumptions & parameter values

Pieter Pot's case is rather extensive and various factors have to be taken into consideration to ensure the models approximate reality. This makes the problem difficult and therefore we make some assumptions. This section considers the assumptions that were made for the integrated approach and in the scheduling algorithm. For all models, we explain our choice for the parameter values.

5.2.1 Integrated approach

We use some parameters concerning production requirements for the integrated approach. The time capacity for each machine is the same and is set on $k_{mt} = 40$ hours. This is based on the total time the people per work place are working. Due to the large number of decision variables in the MILP, solving it exactly is not possible within reasonable time. Therefore we use the Relax & Fix algorithm. We found that even with this heuristic approach the computation time is still too high. Thus, we used a small subset of the data set to test the performance. The moving window is set on one week since this best fits reality. We look at a time window of 15 weeks, to prevent computation times from increasing too much. Recall the sets J and N that contain the included SKUs and the number of micro periods respectively. To gain insights in the trade-off the sizes of J and N, we created three test instances. The sizes of sets J and N are shown in Table 5.1. We found these sizes of set N to be enough to produce each SKU in time. Increasing the size of set J too much results in an insufficient amount of production periods. On the other hand, increasing the amount of production periods, while maintaining the same amount of included SKUs, results in unused production periods. This has a negative impact on the computation times. Set N for instance 1 consist of one element, meaning each machine performs at most one job per week. Therefore, there is no need to schedule the jobs since we schedule on a weekly basis. Nevertheless, we include this instance to test the performance of the algorithm.

	Instance 1	Instance 2	Instance 3
Size set J	20	20	50
Size set N	1	3	4

Table 5.1: Size of test instances

We view the people who work at one social workplace as one machine, hence we define 5 identical machines. For simplicity we define each set G_j to be all machines. Additionally, the sets A_m contain all products for each $m \in M$. We define the change-over costs s_{ij} as zero if the change-over is not dependent on allergies and 38 otherwise. Based on various tests with different parameter value configurations we found the values shown in Table 5.2 to be best.

Parameter	h_j	out of stock A	out of stock B	out of stock C	percentage w
Value	0.02	10000	10000	10000	5

Table 5.2: Parameter values for the integrated approach

5.2.2 Lot-sizing

We do not accept assumptions that make the lot-sizing model differ too much from reality since the model will be used by Pieter Pot. We included the smoothing constraints (4.6) and (4.7) in the MILP because of the importance of balancing the production quantities. However, the percentage that restricts the differentiation in production quantities per week is a parameter value that we need to determine. In consultation with Pieter Pot we chose a five percent interval. Other parameter values are the holding costs h_j, the out of stock costs u_j and the set-up costs p_j. Increasing h_j decreases the benefit of having high inventory values. On the other hand, increasing p_j increases the benefit of having high inventory values. Out of stock costs must be high at any time since we want to minimise the lost sales. Based on various tests with different parameter value configurations we found the values shown in Table 5.3 to be best.

Parameter	h_j	p_j	out of stock A	out of stock B	out of stock C	percentage w
Value	0.04	200	10000	10000	10000	5

Table 5.3: Parameter values for lot-sizing

We use the same parameter values for the VND algorithms. To initialise the VND algorithms we perform branch and bound on the MILP and stop it after a certain number of MILP solutions is found. The trade-off is that performing a large number of iterations results in high computation times for the CPLEX solver. On the other hand, performing little iterations may result in performing many iterations of the VND algorithms, which again increases the computation times. We find the same trade-off in choosing the number of iterations for checking the feasibility of the changed solution. We found 10 to be the best number of iterations for the initialisation and we used no limit for checking the feasibility.

5.2.3 Scheduling

We made a few assumptions to schedule all jobs that are planned by the lot-sizing MILP. First, we view the people who work at one social workplace as one machine. In reality, they work in smaller groups, which would increase the amount of machines but decreases the total available

time of the machine. All machines are considered to be identical. Meaning, they have the same capacities and speed of production.

