Erasmus University Rotterdam

Erasmus School of Economics

Master Thesis Operations Research and Quantitative Logistics

Using different accuracies of an MILP in a matheuristic

Jeroen Keijzer
412958

Supervisor: A.P.M. Wagelmans

April 30, 2020

Contents

Chapter 1

Introduction

Often, when an operations research problem becomes more realistic, it becomes harder to capture. Aspects which seem too complex to capture correctly in an optimization algorithm are often left out or approximated such that a problem can be formulated that can be solved within reasonable time. In practice, problems need to be solved that are both very large in scale and contain aspects that are difficult to translate into a Mathematical Program.

In this thesis, we look into the concept of matheuristics. Matheuristics combine a certain metaheuristic with a Mathematical program. More specifically, we look at a Neighborhood Search that uses an MILP to solve the subproblems for a specific neighborhood. From a practical point of view, this approach has often yielded good results for large and complex problems. However, because of the complex nature of these problems, a decision needs to be made in how the MILP for subproblems is formulated. Let us call a subproblem MILP completely accurate when all aspects of the main problem can be captured correctly. This means that when the result of the MILP is translated back into the bigger problem, the objective values of the MILP would be the same as in the entire problem. This would mean that such an MILP would always find the best possible solution for all variables that are free to change given that the MILP could solve the subproblem to optimality. However, it might not be ideal to have a completely accurate MILP for very complex problems. If it is possible to simplify an MILP such that it becomes less accurate but quicker to solve, it could yield better results for the entire matheuristic. Therefore, there is a trade off between the accuracy of an MILP and the execution speed of this MILP. Given the accuracy and performance of an MILP, the size of a neighborhood can be chosen. This neighborhood would become the subproblem that needs to be solved by the mathematical program. Having a larger neighborhood could produce a more influential solution, as the subproblem would get closer to the original problem that needs to be solved. However, having a larger neighborhood means that a subproblem is created that takes more time to be solved. Often, there is a ‘sweet spot' for which the mathematical programs can be solved quickly whilst not being small enough such that it becomes very hard to find any significant improvements. When these two decisions are made optimally, it can significantly improve the performance of the entire algorithm.

It can be argued that at different points in time in the optimization process, different neighborhoods and accuracies would give the best solutions. In the literature, there is already quite some research done with regard to finding the right neighborhoods for such an optimization process, but not much research has been done regarding accuracy. This thesis will study the impact of having flexibility in the accuracy within an optimization process. This means using multiple MILP formulations for a problem where a decision is made which formulation to use at a specific point in time. A distinction can be made between a more relaxed, quicker formulation which does not entirely capture the problem correctly and a more sophisticated formulation which is slower but is more likely to give a solution that can more accurately be translated back to the entire problem.

1.1 Literature review

This thesis investigates an extension of the variable neighborhood search (VNS) by Hansen and Mladenović (2003). They use multiple neighborhood structures to escape from local optima. Different structures may focus on different obstacles of the main problem. Therefore, multiple neighborhood structures can find optima that a single structure may never be able to find. A good example of this algorithm can be found in Burke, De Causmaecker, Petrovic, and Berghe (2003), where VNS is combined with several metaheuristcs to optimize a nurse rostering problem. Della Croce and Salassa (2014) have created an algorithm for the same problem. However, they combine the VNS algorithm with an MILP. They use the MILP to solve a neighborhood where a number of variables are made free. Another example where VNS is combined with a mathematical program is James and Almada-Lobo (2011), where an iterative neighborhood search is proposed using an MIP for a parallel machine capacitated lot-sizing and scheduling problem. Once a local minimum is found, they switch to a different type of neighborhood such that the local minimum can be escaped from. Using an MIP for the local search phase of the algorithm allows them to always continue with a solution that is better than or equal to their previous solution. Della Croce, Grosso, and Salassa (2014) use a similar procedure to solve the two-machine total completion time flow shop problem.

These are all cases where an MILP is used to solve the problems in the neighborhood. However, other than changing the neighborhood in which they optimize, they do not change their MILP specifically. Erromdhani, Jarboui, Eddaly, Rebai, and Mladenovic (2017) Is an example where this does happen. For the capacitated lot-sizing problem with time windows and setup times, two distinct formulations are used in a variable neighborhood search. These formulations focus on a different aspect of the problem, such that they are used complementarily.

These are all cases for which the result of the MILP that is used in the neighborhood, gives a optimal view for that specific neighborhood. However, if a problem is non-linear and difficult to solve, this might not work as well anymore. Woo and Kim (2019) look into this. They created a matheuristic for solving a specification of the hydrogen supply chain network problem. They formulated an MINLP which would be relaxed into an MILP and solved using a genetic algorithm. This causes their objective function to not be entirely correct anymore and they have therefore built in an evaluation process when returning to the main stage.

1.2 Problem Description

In this thesis, the concept of accuracy in matheuristics will be studied. To investigate this, a supply chain problem with inventory and capacity constraints is considered. Given external demands for certain products, these products will need to be made out of different products. To make these products, other materials might be needed. In this way, an entire supply chain is constructed which builds and constructs products that can eventually be delivered to a customer. We optimize the entire supply chain from raw material to end product.

Figure 1.1: Flow chart for a supply chain

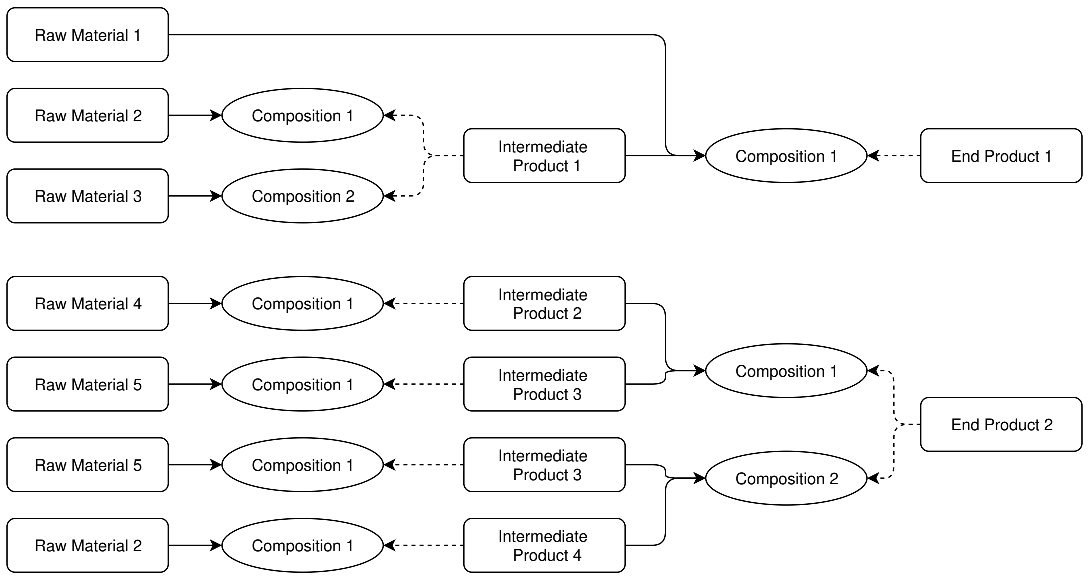


Figure 1.1 Gives a more visual interpretation on how the supply chains for products are organized. There are three types of products. Raw materials, intermediate products and end products. A raw material is a product which is only used to construct other products from.

It does not need products of its own to be created. Intermediate products are products which need other products to be made and it is also an input material for other products. Lastly, end products are the products that are sold to the customers. These products need input materials to be made and are not needed for other products. For a given intermediate or end product, there may be multiple ways to make these. A composition is a way to make a certain product. This means that if there are multiple compositions for a certain product, there are multiple ways to acquire the product. Therefore, a decision needs to be made regarding what composition will be used to make the product. All the dotted arrows indicate that such a decision needs to be made. When a certain composition is chosen, we know which input materials are needed for that composition. All solid arrows indicate flow from input materials to a certain composition. These two supply chains for the two different products seem totally distinct but this is actually not the case. Raw Material 2 can be chosen in both supply chains. Therefore, these supply chains are actually intertwined. On a larger scale these supply chains are even more intertwined such that raw materials can be used for multiple end products.

For certain products, there is a certain inventory that needs to be managed. For these products, there is a target for how many items are desired to be in stock. This target inventory does not have to be the same during the entire period. Next to this, there are resources that are used in the production process of a product. Some of these resources have a certain capacity. For example, in a certain time frame, a machine can only create a limited number of products. Next to this, a resource can be used by multiple products. For example, in Figure 1.1, to make intermediate product 1 and intermediate product 2, the same resources might be needed. This brings an extra dimension in the sense that distinct supply chains are even more intertwined with each other. Using the full capacity of a resource to create intermediate product 2 may result in intermediate product 1 not being able to be produced. Managing the capacities of these resources is a big challenge in the real world. Often, there are bottleneck resources that may hold up an entire supply chain and must therefore be planned as efficiently as possible.

There are several penalties regarding the external demands. These are either for delivering the demand later than requested or delivering less than is desired. In general, we want the supply chain of a certain external demand to take as little time as possible. This means that we incur penalties when a supply chain takes longer than needed. This is such that products do not stay in an unfinished state for long periods of time. The detailed MINLP, together with a more detailed explanation, is described in Section 2.1. This main problem will be referred to as the MINLP.

The main research question of this thesis is to investigate whether using multiple MILP formulations within a VNS improves the performance of an optimization process. What would be the effect of executing multiple MILP accuracies and at which points in time would it be useful to use which MILP? Next to this, what is the effect of re-executing the same neighborhood within the VNS with a more accurate MILP and after what kinds of results is it useful to perform a re-execution?

For the thesis, the following steps will be taken to investigate the research question. First of all, a neighborhood selection process needs to be created. The current MINLP will be reformulated into multiple MILPs according to those neighborhoods. Two different MILPs will be considered, a more simple MIP that can be solved more quickly and a more sophisticated MILP that is closer to the MINLP but is harder to solve. These formulations will be tested separately and in various combinations.