We used three different instances that consist of different sets of jobs. With these instances we check the performance of the Column Generation algorithm. The first instance consists of random SKUs with random production quantities. The other two are output from the lot-sizing problem. Many schedules with the same reduced cost and completion time are possible. This is because of little differences in change-over times. Therefore, we only add the schedules with different reduced costs. The time capacity for each machine is initially set on $k_{mt} = 40$ hours. This is based on the total amount of jars the people per work place are able to produce. However, 40 hours is not always enough time to find a feasible initial solution for larger instances. Hence, we increased the capacity. This practically means more people are needed at each filling location. The size of the instances and the time capacity for the three instances can be found in Table 5.4.

Instance	Instance size	Time capacity
1 (test)	10	40
2 (W40)	20	70
3 (W50)	20	80

Table 5.4: Instance & set size for three instances

Currently, we consider five machines as this is the amount of social workplaces Pieter Pot is currently producing at. Thus, we set $m_{max} = 5$. We use the same instances and parameter values to test the GRASP algorithm. The stopping criteria used is a time limit of two minutes.

Chapter 6

Results

This chapter elaborates on the results obtained by the simultaneous and sequential approach as discussed in Chapters 3 and 4. Based on the objective values and different performance indicators the methods are evaluated independently in Sections 6.1 and 6.2.

6.1 Results Integrated approach

Recall the three requirements provided by Pieter Pot for the lot-sizing problem. The first requirement concerns the deviation in production quantities per week. These values are relatively high for the small instances used. This is the result of including only a small amount of SKUs in each instance. For example, if in one week no production occurs and we plan production of two SKUs in the following week, the differentiation is rather large. The second requirement concerns minimising production of the same SKU in consecutive weeks. Due to the small instance sizes we found production in consecutive weeks does not occur. Thus, we do not show them in the results. Finally, the aim is to minimise out of stocks. This requirement is used to evaluate the integrated approach.

Table 6.1 shows the results obtained for the MILP using the commercial solver CPLEX with a maximum running time of thirty minutes. We view the objective value, maximum number of out of stocks, the lower bound, the optimality gap and the computation time for the three instances. Recall that the instances differ in set size J and N and we chose sizes 20 and 1, 20 and 3 and 50 and 4 respectively. We found an optimal solution for both instance 1 and 2 within the thirty minute time limit. We did not find an optimal solution for the third instance. The best found feasible solution for instance three has an optimality gap of 4.29 from it's lower bound, i.e. 429 percent.

Integrated approach (MILP)	Instance 1	Instance 2	Instance 3
Max out of stocks	0	0	0
Objective value	1809	1804	7317
Computation time (s)	127	1536	1800
Bound	1809	1804	1384
Optimality gap	0	0	4.29

Table 6.1: Results integrated approach

Table 6.2 shows the results using Relax & Fix. Both methods return no out of stocks in the solution for all instances. The Relax & Fix algorithm found feasible solutions close to the optimum within a much smaller computation time compared to the MILP method for instance 1 and 2. For instance 3 Relax & Fix outperformed the MILP on both computation time and objective value.

Integrated approach (Relax & Fix)	Instance 1	Instance 2	Instance 3
Max out of stocks	0	0	0
Objective value	1864	1832	5856
Computation time (s)	39	102	1191

Table 6.2: Results integrated approach Relax & Fix

6.2 Results Sequential approach

We view the results of the two parts of the sequential approach separately. We first compare the results of the MILP and both VND algorithms for the lot-sizing problem. Afterwards, we evaluate the two algorithms that aim to find optimal schedules.

6.2.1 Lot-sizing

Again recall the three requirements provided by Pieter Pot for the lot-sizing problem. We ruled out outliers in production quantities by including this requirement in the constraints of the MILP. To see the influence of the smoothing constraints we also solved the MILP without these constraints. The VND algorithms do not bound the outliers in production quantities with constraints. Hence, we include them as a performance indicator. The second requirement of minimising production of the same SKU in consecutive weeks and the third requirement of minimising out of stocks while maintaining low inventory, are also included as a performance indicator. We show the results of the three solution methods: MILP with and without smoothing constraints and VND(diff). We did not include VND(obj) since the the initial solution of the VND algorithm after 10 iterations is already quite good in terms of objective value. Therefore, we found that VND(obj) does not improve the objective value.