1.3 Data overview

In this section the datasets that will be used in the experiments are described. Every dataset has different characteristics that causes the problem that needs to be solved to be different. The main characteristics of these data sets are shown in Table 1.1.

Table 1.1: Details of the four different datasets. Demands is the number of external demands in this dataset. Depth refers to the maximum depth of a supply chain. Resources refers to the number of resources that can be used to create certain products. Products refers to the number of products that can be created within the supply chain. Compositions shows the number of possible compositions. If the number of compositions is equal to the number of products, all products can only be created in one way.

Dataset	Demands	Depth	Resources	Products	Compositions
1	446	3	13	12	15
2	924	3	11	52	61
3	165	9	7	72	74
4	562	3	20	246	246

All the datasets are either real data or data that is supposed to mimic real world supply chain problems. The first data set is a fabricated dataset that resembles a factory that produces aluminum products. An example would be the aluminum used for aluminum cans. To create such an end product, the material needs to pass several machines, sometimes the same machine multiple times. Furthermore, in this dataset there are relatively little products and compositions that can be created. For a certain end product, the supply chain leading up to that product is fixed and no choices regarding compositions can be made. However, In Table 1.1, we can see that there are more compositions than products. This is because for some products, a decision can be made to purchase this product externally or to produce it from within the current supply chain. Next to delivering the products in time, a bottleneck or this dataset is the handling of over usage of resources.

The second dataset resembles a steel factory where steel products are produced. Again, we can see that there are more compositions than products. There are 52 products and 61 compositions. Contrary to in the first dataset, In this case more compositions than products means that there are multiple ways to make a certain product. It is not a decision between purchasing a product or making the product within the factory. The decision on which path to take to fulfill all demands is more prominent in this scenario. Next to this, there is a relatively large number of external demands that need to be fulfilled. Together with finding the optimal paths for products this is a bottleneck. Another feature for this dataset is that we do not look at the inventories. There is no target inventory that needs to be adhered to.

The third dataset mimics the production process of a car assembly manufacturer. A lot of different input materials are needed to create cars. All these input materials need to be in place at the right times. This results in a deep and complicated supply chain. We can see in Table 1.1 that there is a large number of products. Only 6 of these 72 are end products. All the other products are used in the process of creating a car. For this dataset, the challenges are to handle all the input materials well and use the limited resources to make all cars in time. Because of the complex supply chain, one can imagine that a small change in a certain supply chain can have a large effect on the entire plan.

The fourth data set resembles the supply chain for aluminum products just like dataset 1. We can see in Table 1.1 that the number of products is significantly larger in this dataset than in dataset 1. This dataset mimics the supply chain over two different factories instead of one. The products and compositions are defined in such a way that for all products, only one possible path through the supply chain can be taken. Therefore, the number of compositions equals the number of products. Although all products are different there are only 20 resources. The products are created using similar resources and this means although they are different the supply chains are very much connected with each other.

Chapter 2

Methodology

In this section, the methodology is described. First, the MINLP formulation is explained. After that, the methodology behind the neighborhood search is described.

2.1 MINLP formulation

In this section the MINLP is described in detail. All sets, parameters and variables are explained first and afterwards all contraints and objective terms are defined.

Sets

- R, Set of all resources.
- P, Set of all products where $P = P^M \cup P^I \cup P^{\text{end}}$
- $P^M \subset P$, Set of all raw materials. This is a subset of the products set.
- $P^I \subset P$, Set of all intermediate products. This set is a subset of the products set.
- $P^{\text{end}} \subset P$, Set of all end products. This is a subset of the products set. Also, all demands are for products in this set.
- Q^p, Set of all composition of product p. A product can have several compositions. This means that a certain product can be produced in different ways. This set comprises all the compositions belong to a certain product.
- Q^r, Set of all compositions that use resource r.
- IM^q, Set of all products that are an input material for composition q.
- OQ^p, Set of all compositions for which product p is an input material.

- T, Set of all points in time. If all points in time are in scope, this will be denoted as $[T]$
- D, Set of all external demands.
- D^p, Set of all external demands for product p.

Parameters

- PR_d, The product that is demanded by demand d.
- CV_{pq}, The conversion rate from product p to composition q where p is an input material of q. For example, a car needs 4 wheels. If a car would be produced by composition q and it this composition would need 4 wheels for the car to be produced, the conversion rate would be 4.
- PT_{pq}, The processing time from product p to composition q where p is an input material for q.
- DD_d, The desired delivery time for demand d.
- CAP_r, The capacity of a resource r.
- TI_{pt}, The target inventory for product p at time t.
- DQ_d, The desired quantity of demand d.
- SI_p, The starting inventory of product p.
- UR_{iqr}, Boolean parameter which is equal to 1 if input material i uses resource r to create composition q.

Variables

- y_{pt}, Decision variable which is equal to the quantity demanded in the system at time t for product p.
- x_{qt}, Decision variable which is equal to the quantity supplied in the system at time t for a product using composition q.
- z_{pqt}, Variable which is equal to the quantity demanded at time t for product p that is used to make an output product that uses composition q.

- f^d_{pt}, Variable which is equal to the quantity supplied in the supply chain of demand d at time t. If product p is used in the supply chain of demand d, there need to be non-zero values of this variable somewhere in the time frame. This variable has its corresponding binary variable bf^d_{pt}.

- if^d_p, Variable that is equal to the quantity fulfilled by inventory supply of product p that is in the supply chain of demand d.

- g^d_{qt}, Variable that is equal to the quantity selected for composition q of a certain product belonging to the supply chain of demand d at time t. This variable has a corresponding binary variable bg^d_{qt}.

- h^d_{iqut}, Variable that connects two supplies in a supply chain. This variable is positive if an input material i becomes available at time u and is then used for composition q of the output product that becomes available at time t.

- I_{pt}, Variable which is equal to the Inventory level at time t for product p.

- MI_{pt}, Variable which is set by the inventory level and the target inventory such that the missing inventory is shown.

- OI_{pt}, Variable which is set by the inventory level and the target inventory such that the inventory is shown that is above the target inventory.

- v^d_t, Decision variable which is equal to the quantity delivered for external demand d at time t. This variable has its corresponding binary variable with bv^d_t

- mv_d, Variable that is equal to the missing demand for demand d.

- uc_{pot}, Variable that is equal to the unused capacity at time t for resource r.

- oc_{pot}, Variable that is equal to the overload capacity at time t for resource r.

Constraints

In this section all constraints are formulated and explained.

$$I_{p,t-1} + \sum_{q \in Q^p} x_{qt} - y_{pt} - \sum_{d \in D^p} v_t^d = I_{pt}, \quad p \in P, t = 1, ..., T \tag{2.1}$$

$$I_{p,0} = \mathrm{IS}_p, \quad p \in P \tag{2.2}$$

$$x_{q,t+PT_{iq}} = \mathrm{CV}_{iq} \cdot z_{iqt}, \quad p \in P \setminus P^M, q \in Q^p, i \in \mathrm{IM}^q, t = 0, ..., T - \mathrm{PT}_{iq} \tag{2.3}$$

$$\sum_{q \in \mathrm{OQ}^p} z_{pqt} = y_{pt}, \quad p \in P \setminus P^{\mathrm{end}}, t \in [T] \tag{2.4}$$

$$\mathrm{TI}_{pt} = I_{pt} + \mathrm{MI}_{pt} - \mathrm{OI}_{pt}, \quad p \in P, t \in [T] \tag{2.5}$$

Constraints (2.1) - (2.5) set the basic inventory levels depending on the supplies and demands during the time period. Variables x_{qt} and y_{pt} change depending on the need of supplies inside the system, v_t^d is non-zero on the delivery dates of external demands. Constraints (2.1) set the inventory level using the previous inventory level and the relevant supplies and demands at time t. The sum over Q^p is taken such that every composition of product p is taken into account. Constraints (2.2) Set the initial inventory level. Constraints (2.3) and (2.4) set the relation between the demands and supplies inside the system. Given a composition q of product p, we know which input products are needed to make p. We also know, how much of these products are needed and what the production time is for these. Therefore we know that for supply to be ready at time $t + PT_{iq}$ and this supply would need 3 units of input product i, 3 units of i would be needed at time t. Variable z_{pqt} at the start of the production is set such that enough input products are created. Next to this, a sum over all the possible compositions for which this product is an input material is taken such that enough internal demand is created for this product. Variable z_{pqt} was set to 3 in our previous example, but let us say that there is another composition that needs the same material to be produced at the same time. Constraints (2.4) make sure that enough internal demand is created. (2.5) set the variables for the inventory levels correctly which will later be used in the objective function.

$$\sum_{t=0}^{T} v_t^d + \mathrm{mv}_d = \mathrm{DQ}_d, \qquad d \in D \tag{2.6}$$

$$v_t^d \leq \mathrm{DQ}^d \cdot \mathrm{bv}_t^d, \qquad d \in D, t \in [T] \tag{2.7}$$

$$\sum_{t=0}^{T} \mathrm{bv}_t^d \leq 1, \qquad d \in D, t \in [T] \tag{2.8}$$

$$\mathrm{bv}_t^d = 0, \qquad d \in D, t = 0, ..., DD^d - 1 \tag{2.9}$$

Constraints (2.6) - (2.9) set the variables that have to do with the fulfillment of the external demand. Constraints (2.6) make sure that the fulfilled demand plus the missing demand equals the total quantity for a demand. Constraints (2.7) set the binary variable corresponding to the external demand to 1. Constraints (2.8) make sure that only one binary variable for a resource and demand can be set to 1. Constraints (2.9) make sure that the delivery can only be made on its delivery date or later.