Because of the linear neighbourhood sizes explained in Chapter 4, the sequence in which the algorithm visits the neighbourhoods is ambiguous. Hence, we review different sequences of the defined neighbourhoods and found that most improvements are made in neighbourhoods 2, 3

and 4. N_1 almost never improves the cumulative difference. We found the following sequences to be best:

- Sequence 1: $N_3 - N_4 - N_2 - N_1$
- Sequence 2: $N_4 - N_3 - N_2 - N_1$
- Sequence 3: $N_2 - N_3 - N_4$

Graph 6.1 shows the progress of the cumulative differences over time for sequences 1, 2 and 3. VND(diff) has a high computation time and therefore we set a time limit of 1 hour. We see that the progress over time is quite similar for the three sequences.

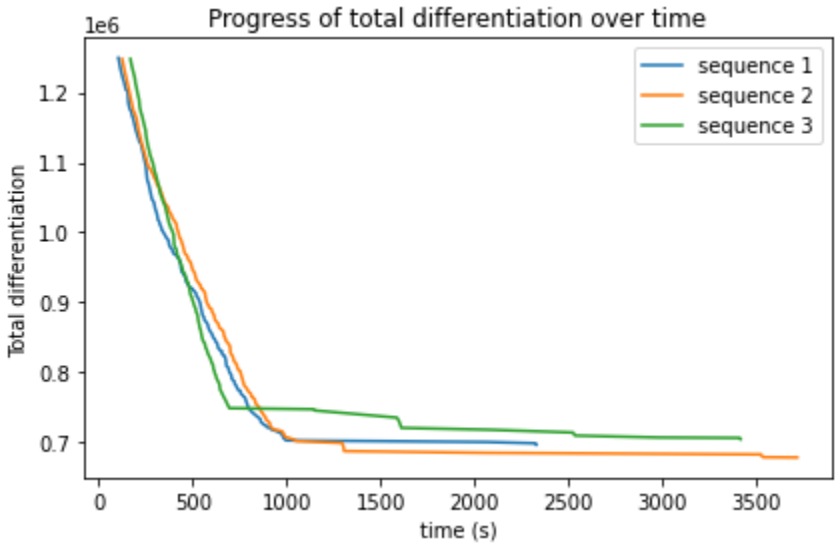


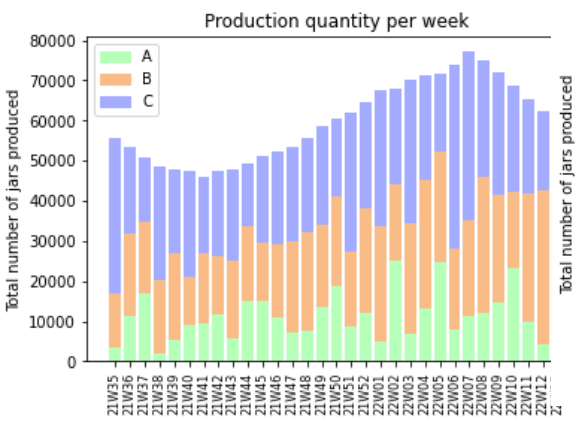
Figure 6.1: Cumulative differences over time for three different neighbourhood sequences

Based on Graph 6.1 sequence 2 performs the best. However, based on the other performance metrics sequence 3 performs the best hence we include the results of sequence 3 for comparison with the other methods. Table 6.3 shows the results for the methods MILP with smoothing constraints, MILP without smoothing constraints and VND(diff). Some out of stocks occur in all three methods. This is due to the fact that we use real data from Pieter Pot which results in some bulk product not being delivered on time or being unavailable. These type of out of stock can always occur. The MILP with smoothing constraints performs the best based on the lowest maximum deviation. Based on the objective value and computation times MILP without smoothing constraints performs the best. However, the maximum deviation in production quantities is more than 40 times as high as in MILP with smoothing constraints. VND(diff) performs average in terms of maximum deviation. However, the objective, the average inventory costs and the computation time are the highest. In terms of the number of times production occurs in consecutive weeks all methods perform approximately the same.