$$v_t^d = \sum_{u=1}^{t} f^d_{\mathrm{PR}_d,u} + \mathrm{if}^d_{\mathrm{PR}_d,t}, \qquad d \in D, t \in [T] \tag{2.10}$$

$$f^d_{pt} = 0, \qquad d \in D, p \in P^{\mathrm{end}} \setminus PR_d, t \in [T] \tag{2.11}$$

$$\sum_{q \in Q^p} g^d_{qt} = f^d_{pt}, \qquad d \in D, p \in P, t \in [T] \tag{2.12}$$

$$\sum_{t=0}^{T} g^d_{qt} \leq \mathrm{CM} \cdot bg^d_q, \qquad d \in D, p \in P, q \in Q^p \tag{2.13}$$

$$\sum_{q \in Q^p} bg^d_q \leq 1, \qquad d \in D, p \in P \tag{2.14}$$

$$\mathrm{CV}_{iq} \cdot g^d_{qt} = \sum_{u=0}^{t-PT_{iq}} h^d_{iqut}, \qquad d \in D, p \in P \setminus P^M, q \in Q^p, i \in \mathrm{IM}^q, t = PT_{iq}, ..., T \tag{2.15}$$

$$\sum_{u=t-\mathrm{PT}_{iq}+1}^{t} h^d_{iqut} = 0, \qquad d \in D, p \in P \setminus P^M, q \in Q^p, i \in IM^q, t \in T \tag{2.16}$$

$$\sum_{q \in OQ^p} \sum_{u=t+\mathrm{PT}_{pq}}^{T} h^d_{pqtu} = f^d_{pt} + \mathrm{if}^d_{pt}, \qquad d \in D, p \in P \setminus P^{\mathrm{end}}, t \in [T] \tag{2.17}$$

$$\sum_{d \in D} g^d_{qt} \leq x_{qt}, \qquad p \in P, q \in Q^p, t \in [T] \tag{2.18}$$

$$\sum_{d \in D} \sum_{t \in [T]} \mathrm{if}^d_{pt} = \mathrm{tif}_p, \qquad p \in P \tag{2.19}$$

$$\mathrm{tif}_p \leq IS_p, \qquad p \in P \tag{2.20}$$

$$f^d_{pt} \leq \mathrm{CM} \cdot \mathrm{bf}^d_{pt}, \qquad d \in D, p \in P, t \in [T] \tag{2.21}$$

Constraints (2.10) - (2.21) handles the supply chain for particular external demands. Every external demand has a certain supply chain for it to be created and these constraints create a supply chain that corresponds to an external demand. (2.10) makes sure that every external demand has corresponding supply. Constraints (2.11) make sure that all supply for end products that do not correspond to the demand are set to 0 and no supply chain can be created for those. Constraints (2.12) make sure that the sum of all compositions of q equal the supply of p for a certain demand. Constraints (2.13) and (2.14) make sure that only one composition can be chosen to make a certain product. Constraints (2.15) uses the same principles as Constraints (2.3). This time it is for demands specifically. Next to this, for Constraints (2.3), we know that if a supply becomes available at time t, the start of production is at time $t - \mathrm{PT}_{iq}$ which means that variable z_{iqt} has to be positive at that time such that there will be demand at the correct time. In this case, however, we link the times for which supply becomes available between supplies in the supply chain. This means that the supply upstream for this supply can be made available in between 0 and $t - \mathrm{PT}_{iq}$. Variable h^d_{iqut} makes sure that these upstream and downstream supplies are linked correctly with each other. These constraints therefore deal with the output such that enough supply is generated for a supply downstream. Constraints (2.16) make sure that a variable h^d_{iqut} can only be positive if the difference between time u and time t is bigger than the production time to create composition q from product p. Constraints (2.17) cover the other end. They make sure enough input is generated for h^d_{iqut}. Note that this can either be a supply inside the system or an inventory supply. A supply inside the system needs supplies of its own but the inventory supplies do not. Therefore if an inventory supply is chosen, it would not need supplies of its own since the supply is taken from the inventory. Constraints (2.18) link the supply for a supply chain of a specific external demand d with all the supply that is supplied at a certain time. Constraints (2.19) set the variable for the total inventory supply used during the time period. Constraints (2.20) make sure that the total inventory supply do not exceed the inventory supply. Constraints (2.21) set the boolean variables for the supplies of the supply chain of demand d.

$$\sum_{q \in Q^r} \sum_{i \in IM^q} \sum_{u=max\{t-\mathrm{PT}_{iq},0\}}^{t-1} \mathrm{UR}_{iqr} \cdot z_{iqu} + \mathrm{uc}_{rt} - \mathrm{oc}_{rt} = \mathrm{CAP}_r, \qquad r \in R, t = 1, ..., T \tag{2.22}$$

Constraints (2.22) set the capacities for resources. We assume that the start and end times of products using resources are discrete. This means that at a certain time t a production starts or ends for a product. To calculate the capacity of a machine, we look at the number of products that are still in production in between t and $t-1$. We can conclude from this that all products that start between $t-1$ and $t - \mathrm{PT}_{iq}$ are still in production. Boolean parameter UR_{iqr} makes sure that only production flow is taken into account for flow that uses the machine. The last two variables set the equality.

$$y_{pt}, I_{pt}, \mathrm{MI}_{pt}, \mathrm{OI}_{pt}, \geq 0, \qquad p \in P, t \in [T] \tag{2.23}$$

$$x_{qt}, z_{pqt} \geq 0, \qquad p \in P, q \in Q^p, t \in [T] \tag{2.24}$$

$$v^d_t, \mathrm{mv}_d \geq 0, \quad \mathrm{bv}^d_t, \mathrm{bv}^d_{t+}, \mathrm{bv}^d_{t-} \in \{0,1\}, \qquad d \in D, t \in [T] \tag{2.25}$$

$$f^d_{pt}, \mathrm{if}^d_{pt}, g^d_{qt} \geq 0, \quad \mathrm{bf}^d_{pt}, \in \{0,1\}, \qquad d \in D, p \in P, q \in Q^p, t \in [T] \tag{2.26}$$

$$h^d_{pqut} \geq 0 \qquad d \in D, p \in P^O, q \in OQ^p, t \in [T], u = 0, ..., t-1 \tag{2.27}$$

$$\mathrm{uc}_{mt}, \mathrm{oc}_{mt} \geq 0, \qquad m \in M, t \in [T] \tag{2.28}$$

Constraints (2.23) - (2.28) set the variable bounds.

Objective

$$\min \sum_{p \in P} \sum_{t=0}^{T} \Big(C^{MI}_p \cdot (\mathrm{MI}_{pt})^{\gamma_{MI}} + C^{OI}_p \cdot (\mathrm{OI}_{pt})^{\gamma_{OI}} \Big) + \tag{2.29}$$

$$\sum_{d \in D} \Big(C^{MD}_d \cdot (\mathrm{mv}_d)^{\gamma_{MD}} + \sum_{t=0}^{T} C^{LD}_d \cdot (t - \mathrm{DD}_d)^{\gamma_{LD}} \cdot \mathrm{bv}^d_t \Big) + \tag{2.30}$$

$$\sum_{t=0}^{T} \sum_{r \in R} \Big(C^{uc}_r \cdot \mathrm{uc}_{rt} + C^{oc}_r \cdot \mathrm{oc}_{rt} \Big) + \tag{2.31}$$

$$\sum_{t=0}^{T} \sum_{d \in D} \sum_{p \in P \setminus P^{\mathrm{end}}} \sum_{q \in Q^p} \sum_{i \in IM^q} \sum_{u=0}^{t-\mathrm{PT}_{iq}} \Big(C^{SH}_i (t - \mathrm{PT}_{iq} - u) \cdot h^d_{iqut} \Big) + \tag{2.32}$$

$$\sum_{t=0}^{T} \sum_{d \in D} \sum_{p \in P} \Big(C^{SH}_p \cdot t \cdot \mathrm{if}^d_{pt} \Big) + \tag{2.33}$$

$$\sum_{t=0}^{T} \sum_{d \in D} \sum_{u=0}^{t} \Big(C^{SH}_{PR_d} \cdot (t - u) \cdot \mathrm{bv}^d_t \cdot \mathrm{f}^d_{\mathrm{PR}_d, u} \Big) + \tag{2.34}$$

$$\sum_{p \in P} \Big(C^{UIS}_p \cdot (\mathrm{IS}_p - \mathrm{tif}_p) \Big) + \tag{2.35}$$

$$\sum_{p \in P} \sum_{t=0}^{T} \Big(C^{SH}_p \cdot (T - t) \big(\sum_{q \in Q^p} x_{qt} - \sum_{d \in D} f^d_{pt} \big) \Big) + \tag{2.36}$$

$$\sum_{t=0}^{T} \sum_{p \in P} \sum_{q \in Q^p} \Big(C^{\mathrm{CP}}_q \cdot x_{qt} \Big) \tag{2.37}$$