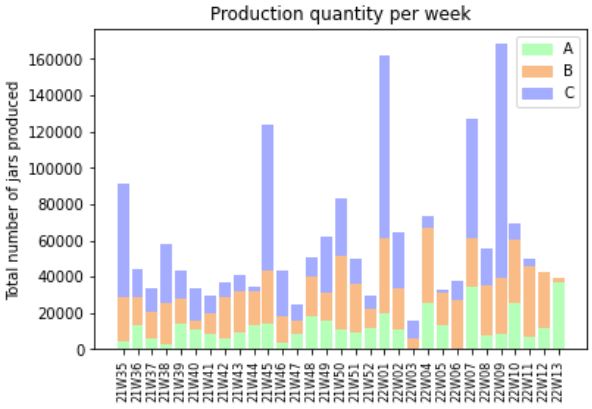
Lot-sizing	MILP (smoothing)	MILP (no smoothing)	VND(diff)
Max deviation in production amount	3199	133007	41359
Max out of stocks	15	15	15
Production in consecutive weeks	1	0	2
Objective value	51818824	51780436	53268699
Lower bound	51731476	51778655	-
Optimality gap	0.0017	0.000034	-
Average inventory costs	364449	368878	381137
Computation time (s)	3600	309	3845

Table 6.3: Results of MILP and VND(diff)

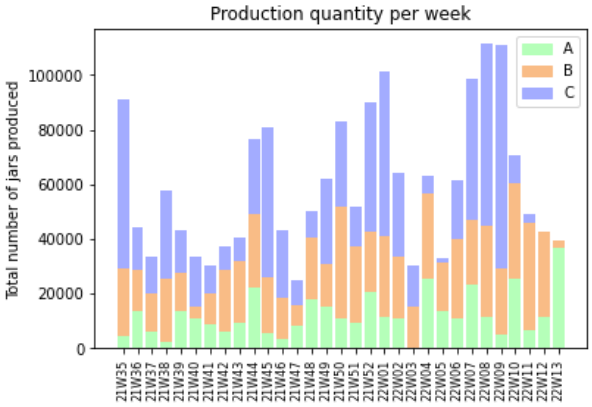
The figures in 6.2 show production amounts in jars for VND(diff), the MILP with and without smoothing constraints. We clearly see the influence of the smoothing constraints on the MILP. By not including the smoothing constraints large outliers in production quantities occur. The maximum deviation in production amount is decreased with 69% by VND(diff) compared to the MILP without smoothing constraints but large outliers in production quantities are still present.



(a) MILP: Production quantities



(b) MILP (no smoothing): Production quantities



(c) VND(diff): Production quantities

Figure 6.2: Production quantities for weeks 21W35 to 22W13

6.2.2 Scheduling

No requirements were given with regard to the scheduling problem. Hence, the metrics we use are the total completion time and the computation time. To validate the results of the Column Generation heuristic we calculate the optimality gap with the optimal solution to the linear relaxation of our problem (LP). The IP and LP objectives for the Set Partitioning problem are the same, leading to an optimality gap of zero. Thus, we can calculate the optimality gap for the GRASP algorithm with the solution provided by the Column Generation algorithm. Table 6.4 and 6.5 show the results of using Column Generation and GRASP respectively. Table 6.4 shows that GRASP returns the same objective value per instance.

Metric	Instance 1	Instance 2	Instance 3
Objective (LP)	87.6	221.1	212.5
Objective (IP)	87.6	221.1	212.5
Computation time (s)	0.67	2.52	4.20
Columns generated	164	1160	802
Optimality gap	0	0	0

Table 6.4: Results Column Generation heurisitc

Metric	Instance 1	Instance 2	Instance 3
Objective	87.6	221.1	212.5
Computation time (s)	120	120	120
Optimality gap	0	0	0

Table 6.5: Results GRASP

Comparing the computation times can not be done due to the fixed time limit. That is why we chose to analyse the times at which improvements in the objective value are made by the GRASP algorithm. Figure 6.3 shows the results up till the point in time where the best solution was found. We did not include instance 1 since this instance is not based on real data and because it is relatively small. The figure shows that the GRASP algorithm is able to find a good feasible solution within 1 second.

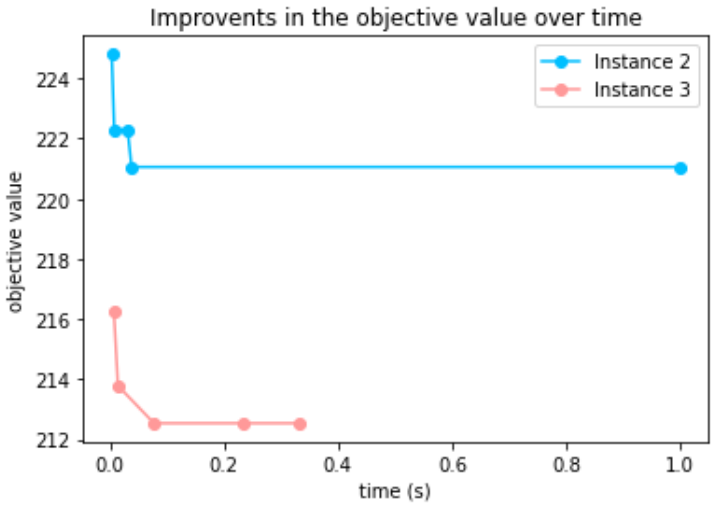


Figure 6.3: Objective values over time for instance 2 and 3

In order to validate the performance of the different neighbourhoods in the algorithm, we keep track off the number of times a neighbourhood search results in an improvement. Table 6.6 clearly shows that neighbourhoods 2, 5, 6, 7 do not result in improvements for all instances. Neighbourhood 1 is the most effective for all instances.

Instances	N1	N2	N3	N4	N5	N6	N7
1	55254	0	24194	47805	0	0	0
2	15680	0	5314	8670	0	0	0
3	12709	0	4455	8405	0	0	0

Table 6.6: Improvements per neighbourhood

In Figure 6.4 we show one of the schedules made using the Column Generation heuristic. Schedules differ each time the algorithm is run. This is because multiple sequences are possible that result in the same total completion time.

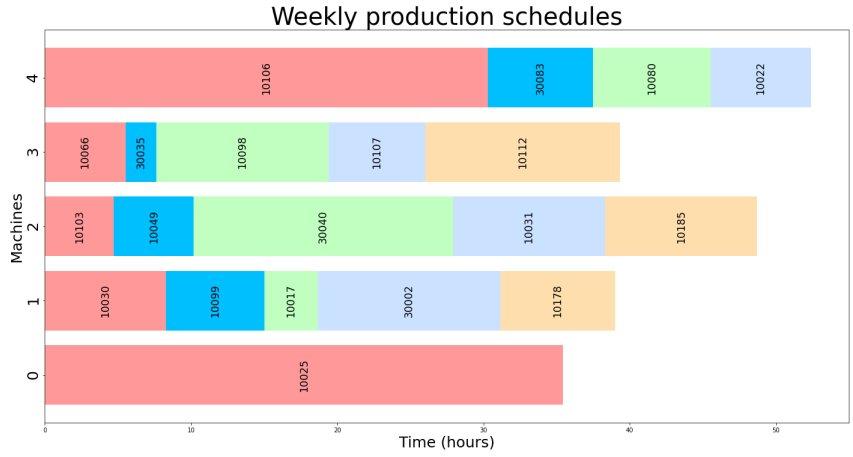


Figure 6.4: Job schedule for instance 2

Chapter 7

Conclusions

In this research we evaluated what methods are the most suitable for optimising the production of online circular grocer Pieter Pot. The goal was to answer the following research question:

What algorithm is the best for optimising the production planning and scheduling of Pieter Pot?