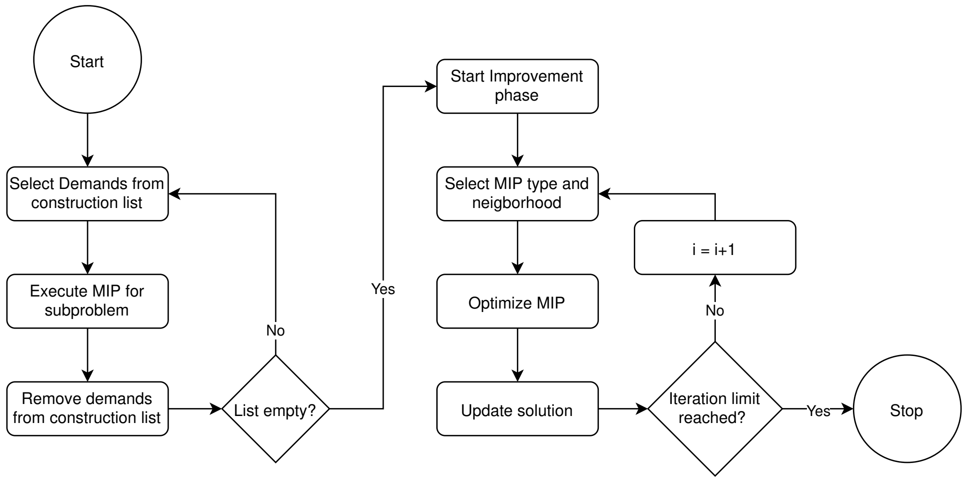
The objective function consists of several different parts. Each part addresses a specific cost component of the problem. All variables have a certain cost parameter C and several variables have an exponent γ. Term (2.29) makes sure the costs of missing inventories and overload inventories are taken into account. Term (2.30) calculates the costs for missing demand and it also calculates the cost for lateness of a delivery. Regarding this lateness, it is more desirable to have more demands a little late than to have a few demands very late. Therefore, there is an exponent in this term. Term (2.31) calculates the capacity costs for machines. If there is overload capacity or unused capacity, at a given time, the costs will be positive. Terms (2.32) - (2.34) include costs regarding supply holding. We want to minimize the processing time for a given demand. Supply holding time is the time for which a product is in stock and not being processed. Therefore, we want to know the time difference between two products that are in the supply chain of demand d minus the processing time between those products. Term (2.32) accounts for the supply holding costs regarding two internal supplies. (2.33) accounts for the supplies fulfilled by inventory supply. For this we set the time of the inventory supply to the beginning of the time frame since it is already available at the beginning of the period. Term (2.34) accounts for the time in between a supply for an end product becoming available for an external demand and the delivery of this external demand. Term (2.35) gives penalties to inventory supply that is not used in the entire period. Term (2.36) penalizes supply that is not being used for demand. This supply can be useful to adhere to the inventory level. We penalize supply that is not used with supply holding costs until the end of the planning horizon. Term (2.37) incurs certain costs for using a certain composition.

2.2 Neighborhood search heuristic

Before going into detail on how neighborhoods are selected and the MILPs are formulated, let us give a brief overview on how the general matheuristic works. Figure 2.1 gives a visual overview of how the process works. There is a construction phase and an improvement phase. The main goal for the construction phase is to find a relatively good starting point. We sort all external demands for their desired delivery dates, with the delivery dates closest to the beginning of the time period first. We then plan all external demands in several iterations. Each iteration selects a number of external demands from the sorted list and an MILP optimizes the plan for these demands. The selected external demands are then removed from the list and the optimal values of these variables will become fixed. We continue with the next iteration where we optimize new demands given the plan that we have acquired from the previous iterations. We continue this procedure until the list is empty. At this point, every demand has been in scope and most demands will be scheduled. How the mathematical program that is used in this procedure is formulated, will be discussed in Section 2.3.

Once the construction phase is completed, we continue with the improvement phase of the

Figure 2.1: Flow chart for a the neighborhood search algorithm


Start
Select Demands from construction list
Execute MIP for subproblem
Remove demands from construction list
List empty?
No
Yes
Start Improvement phase
Select MIP type and neigborhood
Optimize MIP
Update solution
i = i+1
No
Iteration limit reached?
Yes
Stop

heuristic. In this phase we try to improve the current plan. There are two main decisions to be made, what neighborhood structure and which MILP. When those two decisions are made the actual neighborhood and the neighborhood size can be determined.

We have two different neighborhood structures. Given that we are in the improvement phase of the optimization process, a solution exists in which all demands have most probably been fulfilled up to a certain point and they have their corresponding upstream supplies. For the first neighborhood, we select a number of external demands. For such demands, we select the upstream variables that are related to the external demands and free them. All the supplies and demands related to external demands that are not selected are now fixed. We solve an MILP for this neighborhood and look whether the solution of this MILP will improve the entire plan. Which variables are precisely made active is discussed in 2.3.

The second neighborhood focuses more on resources and a specific time period. We select several resources and free all variables connected to those machines for a certain time period. The time period can be chosen randomly or can be based on how heavily loaded a resource is. A cost improvement might be found if a resource is either underused or overused. Therefore, a selection with the usage of a certain resource and corresponding time is useful to take into account. The idea behind this is that redistributing the resources in a certain way might yield improvements. As the goal of this thesis is to see what the influence of multiple MILP formulations is on a neighborhood search algorithm, no other neighborhood types are formulated. During the experiments, the neighborhood types are chosen randomly.

2.3 MILP formulations

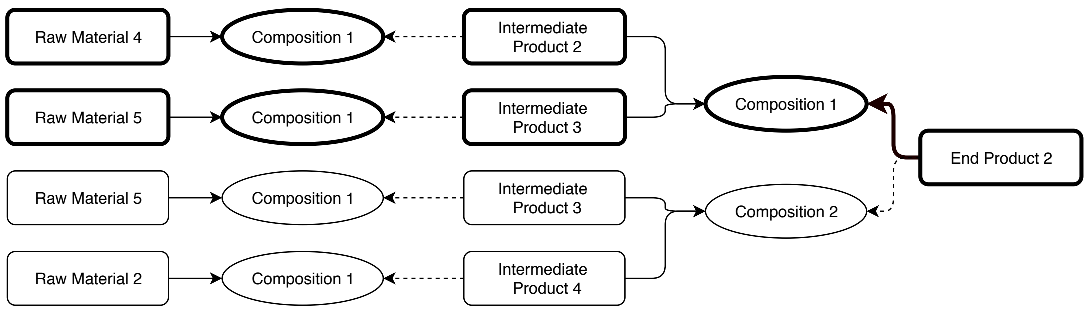
In this section we describe how the different MILPs will be formulated. These are Formulation 1 and Formulation 2. Formulation 1 is a more simple Formulation that can be solved quicker or with a larger neighborhood. Formulation 2 is more complex and less easy to solve but captures the MINLP better. Therefore, we continue to refer to Formulation 1 and 2 as the small and large MILPs respectively.

We will first discuss several changes from the MINLP that are done in both Formulations. When solving a subproblem for a particular neighborhood, we limit the number of variables that are created such that the MILP will generate an acceptable solution more quickly. Only a certain number of variables can be activated such that the problem stays solvable within reasonable time. Therefore, if variables are activated in a more clever way, smarter neighborhoods are created that can find better solutions. In the supply chain of external demand d, instead of freeing all variables with all possible times in the planning horizon, we only free the variables that have more logical times. In general, it should be preferable for supplies that follow each other to be relatively close together. Therefore, creating a variable for an upstream product that has a very high duration until the next step of the supply chain begins would often not be optimal. Because of this, a decision variable only has a set of times that are active in an iteration. This is done for all variables related to external demands. These are f^d_{pt}, if^d_{pt}, g^d_{qt} and h^d_{pqut}. To decide which variables to activate for an iteration we look to the current plan in the improvement phase and several parameters in the construction phase. In the improvement phase the times are chosen depending on the times for which the variables are currently non-zero. On a timeline of a given product of a certain demand, there can be multiple variables that are non-zero. Therefore, we look at the earliest and the latest times and set the active time frame. For the variables g^d_{qt}, the period for which variables will be activated is described as follows.

$$t \in [\tau^{\min}_{dq} - \epsilon, \tau^{\max}_{dq} + \epsilon] \tag{2.38}$$

In definition (2.38), ϵ is a constant that can be chosen to adjust for the length of the time frame. τ_{dq} refers to a time where a certain variable g^d_{qt} is non-zero in the current plan. The min and max refer to the earliest and latest times respectively. For example, a variable g^d_{qt} has a value of 5 at $t = 5$, 3 at $t = 7$, 4 at $t = 9$ and 0 on all other times. If this variable is selected, $\tau^{\min}_{dq}$ would be 5 and $\tau^{\max}_{dq}$ would be 9. Variables if^d_{pt} are activated similarly as g^d_{qt}. Variables f^d_{pt} are activated if a corresponding g^d_{qt} or if^d_{pt} is active. For variables g^d_{qt}, this means that if a variable g^d_{qt} is active the variable f^d_{pt} is active for the same demand, time and product p for which composition q is a composition. Variables h^d_{pqut} are active if its corresponding variables f^d_{pu} and g^d_{qt} are both active.

Figure 2.2: Choosing a composition



Recall Figure 1.1 where the flow is explained in a supply chain. In Figure 2.2, we focus on the supply chain of the second end product. The decision is made to choose the first composition. With the current definition, only the variables that are in the chosen composition will be freed as these are the only variables that are active in the current plan. All variables that follow from the second composition would still be locked. Because we want to give the optimizer the option to switch to compositions, variables have to be activated here as well. This is done such that the variables are active at the same time as their alternative in the current composition. If this would not be possible because the production time is longer for products in this composition, the active time frame is moved further back.

Since there is no existing plan in the construction phase, the variables that are active are selected based on desired demand dates and the production times for products in the supply chain. For the construction phase, the active set of times is described similarly to the improvement phase, with the the difference that there is only one τ_{dq}. Therefore, $\tau_{dq} = \tau^{\min}_{dq} = \tau^{\max}_{dq}$. In the construction phase, we choose τ_{dq} according to the desired delivery date of d and the production times upstream such that that demand can be fulfilled. The starting point is the desired delivery date for the end product of a given demand d. Therefore, $\tau_{dq} = \text{DD}_d \quad \forall q \in Q^{\text{PR}_d}$. Once we know this we can traverse through the supply chain and set $\tau_{dj} = \tau_{dq} - \text{PT}_{iq} \quad \forall j \in Q^i$. The activation of the other variables is done in the same way as in the improvement phase.

For the small MILP, we want to simplify variable h^d_{iqut}. There are many combinations for this variable which means that a neighborhood would become too large very quickly. Constraints (2.15) - (2.17) have three purposes. Firstly, they balance upstream and downstream supply such that upstream supply needs to be created to get a valid supply chain. Next to this, they make sure that downstream supply can only be made available after a certain time. Lastly, they link specific times for downstream and upstream supply such that it is known which upstream supply is used for which downstream supply. This is needed to calculate the supply holding cost term (2.32).

For the small MILP, we replace the constraints (2.15) - (2.17). This is done in the following way. Instead of h^d_{iqut} we introduce variables η^d_{iqt}. Variables η^d_{iqt} can be described as connecting product i to composition q in the supply chain of demand d where t represents the time of which product i becomes available. To make sure that balancing constraints for the supply chain will stay valid, we introduce the following constraints:

$$\sum_{u=0}^{t} \mathrm{CV}_{iq} \cdot g^d_{qu} \leq \sum_{u=0}^{\max\{t-\mathrm{PT}_{iq},0\}} \eta^d_{iqu}, \qquad d \in D, p \in P \setminus P^m, q \in Q^p, i \in IM^q t \in [T] \tag{2.39}$$

$$\sum_{q \in OQ^p} \eta^d_{pqt} = f^d_{pt} + \mathrm{if}^d_{pt}, \qquad d \in D, p \in P \setminus P^{\mathrm{end}}, t \in [T] \tag{2.40}$$

Constraints (2.39) make sure that all the supply that is needed up to time t is accounted for. Up to time t, the supply that is generated for composition q cannot be more than the supply that is reserved for the input products of composition q, corrected for the conversion CV_{iq}. Constraints (2.40) make sure that enough products for product p is reserved for the supply chain of demand d. Because of the simplifications, we are unable to identify what the exact supply holding duration for products is since we do not directly link the times of two supplies in a supply chain anymore. Therefore, we would need to approximate this. This can be done by using the current plan. Given a fulfillment for demand d from product p to composition q, we take the average of the times that are non-zero in the current plan for composition q. This average can be determined like this:

$$\lambda^d_q = \left\lceil \frac{\sum_{t=0}^{T} t \cdot g^d_{qt}}{\sum_{t=0}^{T} g^d_{qt}} \right\rceil, \qquad d \in D, p \in P, q \in Q^p \tag{2.41}$$

The supply holding term of (2.32) would then become the following:

$$\sum_{d \in D} \sum_{p \in P \setminus P^{\mathrm{end}}} \sum_{q \in OQ^p} \sum_{t=0}^{\lambda^d_q} \left(C^{SH}_p (\lambda^d_q - \mathrm{PT}_{pq} - t) \cdot \eta^d_{pqt} \right) \tag{2.42}$$

Term (2.42) uses the average times of the previous plans λ^d_q. This causes the supply holding term to not be entirely accurate anymore.

In the objective of the MINLP, there are several non-linear terms. For subproblems, these terms need to be linearized. The exponential terms will be dealt with in the following way. We replace the non-linear variables with stepwise linear variables. This is done as follows. We cut the variable with an exponent up into J linear variables. Each of these variables has an upper and a lower bound. For a given variable x, let χ_j be one of the stepwise linear equivalent. We then formulate the upper bound for this variable to be δ_j. The lower bound of the variable will be the upper bound of the previous variable, δ_{j-1} with $\delta_0 = 0$. For each non-linear variable, we add the following constraints:

$$x = \sum_{j=1}^{J} \chi_j \tag{2.43}$$

$$\chi_j \leq \delta_j - \delta_{j-1} \quad \forall j = 1, ..., J-1 \tag{2.44}$$

Variable χ_J has no upper bound since it is the last segment. Regarding the objective function we add a constant μ_j for all stepwise linear variables. This constant will be multiplied with the current constant in the objective. This constant can be calculated in the following way:

$$\mu_j = \frac{\delta_j^{\gamma} - \delta_{j-1}^{\gamma}}{\delta_j - \delta_{j-1}}, \qquad j = 0, ..., J-1 \tag{2.45}$$

$$\mu_J = 2\mu_{j-1} \tag{2.46}$$

Since the last segment does not have an upper bound, we set the cost for that segment to twice the cost of the previous segment. The difference between the two MILP formulations is that in the more simple formulation, the number of segments will be smaller. To be precise, the number of segments used in the small MILP is 5 and the number of segments used in the large MILP is 30.

Term (2.34) multiplies two variables with each other. To linearize this, we formulate a parameter DT_d which is the delivery time for demand d in the current plan. In the construction phase we will set $\text{DT}_d = \text{DD}_\text{d}$. The new term would become the following:

$$\sum_{d \in D} \sum_{t=0}^{\text{DT}_d} \left(C^{SH}_{PR_d} \cdot (\text{DT}_d - t) \cdot f^{d}_{\text{PR}_d, t} \right) \tag{2.47}$$

Term (2.47) is linear. However, if a decision is made such that the delivery time of a demand changes, this term would not correctly portray the costs anymore. This is used in the small MILP. The accuracy problem can be solved by creating an extra variable which links the final fulfillment with the delivery variable of the demand. This is done in the large MILP. This can be formulated in the following way.

$$\sum_{u=0}^{t} w_{ut}^{d} \leq v_{t}^{d}, \qquad d \in D, t = \mathrm{DD}^{d}, ..., T \tag{2.48}$$

$$\sum_{u=\max\{\mathrm{DD}^{d},t\}}^{T} w_{tu}^{d} = f_{\mathrm{PT}_{d},t}^{d}, \qquad d \in D, t = 0, ..., T \tag{2.49}$$

$$w_{ut}^{d} \geq 0, \qquad d \in D, t = \mathrm{DD}_{d}, ..., T, u = 0, ..., t \tag{2.50}$$

The resulting term in the objective function would become:

$$\sum_{d \in D} \sum_{t=\mathrm{DD}^{d}}^{T} \sum_{u=0}^{t} \left(C_{\mathrm{PR}_{d}}^{SH} \cdot (t-u) \cdot w_{ut}^{d} \right) \tag{2.51}$$

The extra constraints (2.48) - (2.50) set variable w_{ut}^{d} such that it links the times from which the supply for demand d is ready to be transported until the time it is actually delivered. Subscript u indicates the time it is ready and subscript t indicates the time it is delivered. Constraints (2.48) make sure that w_{ut}^{d} can only be positive if its corresponding delivery variable v_{t}^{d} is positive at time t. Constraints (2.49) make sure that variable w_{tu}^{d} is set positive to the time the end product is finished. The addition to the term (2.51) calculates the corresponding costs for the supply holding. Because of these constraints there is a linear term in the objective function that represents the costs of supply holding accurately. However, this has the result that more variables need to be constructed and the resulting formulation will be more difficult to solve. Therefore, this is only used in the large MILP.

Summarizing, the small MILP has the following differences with the large MILP. the small MILP makes use of variables η_{iqt}^{d} for the linking two supplies in a supply chain whereas the large MILP uses variables h_{iqut}^{d}. Next to this, for variables with an exponential term, the small MILP has less segments and less stepwise linear variables the large MILP makes use of variable w_{ut}^{d} to accurately portray the supply holding costs from the date the product is ready to be transported to the customer until the product is actually transported to the customer. In the small MILP this is approximated.

2.4 Algorithms

Now that all the building blocks of the heuristics have been explained, we can discuss all the different algorithms that will be experimented on. The difference between all these algorithms is the procedure of selecting an MILP. First of all, the heuristic is executed with either only the large MILP or the small MILP to get benchmarks on how the heuristic performs when only one MILP is used. Three different types of algorithms will be examined that use both MILPs. These algorithms are called switch, stuck and re-execute.

The first of these, switch, is the simplest of the three algorithms. This algorithm uses the assumption that the large MILP will be more effective in a later stage of the improvement process and the small MILP will be more effective in the beginning. After a certain point in the optimization process the algorithm will switch from using the small MILP to using the large MILP. For the rest of the process, the large MILP will be used.

The second algorithm is the stuck algorithm. This algorithm uses the assumption that the large MILP has a higher chance of getting out of local optima. The small MILP will be used when the optimizer is able to find good improvements. If the optimizer was not able to find substantial improvements in the previous couple of iterations, the large MILP will be used for a couple of iterations until a substantial improvement is found. Similarly to the switch algorithm, this algorithm will use the small MILP more often in the beginning of the optimization process and the large MILP in the latter stages. However, this algorithm is a bit more sophisticated in choosing the right moments for using the large MILP.

The last algorithm is called the re-execute algorithm. This algorithm uses the assumption that when there is a substantial mismatch between the result of the small MILP and the score of the actual plan, a potential improvement is missed. In this algorithm, the small MILP is used as a base. After each iteration, the mismatch of that iteration is measured. If the mismatch is substantial, the iteration will be re-executed using the large MILP. This means that exactly the same neighborhood is used for the next iteration. Because of the large mismatch, there is an unfulfilled potential of the iteration that the large MILP might be able to find. However, the downside of this algorithm is that effectively, a number of iterations will take considerably longer to execute as a re-execute will make a single iteration at least twice as long. However, in the latter stages of the optimization process, there might be quite a number of iterations for which the neighborhood is not suitable to find a substantial improvement. In these circumstances, finding a substantial improvement in a certain iteration becomes more important than executing more iterations. Therefore, it becomes more important to use all the potential of the neighborhoods in these iterations. With this algorithm, this potential will be used whilst limiting the total execution time by using the small MILP as a base.

In the next chapter, a more in depth analysis will be made regarding the differences between the small and the large MILP and the influence of mismatches on iterations.

Chapter 3

Results

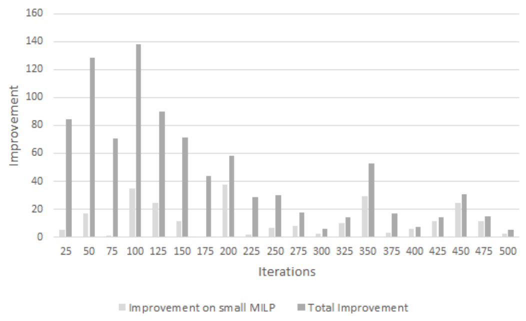
In this chapter, the results of the experiments are shown and interpreted. First of all, we look into the different MILPs on the iterative level. This means that we are able to see the differences between the MILPs purely. After this, we analyze the experimentation of the algorithms. All results from the heuristics mentioned in Section 2.4 will be shown and interpreted.