We divided the production planning in two stages: lot-sizing and scheduling and chose two main approaches to optimise these stages. The first approach focused on simultaneously solving the lot-sizing and scheduling problem. Based on prior research we found that solving the problems simultaneously resulted in high computation times due to the large amount of decision variables. Therefore, we chose to use a Relax & Fix heuristic. For instance 1 and 2 this resulted in a decrease in computation time of 69% and 93% respectively. Additionally, Relax & Fix found good objective values for all instances. For larger instance sizes Relax & Fix is able to find solutions closer to the optimum within less time than the MILP. Thus, we can conclude that Relax & Fix is a good alternative for solving these type of problems. However, we can only conclude this for a small subset of all SKUs.

The second approach solved the two stages sequentially. Both stages are solved using a MILP approach as well as a heuristic approach. For lot-sizing we found that solving the problem using an MILP with smoothing constraints meets the requirements of averaging production quantities in consecutive weeks best. We found a good feasible solution within sixty minutes with a differentiation in production in consecutive weeks of less than 5 percent. The computation time for MILP without smoothing constraints is much smaller but there is large differentiation in production quantities. This is an important model requirement so we prefer the MILP with smoothing constraints. VND(diff) decreased the differentiation in production quantities compared to the MILP without smoothing constraints. However, it performed the least on all other performance metrics compared to both the MILP with and without smoothing constraints.

We first solved the scheduling problem using a Column Generation heuristic. Because of the relatively small amount of jobs that needed to be scheduled, the computation times are reasonable. In our problem the Column Generation heuristic provided optimal solutions. However, this can not be guaranteed. Additionally, if the number of jobs increases, the computation time will also increase. This is mostly due to the exact pricing problem. In that case, the GRASP

heuristic is a good alternative. The three test instances found the same objective values within one second. This means that the algorithm was able to escape local minima in order to find the global minimum. We can conclude that GRASP in combination with VND and the specified neighbourhoods is a good alternative for finding optimal schedules.

Chapter 8

Recommendations

During this research, we noted multiple ideas that are interesting to study in further research because it could improve optimisation of production planning and scheduling. We divide our recommendations into three categories: the integrated approach, lot-sizing and scheduling.

Recommendations concerning the integrated approach:

- **Improvements Relax & Fix**
  In literature, Relax & Fix algorithms are widely studied and have proven to be a good solution approach. However, Relax & Fix algorithms only work for relatively small instance sizes. In our problem the algorithm might be improved by allowing more than one job to be performed in each micro period. This might speed up the algorithm despite having so many decision variables.

- **Other types of heuristics**
  We chose to use a mathematical programming based heuristic to find a good feasible solution to our problem. In the literature, however, we found other heuristic algorithms that do not need a MILP formulation, such as genetic algorithms. Implementing this heuristic and comparing the performance might be interesting.

- **Add smoothing constraints**
  The model requirement concerning minimising differentiation in production in consecutive weeks is not considered in the integrated approach. This is because we only look at a small subset of the SKUs. However, if Pieter Pot considers to improve the integrated approach so that it can be used, it might be interesting to add constraints regarding this requirement.

Recommendations concerning lot-sizing:

- **Escaping local optima**
  We chose to use four linear neighbourhoods in the VND algorithm for lot-sizing. To improve the results of both VND(obj) and VND(diff) it may be interesting to implement neighbourhoods of different sizes. This way we could eliminate local optima.

- **Dynamic programming**

We chose an MILP and heuristic approach to solve our problem. However, dynamic programming is an approach that often finds optimal solutions quicker. Follow-up studies might compare the computation times of solving our problem using a MILP model and dynamic programming.

- **Other types of heuristics**
  Next to the metaheuristic used, there are various other heuristics that might be interesting for solving our problem. Implementing a tabu-search algorithm may speed up computation times since weak solutions are skipped in a predefined number of iterations.