3.1 MILP differences

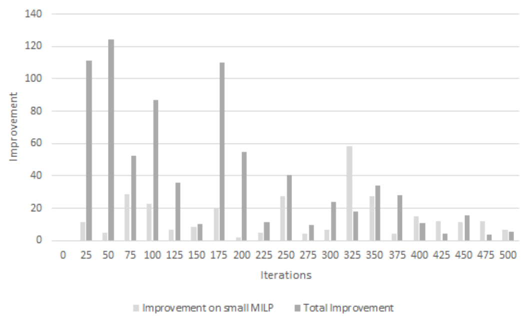
In this section we examine the differences between the small and the large MILP. Since the large MILP incorporates more sophisticated aspects of the problem it should yield better results on the iterative level. This has been tested on datasets 1 and 2. For 500 iterations in the improvement phase the small and large MILPs are compared with each other. Given a certain neighborhood both the small and large MILP are executed. After translating the resulting plans back to the main plan, the scores that result from that plan are compared with each other. This has been executed with the small MILP as base and with the large MILP as a base. Using an MILP as a base means that we always continue the optimization process with that MILP. For example, if we use the small MILP as a base and the large MILP yields a better result in a given iteration, we still use the plan that resulted from the small MILP as the base plan for the next iteration. We are not interested in the performance of the entire optimization process in this experiment but only in the iterative performance of the two MILPs using the same neighborhoods. Next to this, using a single MILP as a base can give information on how an MILP behaves in a plan that has been constructed using the other MILP. Since the two MILPs can follow different paths in creating plans, their behaviour might differ when a plan that has been constructed for which the other MILP is used.

Figures 3.1a and 3.1b show the performance of the large MILP versus the small MILP in comparison with the overall improvements found in those iterations for dataset 1. Figure 3.1a shows the results for when the large MILP is used as a base and Figure 3.1b shows the results for when the small MILP is used as a base. For the run with the large MILP as a base, The start and end scores were 12959.85 and 12036.24 respectively. For the run with the small MILP as a base, The start and end scores were 13023.09 and 12231.62 respectively.

Figure 3.1: Differences between the improvements found of the small and large MILP compared to the total improvements found for dataset 1. The results are shown per 25 iterations.



(a) Large MILP as base.

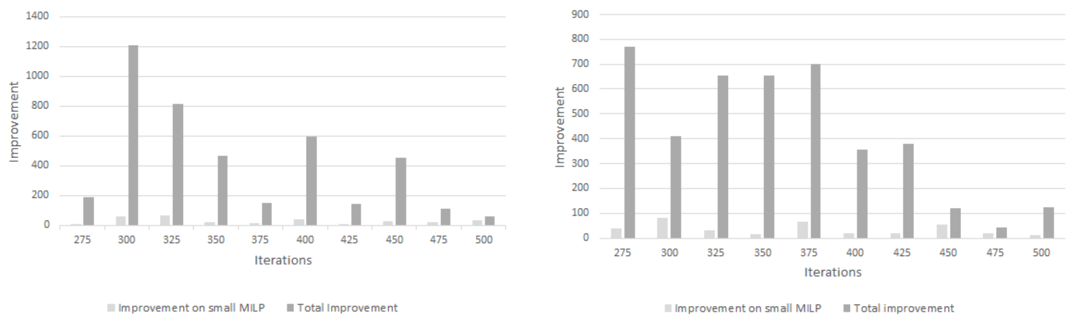


(b) Small MILP as base.

Regarding the optimization process, the figures show behaviour for which the total improvement decreases over time. This seems reasonable since improvements will be more difficult to find in the later stages of the optimization process. When looking at the performance of the large MILP in comparison to the small MILP, there does not appear to be that much of a decrease over time. Regarding Figure 3.1a, in the later stages of the optimization process, the improvement found by the large MILP that has not been found by the small MILP becomes closer to the total improvement found. This could have two explanations. The first explanation is that the large MILP is able to find solutions that the small MILP is not able to find because of the extra depth that is brought to the large MILP. It could also be the case that the small MILP has difficulty finding improvements in a plan that is built from solutions of the large MILP. Regarding Figure 3.1b, the improvement of the large MILP on the small MILP can be interpreted as the improvement that is found on top of the total improvement found. Therefore, it is possible that the improvement of the large MILP that is not found by the small MILP is larger than the total improvement found as the small MILP is used as a base. Just as in Figure 3.1a, we can see that the improvement found by the large MILP that is not found by the small MILP shows less of a decrease over time than the total improvement found. This would illustrate that even though it is more difficult to find solutions at this stage, the large MILP is able to find better plans. When looking at iterations 301-325, we see that the improvement found by the Large MILP is more than 3 times as large as the total improvement found. This could suggest that the small MILP has made several different mistakes. If a certain plan needs a lot of changes for it to improve upon, one can imagine that more mismatches are likely to occur than when only little changes need to be made in respect to the previous plan. Therefore, in this case, the small MILP might find it significantly more difficult to find a good improvement than the large MILP. This would suggest that using the large MILP in such circumstances would benefit the total optimization process.

Figures 3.2a and 3.2b show the results for dataset 2. Only the last 250 iterations are illustrated since the first iterations show a too big of a total improvement that it is hard to clearly interpret the rest of the process. For the run with the large MILP as a base, The start and end scores were 21522.09 and 17337.47 respectively. For the run with the small MILP as a base, The start and end scores were 23001.62 and 18788.18 respectively. We can see that the impact of the large MILP is smaller in comparison to the total improvement than in dataset 1. However, just as with dataset 1, the improvement found by the large MILP that the small MILP is not able to find does not seem to be very time dependent.

Figure 3.2: Differences between the improvements found of the small and large MILP compared to the total improvements found for dataset 2. The results are shown per 25 iterations.



(a) large MILP as base.

(b) small MILP as base.

To have a more clear interpretation on what the influence of the large MILP during the optimization process is, we can have a look at Table 3.1. For all of the datasets and bases, the influence of the large MILP becomes larger over time. This would indicate that using the large MILP at a later stage of the optimization process would increase the performance more substantially.

Table 3.1: Influence of large MILP on total improvement. Improvement of large MILP divided by the total improvement per 100 iterations. The base column indicates which base is used.

	Base	1-100	101-200	201-300	301-400	401-500
Dataset 1	Large	0,11	0,27	0,31	0,55	0,69
	Small	0,24	0,30	0,46	1,40	1,97
Dataset 2	Large	0,00	0,02	0,03	0,07	0,21
	Small	0,00	0,01	0,07	0,06	0,25

Table 3.2: Category definition per dataset.

	Cat. 1	Cat. 2	Cat. 3
Dataset 1	<0.1 %	[0.1 %, 0.2 %]	>0.2 %
Dataset 2	<0.3 %	[0.3 %, 0.5 %]	>0.5 %

Next to these experiments, various experiments were performed to measure the size of mismatches and the subsequent effect on rollbacks. This might give an indication under which circumstances using the large MILP would be useful. In this case, the absolute mismatch is measured. The absolute mismatch is the value of the new plan minus the value of the plan the small MILP calculated. This means that even though the small MILP found an improvement, there could still be a substantial mismatch hidden. 500 iterations were performed on dataset 1 and dataset 2 using the small MILP and the large MILP. Each iteration is categorized based on the mismatch that occurred in this iteration using the small MILP. There are three categories which are defined based on the different datasets. Table 3.2 shows the category definition for the datasets.

To give some context on why these percentages were selected, let us look at the performance of the small MILP. We will discuss this in more detail in Section 3.2. For dataset 1, the average improvement in an iteration of the small MILP is around 0.07 % of the total score after the construction phase is completed. The three classifications were made such that large, medium and small mismatches could be distinguished for dataset 1. A similar logic was used for the creation of the categories for dataset 2.

Table 3.3 shows total improvement that the large MILP on the small MILP per category. We can see that for both datasets the majority of the iterations have a small mismatch. The most significant improvements are made in iterations that are listed in category 3. This suggests that the majority of iterations would not benefit from using the large MILP. However, if the large MILP can be used at the right times, the improvement phase might work more efficiently.

Table 3.3: The total improvement of the large MILP on the small MILP per category. Occ. is the number of occurences that a mismatch belonged to the category.

	Cat. 1	Occ.	Cat. 2	Occ.	Cat. 3	Occ.
Dataset 1	58,86	347	32,85	114	142,71	41
Dataset 2	8,01	380	21,99	22	345,51	98

3.2 Algorithm results

In this section the results of the various algorithms are reported and interpreted. The results are obtained using software of Dassault Systèmes Delmia Quintiq. The results are examined per dataset. All algorthims are repeated 10 times and the average results are reported. Next to this we distinguish a short and a long run. Each algorithm type is repeated 10 times for a short run and 10 times for a long run. The duration of the algorithms differs per dataset as well. Benchmark times were used which were most fit for a certain dataset. For each dataset, we report the average number of iterations. We also report the average score and the corresponding standard deviation. Finally, we report the percentage of iterations that result in a plan that is worse and consequently causes the heuristic to roll back to the previous plan.

Regarding the algorithms that use both the small and large MILP, the following settings were used for the experiments. For the stuck algorithm, we look at the previous 10 iterations and if no improvement has been found of more than 0.5 % of the total score in these iterations, we switch to the large MILP. When the last couple of iteration has found such an improvement, we switch back to the small MILP for the next iteration. Regarding the switch algorithm, the last quarter of the optimization process uses the large MILP. When three quarters of the optimization process is completed, we switch to the large MILP and use that for the rest of the duration. For the re-execute algorithm, a re-execution takes place if an absolute mismatch is measured that is bigger than 15 points. For datasets 1 and 2, this size seemed to include only the iterations where a substantial improvement could be found as mentioned in Section 3.1. For datasets 3 and 4, this research has not been done and we therefore use the same threshold for these runs as well.

Dataset 1

We start with examining the results of dataset 1. Table 3.4 shows the results of the short runs for this dataset. For this dataset, a short run is a run of at most 20 minutes or 290 iterations. We can see that the algorithm using the small MILP performs best for this dataset and the algorithm using the large MILP performs worst. For this dataset, the extra number of iterations performed is more important than using all the potential of individual iterations. The small MILP performs best regarding all the algorithms. Closely behind the small algorithm is the re-execute algorithm. What is interesting to notice is that the re-execute algorithm is slightly worse but seems to be more reliable as the sample standard deviation is lower by 54,4. Since, in practice, it is often more desirable to have an algorithm that is able to beat a certain benchmark in every run instead of having an algorithm that is sometimes able to find a very good solution, it could be argued that the performance of the re-execute algorithm is better.

Table 3.4: Results for the short runs of dataset 1. Rollbacks is the percentage of iterations which resulted in a worse plan

Algorithm	Duration	Iterations	Mean score	st. dev	Rollback %
Small	18:24	290	11920	205,48	17
Large	20:00	262	12008	149,34	10
Stuck	19:18	287	11989	155,14	14
Switch	18:36	290	11987	175,18	15
Re-execute	19:54	278	11928	151,08	15

Table 3.5 shows the results of the long runs for dataset 1. For this dataset, a long run is at most 35 minutes or 390 iterations. These results give some insight of when all algorithms are able to do the same number of iterations as every algorithm performed the 390 iterations. A few interesting observations can be made from this. First of all, although the large algorithm is able to find a very good average score, its average duration is significantly longer than the other algorithms. This is expected as the large MILP is slower and incorporates more details of the problem. What is less expected is that the re-execute algorithm is able to find a better average score than the large MILP. It also needs significantly less time than the large MILP. This suggests that only using the large MILP in situations where it is most needed already gives good results. Also something notable is the bad performance of the stuck algorithm. It has a worse average score as the small algorithm and it has a longer duration. An explanation for this could be that the algorithm is switching between MILP types quite often. Using the large MILP in a couple of iterations could result in a plan that is more suited to the large MILP. When we start using the small MILP again, this could result in more and worse mismatches for the upcoming iterations. The switch algorithm does not have this problem as when it starts using the large MILP, it does not switch back to the small MILP again. The re-execute algorithm could also have this problem since after a re-execution we continue with the small MILP. However, as the large MILP is used more sparsely and on specific occasions, the benefits of using it seem to be stronger than the hindrances. The standard deviation of the re-execute algorithm is lower than all the other algorithms. All in all, it seems that the re-execute algorithm works best for this dataset and using a combination of the small and large dataset could indeed improve performance.

Table 3.5: Results for the long runs of dataset 1.

Algorithm	Duration	Iterations	Mean score	St. dev.	Rollback %
Small	00:28:06	390	11786	118,73	24
Large	00:32:30	390	11679	122,16	18
Stuck	00:29:48	390	11803	150,10	20
Switch	0:28:42	390	11719	101,87	21
Re-execute	0:28:24	390	11669	84,05	21

Dataset 2

The results for this dataset is shown in Tables 3.6 and 3.7. For this dataset, each short run takes 30 minutes and each long run takes 45 minutes. Recall that for this dataset, the inventory adherence is not taken into account. Effectively, this means that the supply holding elements of the problem is the difference between the small and large MILP. Let us first look at the results of the short runs. Of all algorithms, we can see that the small algorithms performs best. It seems that the large MILP is not very suitable when the runs only last 30 minutes. We do see that the rollback percentage is significantly lower in the large algorithm but the fact that on average 73 iterations less can be performed has the effect that the average score is lower. What is noticeable is that the stuck algorithm has performed more iterations as the small algorithm. This is not logical as the stuck algorithm makes use of the slower large MILP on certain occasions. A possible explanation for this is that because of the large number of switches the plan becomes too complex. A new neighborhood would then have less room for improvements and would quickly find the same solution as the current plan. Lastly, the re-execute algorithm does not seem to work well in these settings. This suggests that the re-execute moments do not yield many improvements and performing more unique iterations is more beneficial to get to a good score.

Table 3.6: Results for the short runs of dataset 2.

Algorithm	Iterations	Mean score	St. dev.	Rollback %
Small	364	12229	423,88	23
Large	291	12556	447,44	12
Stuck	415	12375	357,28	21
Switch	323	12503	236,84	24
Re-execute	327	12631	616,48	17

Regarding the long runs on dataset 2, we can see a big shift in comparison to the short runs. Where the small algorithm beats all other algorithms with more than 150 points, running for an extra 15 minutes resulted in two algorithms overtaking it. This suggests that the short MILP has more difficulty finding improvements in the later stage of the optimization process. Running for an extra 15 minutes resulted in a drop of 54 points whereas the large algorithm dropped with 454 points. The best algorithm for this instance appears to be the switch algorithm. As the small algorithm works best for the shorter run and the large algorithm works best for the long run, it should be no surprise that an algorithm that uses the small MILP in the beginning and the large MILP in the end works best. Something that is noticeable as well is that the re-execute algorithm and the stuck algorithm do not perform well. Again, the stuck algorithm performs the most iterations but a lot of these iterations seem to have little to no effect. The good results obtained in the first dataset can not be replicated in this dataset for the re-execute algorithm. This may be due to the fact that the re-execute algorithm only performs a re-execution with the large MILP when a substantial mismatch is found. However, when the small MILP performs an iteration that has no mismatch with the actual plan, it does not mean that the small MILP found the best possible plan for that neighborhood. Because the small MILP does not calculate the costs perfectly, there may be a configuration that has lower costs than the small MILP is able to see. Because it does not see this, the optimizer chooses a different configuration. In this case there is no mismatch and we therefore do not perform an iteration with the large MILP. Therefore, with the re-execute algorithm a lot of interesting neighborhoods could be overlooked in the later stages of the optimization process whereas these would not be overlooked if the large MILP would be used directly.

Table 3.7: Results for the long runs of dataset 2.

Algorithm	Iterations	Mean score	St. dev.	Rollback %
Small	548	12175	487,95	36
Large	422	12102	230,80	20
Stuck	565	12345	482,78	27
Switch	531	11924	349,29	33
Re-execute	512	12287	225,13	23

Dataset 3

Tables 3.8 and 3.9 show the results for the third dataset. Recall that the difficulty for this dataset is its comprehensive supply chain and the large number of input materials to create end products. For this dataset, the short runs have a duration of 15 minutes and the long runs have a duration of 30 minutes. As the algorithms only run for a short amount of time in this dataset, being able to perform more iterations is important. The small algorithm is able to perform 46.9 % more iterations than the large algorithm on average. In the second dataset, the small algorithm performs about 25 % more iterations than the large algorithm. This indicates that the large MILP needs to find more substantial improvements in comparison with the small MILP to be able to compete. Looking at the results, this is not the case. The small algorithm performs substantially better than all other algorithms. The algorthims that combine the two MILPs come closer to the small algorithm but are not able to get on the same level. Because of the comprehensive supply chain too many variables need to be constructed to let the large MILP accurately portray the problem. This has the effect that the MILP is too slow and is not able to find good enough solutions with the limited number of iterations it has.

Table 3.8: Results for the short runs of dataset 3.

Algorithm	Iterations	Mean score	St. dev.	Rollback %
Small	119	16947	1025,46	12
Large	81	17995	590,38	10
Stuck	101	17128	1255,98	10
Switch	117	17219	937,94	14
Re-execute	106	17126	1607,59	10

Looking at Table 3.9, we can see that not much has changed in comparison with the short runs. The small algorithm performs substantially better than all other algorithms. What is interesting is that the mean score of the large algorithm for the long runs is higher than that of the short runs for the small algorithm. The large algorithm has performed more iterations than the short runs of the small algorithm by this time. This indicates that the large MILP is not working properly for this dataset. There are several iterations where the subproblem has become too complex, such that the MILP is not able to be solved within reasonable time. Therefore, an iteration is cut off before the optimizer was finished with optimizing that subproblem. Knowing this, it is not surprising that an algorithm using a combination of the small and large MILPs does not yield better results than only using the small MILP. Out of the other algorithms, the switch algorithm performs best. This is most likely due to the fact that it uses the small MILP for an interrupted time in the beginning.

Table 3.9: Results for the long runs of dataset 3.

Algorithm	Iterations	Mean score	St. dev.	Rollback %
Small	282	15120	787,90	18
Large	147	17670	508,57	20
Stuck	174	16294	1581,91	22
Switch	274	15688	1350,60	22
Re-execute	169	15861	1418,42	16

Dataset 4

Tables 3.10 and 3.11 show the results for the fourth dataset. For this dataset, the short runs last one hour and the long runs take 1 hour and 15 minutes. Comparing the small and the large algorithm for the short runs, we can see that there is a big difference in number of iterations executed. The small algorithm performs more than 250 extra iterations than the large algorithm in the same time frame of an hour. This is also reflected in their respective mean scores. This indicates that any algorithm can only compete with the small algorithm if the large MILP is used sparsely. Looking at the results of the other algorithms, we can see that the use of the large

MILP has resulted in significantly slower heuristics as far fewer iterations have been performed.

Table 3.10: Results for the short runs of dataset 4.

Algorithm	Iterations	Mean score	St. dev.	Rollback %
Small	699	14799	226,12	29
Large	425	15475	217,90	25
Stuck	574	15373	280,45	27
Switch	639	15068	106,66	21
Re-execute	493	15114	169,56	22

Regarding the long runs, we can see from Table 3.11 that the small algorithm performs best. All other algorithms struggle with the speed to be able to compete. Even in the later stages where we have seen that the large MILP has shown to perform more strongly in other datasets, it does not seem to be able to overtake the small algorithm here. With these settings, the small algorithm gets the best result and is most reliable. We can see that the switch algorithm comes close but still has trouble with the number of iterations. Regarding the stuck algorithm, this instance shows similar results as to other datasets, in the sense that using a mixture of the MILPs in this sense is more likely to harm the optimization process than to benefit it. This shows that combining MILPs needs to be done with great care.

Table 3.11: Results for the long runs of dataset 4.