Recommendations concerning scheduling:

- **Dynamic programming for pricing problem**
  We used a MILP model to guarantee an optimal solution to our pricing problem in the Column Generation algorithm. However, another approach that is often used is dynamic programming. Dynamic programming algorithms are often faster in finding an optimal solution. Therefore, the pricing heuristic could be neglected. Follow-up studies might compare the results of this approach to our pricing approach.

- **Branch & Price**
  In order to guarantee an optimal solution using Column Generation, a Branch and Price algorithm can applied instead of a Column Generation heuristic. This however might increase computation times.

- **Non-identical machines**
  An addition to our current scheduling problem could be to view the machines as non-identical. This way the problem better resembles reality and we could obtain more precise schedules.

- **Change neighbourhoods of VND in GRASP**
  We found that not all neighbourhoods used in the VND algorithm improved the solution. Therefore, the neighbourhoods can be changed to more efficient neighbourhoods. It might be interesting to see how this updated GRASP algorithm performs on bigger instance sizes compared to our GRASP algorithm.

References

Al-Salem, A. (2004). Scheduling to minimize makespan on unrelated parallel machines with sequence dependent setup times. *Engineering Journal of the University of Qatar*, *17*(1), 177–187.

Arnaout, J.-P., Musa, R., & Rabadi, G. (2014). A two-stage ant colony optimization algorithm to minimize the makespan on unrelated parallel machines—part ii: enhancements and experimentations. *Journal of Intelligent Manufacturing*, *25*(1), 43–53.

Arnaout, J.-P., Rabadi, G., & Musa, R. (2010). A two-stage ant colony optimization algorithm to minimize the makespan on unrelated parallel machines with sequence-dependent setup times. *Journal of Intelligent Manufacturing*, *21*(6), 693–701.

Bazrafshan, R., Hashemkhani Zolfani, S., & Al-e hashem, S. M. J. (2021). Comparison of the sub-tour elimination methods for the asymmetric traveling salesman problem applying the seca method. *Axioms*, *10*(1), 19.

Beraldi, P., Ghiani, G., Grieco, A., & Guerriero, E. (2008). Rolling-horizon and fix-and-relax heuristics for the parallel machine lot-sizing and scheduling problem with sequence-dependent set-up costs. *Computers & Operations Research*, *35*(11), 3644–3656.

Buschkühl, L., Sahling, F., Helber, S., & Tempelmeier, H. (2010). Dynamic capacitated lot-sizing problems: a classification and review of solution approaches. *Or Spectrum*, *32*(2), 231–261.

Chen, Z.-L., & Powell, W. B. (2003). Exact algorithms for scheduling multiple families of jobs on parallel machines. *Naval Research Logistics (NRL)*, *50*(7), 823–840.

Dantzig, G., Fulkerson, R., & Johnson, S. (1954). Solution of a large-scale traveling-salesman problem. *Journal of the operations research society of America*, *2*(4), 393–410.

Diana, R. O. M., de França Filho, M. F., de Souza, S. R., & de Almeida Vitor, J. F. (2015). An immune-inspired algorithm for an unrelated parallel machines' scheduling problem with sequence and machine dependent setup-times for makespan minimisation. *Neurocomputing*, *163*, 94–105.

Fandel, G., & Stammen-Hegene, C. (2006). Simultaneous lot sizing and scheduling for multi-product multi-level production. *International Journal of Production Economics*, *104*(2), 308–316.

Fanjul-Peyro, L., Ruiz, R., & Perea, F. (2019). Reformulations and an exact algorithm for unrelated parallel machine scheduling problems with setup times. *Computers & Operations Research*, *101*, 173–182.

Ferreira, D., Morabito, R., & Rangel, S. (2009). Solution approaches for the soft drink integrated production lot sizing and scheduling problem. *European Journal of Operational Research*, *196*(2), 697–706.

Ferreira, D., Morabito, R., & Rangel, S. (2010). Relax and fix heuristics to solve one-stage one-machine lot-scheduling models for small-scale soft drink plants. *Computers & Operations Research*, *37*(4), 684–691.