Algorithm	Iterations	Mean score	St. dev.	Rollback %
Small	845	14703	144,45	32
Large	542	15065	194,91	25
Stuck	693	15264	146,58	27
Switch	767	14778	215,00	32
Re-execute	553	15103	347,38	23

What is noticeable as well is that the re-execute algorithm performs quite badly, seeing that it has executed 493 and 553 iterations for the short and long runs respectively. This number includes the re-executions which means that the number of distinct iterations that has been performed is even less than this. This indicates that too many re-executions have been performed. Knowing this, extra experiments were performed on the re-execute algorithm with the instance of dataset 4. Instead of performing a re-execution when a mismatch is measured that is larger than 15 points, we now perform a re-execution when a mismatch of more than 30 points is measured. The results of the new re-execute experiments can been seen in Table 3.12. The first thing we notice is that the average number of iterations is significantly higher for these experiments in comparison with the previous re-execution runs. Secondly, the performance is better as well. We see a mean score that is only 25 points higher than the small algorithm and with a relatively low standard deviation. The standard deviation is about 100 lower than that of the small algorithm. Therefore, it could be argued that the results of these runs are at least comparable to the results of the small algorithm. Where the short runs show comparable results, the long runs show considerable improvement on the small algorithm. The long runs are on average 60 points better than the results of the small algorithm. Next to this, the standard deviation is twice as small indicating that these results from the re-execute algorithm are more reliable as well. This indicates that using the re-execution method in a restrained way such that only the situations where there is a high improvement chance does have a positive impact on the optimization process. It also shows that it is important to know where to set the threshold to do re-executions.

Table 3.12: Results for the re-execute algorithm with the new re-execution threshold. The Diff. small column shows the difference is for these experiments with the small algorithm for their respective durations.

Length	Iterations	Mean score	St. dev.	Rollbacks	Diff. small
Short	635	14824	124,67	23	25
Long	720	14643	71,52	26	-60

Chapter 4

Conclusion

Neighborhood searches have been extensively researched in literature as they are heuristics that are applicable to a wide range of logistical and operations research problems. The VNS is a very good example of this. Hansen and Mladenović (2003) evaluate their VNS metaheurisic on seven criteria. These criteria are:

- Simplicity. A metaheuristic should be widely applicable and should not be too complex.
- Coherence. The steps taken in the metaheuristic should follow from the principles of the metaheuristic.
- Efficiency. The heuristic should provide good solutions to problems.
- Effectiveness. Good solutions should be found within reasonable time.
- Robustness. Metaheuristics should give reasonable solutions for a wide range of instances, not be perfected for particular instances.
- User-friendliness. Heuristics should be easy to understand and easy to use.
- Innovation. Heuristics should lead to new types of applications.

The matheuristic that is discussed in this thesis makes use of the principles of the VNS algorithm and is a more complex version of the VNS metaheuristic. VNS is combined with an MILP to solve the neighborhoods selected by the heuristic. In this thesis, two different MILPs have been formulated that have a different level of accurate depiction of subproblems. This is in contrast to using only a single MILP for solving the subproblems. Using the measures described above to evaluate the metaheuristic, we can see that we lose points on two aspects. Specifically, the version of the matheuristic discussed in this thesis is more complex and less user-friendly. Incorporating multiple MILPs would need a good understanding of the problem that one is solving and this heuristic would be less widely applicable. It should be possible to create an MILP for a subproblem that is less accurate and quicker than creating a completely accurate MILP.

In Section 1.2 a real life problem regarding supply chains within factories has been described. In Section 2.1, this complex real world problem regarding extensive supply chains was formulated into an MINLP. This problem is complex enough such that it qualifies to be solved with a matheuristic. Section 2.2 described the general framework of the variable neighborhood search matheuristic that has been used for the optimization process. Section 2.3 elaborated on the matheuristic by introducing the different MILP formulations, the small MILP and the large MILP. The big differences between the two MILPs is the way the supply holding costs are calculated and how thoroughly the non-linear variables are captured. In section 2.4 the different algorithms that have been tested were defined. Next to the small and large algorithms, the switch, stuck and re-execute algorithms were defined.

Examining the results of the different algorithms in Section 3.2, it is difficult to come to a clear conclusion. As the algorithms that use multiple MILPs are less user-friendly and simple, they need to be better in other categories. Looking at the categories defined, an algorithm is efficient for a certain dataset when it provides a good solution. An algorithm is effective if a reasonable solution is found within reasonable time and when the algorithm is more reliable. An algorithm would be robust if it works well on multiple instances.

Let us first examine the small and large algorithms. For dataset 1, it could be argued that the small algorthim is efficient and effective and the large algorithm is less efficient and effective. The small algorithm is able to achieve a reasonable score more quickly. Although the large algorithm is eventually able to find a better score with the same number of iterations as the small algorithm, it needs significantly longer to do so. Looking at dataset 2, it could be argued that the small algorithm is more effective but the large algorithm is more efficient. The large algorithm is able to find a better solutions for longer runs but the small algorithm is able to find better solutions for the shorter runs. For dataset 3 and dataset 4, the small algorithm is significantly more efficient and effective as the large algorithm. As the small algorithm works better overall, one could say it is more robust than the large algorithm. Because of this, the small algorithm is the one that needs to be beaten by the algorithms that incorporate both MILP types.

Examining the stuck algorithm over all datasets, we can conclude that this algorithm does not have the desired effect. When an algorithm gets stuck, employing the large MILP does not help to continue the optimization process more efficiently. It is less effective, efficient and robust than the small algorithm.

The switch and re-execute algorithm yield more interesting results. For dataset 1, the switch algorithm has similar results to the small algorithm. For the short runs it is slightly worse and for the long runs it is slightly better but with an average longer running time of 45 seconds. The best results of the switch algorithm is for dataset 2. Although the short runs yield worse results for the switch algorithm in comparison to the small algorithm, the long runs show interesting results. It performs far better than all other algorithms. Seeing that the large algorithm behaves more efficiently for the long runs of this dataset as well, it seems logical to conclude that the switch algorithm uses both MILPs optimally. In the first part, the small MILP works better and in the latter part of the optimization, the large MILP works better. However, the results of the last two datasets are worse for this algorithm. It appears that when the large algorithm is further behind the small algorithm, the switch algorithm also has more difficulty to perform well. Therefore, we could say that for specific instances the switch algorithm is efficient and effective but it does not appear to be very robust.

The re-execute algorithm has some promising results. For the first dataset, the results show efficiency and effectiveness. For the short runs the algorithm has a similar score to the small algorithm but with a lower standard deviation. The results for the long runs are better and also with a relatively low standard deviation. However, datasets 2 and 3 do not show particularly good results for the re-execute algorithm. For dataset 2 this could be because it uses the large MILP in places where it is not very useful yet. We have stated in section 3.1 that in the first part of the improvement phase, the heuristic makes big improvements. Re-executing a neighborhood might not be a good thing to do at this stage since the relative extra improvement found would be little. This might have slowed down the algorithm substantially for which it is not able to make up it the later stages of the optimization process. For the third dataset, the large MILP does not seem to be beneficial at all. However, using the re-execution very sparsely might yield marginally better results. Only in cases of extreme mismatches the re-execution could be used. We have seen in dataset 4 that differentiating in the threshold for when a re-execution takes place results in big differences. With the old settings, the re-execution takes place too often and it is not able to perform well. With the new settings, the re-execute algorithm is able to perform comparable to the small algorithm for the short runs and substantially better for the long runs. Therefore, it could be argued that the re-execute algorithm is efficient and effective for several instances and is relatively robust.

The main research question of this thesis was whether using multiple MILP formulations within a VNS improves the performance of an optimization process. We can conclude that combining two MILPs has not resulted in algorithms that perform clearly better than algorithms using a single MILP. However, we can also see that on several occasions, a combination of MILPs has yielded improvements in the optimization process. The biggest problem is with the robustness of these algorithms. Before using a certain algorithm, it is important to know the specifications of the dataset. We have seen that in a dataset where a small MILP performs best in short runs and a large MILP performs best in long runs, a switch algorithm could be beneficial for the optimization process. We have also investigated what the effect was of re-executing the same neighborhood within the VNS with the more accurate MILP. This approach produced The most promising results. It seems that using a re-execution at the right times yields improvements but there is a fine line with finding improvements and using the re-execution too often. Further research could find out if the re-execute algorithm is robust and if it is able to be used in different problems effectively. Perhaps using a combination with the switch and re-execute algorithms could be interesting. One could start with re-execution after the improvement by the re-execution becomes a larger part of the total improvement found in an iteration.

References

Burke, E., De Causmaecker, P., Petrovic, S., & Berghe, G. V. (2003). Variable neighborhood search for nurse rostering problems. In *Metaheuristics: computer decision-making* (pp. 153–172). Springer.

Della Croce, F., Grosso, A., & Salassa, F. (2014). A matheuristic approach for the two-machine total completion time flow shop problem. *Annals of Operations Research*, *213*(1), 67–78.

Della Croce, F., & Salassa, F. (2014). A variable neighborhood search based matheuristic for nurse rostering problems. *Annals of Operations Research*, *218*(1), 185–199.

Erromdhani, R., Jarboui, B., Eddaly, M., Rebai, A., & Mladenovic, N. (2017). Variable neighborhood formulation search approach for the multi-item capacitated lot-sizing problem with time windows and setup times. *Yugoslav Journal of Operations Research*, *27*(3), 301–322.

Hansen, P., & Mladenović, N. (2003). Variable neighborhood search. In *Handbook of metaheuristics* (pp. 145–184). Springer.

James, R. J., & Almada-Lobo, B. (2011). Single and parallel machine capacitated lotsizing and scheduling: New iterative mip-based neighborhood search heuristics. *Computers & Operations Research*, *38*(12), 1816–1825.

Woo, Y.-B., & Kim, B. S. (2019). A genetic algorithm-based matheuristic for hydrogen supply chain network problem with two transportation modes and replenishment cycles. *Computers & Industrial Engineering*, *127*, 981–997.