Fleischmann, B., & Meyr, H. (1997). The general lotsizing and scheduling problem. *Operations-Research-Spektrum*, *19*(1), 11–21.

Fleszar, K., Charalambous, C., & Hindi, K. S. (2012). A variable neighborhood descent heuristic for the problem of makespan minimisation on unrelated parallel machines with setup times. *Journal of Intelligent Manufacturing*, *23*(5), 1949–1958.

Garey, M. R., Johnson, D. S., & Sethi, R. (1976). The complexity of flowshop and jobshop scheduling. *Mathematics of operations research*, *1*(2), 117–129.

Gomez Urrutia, E. D., Aggoune, R., & Dauzère-Pérès, S. (2014). Solving the integrated lot-sizing and job-shop scheduling problem. *International Journal of Production Research*, *52*(17), 5236–5254.

Helber, S., & Sahling, F. (2010). A fix-and-optimize approach for the multi-level capacitated lot sizing problem. *International Journal of Production Economics*, *123*(2), 247–256.

Karimi, B., Ghomi, S. F., & Wilson, J. (2003). The capacitated lot sizing problem: a review of models and algorithms. *Omega*, *31*(5), 365–378.

Kuik, R., Salomon, M., Van Wassenhove, L. N., & Maes, J. (1993). Linear programming, simulated annealing and tabu search heuristics for lotsizing in bottleneck assembly systems. *IIE transactions*, *25*(1), 62–72.

Lopes, M. J. P., & de Carvalho, J. V. (2007). A branch-and-price algorithm for scheduling parallel machines with sequence dependent setup times. *European journal of operational research*, *176*(3), 1508–1527.

Meyr, H. (2000). Simultaneous lotsizing and scheduling by combining local search with dual reoptimization. *European Journal of Operational Research*, *120*(2), 311–326.

Meyr, H. (2002). Simultaneous lotsizing and scheduling on parallel machines. *European Journal of Operational Research*, *139*(2), 277–292.

Rabadi, G., Moraga, R. J., & Al-Salem, A. (2006). Heuristics for the unrelated parallel machine scheduling problem with setup times. *Journal of Intelligent Manufacturing*, *17*(1), 85–97.

Rohaninejad, M., Kheirkhah, A., & Fattahi, P. (2015). Simultaneous lot-sizing and scheduling in flexible job shop problems. *The International Journal of Advanced Manufacturing Technology*, *78*(1-4), 1–18.

Spliet, R. (2020). *Mp set 2 column generation.* University Lecture. (Accessed: 2021–25-05)

Stadtler, H. (2003). Multilevel lot sizing with setup times and multiple constrained resources: Internally rolling schedules with lot-sizing windows. *Operations Research*, *51*(3), 487–502.

Tempelmeier, H., & Helber, S. (1994). A heuristic for dynamic multi-item multi-level capacitated lotsizing for general product structures. *European Journal of Operational Research*, *75*(2), 296–311.

Toledo, F. M. B., & Armentano, V. A. (2006). A lagrangian-based heuristic for the capacitated lot-sizing problem in parallel machines. *European Journal of Operational Research*, *175*(2), 1070–1083.

Van den Akker, M., Hoogeveen, H., & Van De Velde, S. (2005). Applying column generation to machine scheduling. In *Column generation* (pp. 303–330). Springer.

Van den Akker, M., Hoogeveen, H., & van de Velde, S. L. (1999). Parallel machine scheduling by column generation. *Operations Research*, *47*(6), 862–872.

Van den Heuvel, W. (2020). *Math programming heuristics.* University Lecture. (Accessed: 2021–27-05)

White, C. H., & Wilson, R. C. (1977). Sequence dependent set-up times and job sequencing. *The International Journal of Production Research*, *15*(2), 191–202.

Xiao, J., Zhang, C., Zheng, L., & Gupta, J. N. (2013). Mip-based fix-and-optimise algorithms for the parallel machine capacitated lot-sizing and scheduling problem. *International Journal of Production Research*, *51*(16), 5011–5028.