Erasmus University

MASTER'S THESIS

Adaptive product classification for inventory optimization in multi-echelon networks

Author:
Francesco Russo
student n. 423992

Supervisors:
Prof. Dr. Ir. Rommert Dekker
Dr. Ir. Michiel Jansen

A thesis submitted in partial fulfillment of the requirements for the degree of Master of Science in Operations Research and Quantitative Logistics

June 13, 2019

Abstract

Classification of products is often done when companies wish to use a differentiated policy for a certain planning decision (e.g., replenishment or demand planning). For this reason, a classification criterion needs to contain the dimensions that are most relevant for the planning decision. For example, ABC classification is often used on the dimension of sales volume, value, or both. Sometimes it is clear what the relevant dimensions are for classification, however, the classification for replenishment policies in multi-echelon networks is far from trivial. Identifying the most important features becomes challenging. One way to tackle this problem is to find which features determine the shape of the trade-off curve between safety stock and total inventory costs.

The object of this thesis is to propose an alternative method to classify products in multi-echelon networks. Such a method allows lower inventory costs to be achieved at high service levels, and it also provides much insight into the important characteristics of the supply chain structure. In addition, the thesis compares the optimal inventory costs that are calculated using the proposed method versus the optimal inventory costs calculated using the ABC–XYZ product categorization method. The genetic algorithm is the tool that is used for optimizing inventory costs while ensuring high service levels.

Contents

1 Introduction **3**

2 Multi-echelon inventory optimization **5**
2.1 Multi-echelon inventory optimization 5
2.2 Simulation model 6
2.3 Safety stock adjustment procedure 10

3 Problem description **11**

4 Literature review **14**
4.1 Product Classification 14
4.1.1 Clustering algorithm 16
4.1.2 Classification algorithm 17
4.2 Multi-echelon inventory optimization 17
4.2.1 Optimality 17

5 Methodology and approach **19**
5.1 Network setup and policy definition 19
5.2 Products classification 19
5.2.1 Product features 25
5.2.2 Clustering algorithm using k-means 27
5.2.3 Classification algorithm using decision trees 29
5.3 Optimal safety stock levels 31
5.3.1 Genetic algorithm 32

6 Results **37**
6.1 Case introduction: Three-stage distribution network 37
6.2 Simulation model settings 37
6.3 Product classification 39
6.3.1 Full factorial experiment 39
6.3.2 Clustering via k-means 43
6.3.3 Classification via decision trees 45
6.4 Three-echelon inventory optimization 48
6.4.1 Initial population for the genetic algorithm 48
6.4.2 Benefits due to initial population and class crossovers 52
6.4.3 Optimal k-factors 52
6.4.4 Genetic algorithm comparison with ABC-XYZ 56
6.5 Overall approach 59

7 Conclusion **62**

Appendices **65**

A Genetic algorithm initial population **66**

1 | Introduction

It is common practice for companies to make use of inventories due to a mismatch between supply and demand. However, having too much inventory on shelf is something that companies want to avoid. One reason for this is, inventory is capital invested, and products can be out of market in a relatively short time, especially for certain industries such as electronics and food. However, the business market has an extremely competitive and uncertain nature, and in this matter, a high level of inventory increases a company's responsiveness to market uncertainties. Moreover, high inventory levels can help companies to improve customer service, lower ordering costs and setup costs, maximize labour and equipment utilization, prevent stock-outs, and subsequently do not miss out on business opportunities that are available.
Companies manage inventories using policies, and once a certain policy is defined, it is considered good practice to categorize products. Categorization is done for mainly three reasons, namely 1) to reduce complexity, specifically in reducing the number of decision variables from possible thousands to a reasonable number; 2) diversification, which refers to recognising differences among products, and 3) managerial reasons, since it is easier for planners to focus on a reduced number of product categories.

One of the most popular methods used for classifying products in inventory management is the ABC analysis [13]. This method is popular in the industries because it is simple to implement in practice and it provides good results when applied to single stage inventory optimization. However, ABC is arguably not the best classification method when applied to multi-stage inventory optimization [33]. This is because potential network-dependency properties are not taken into account in the ABC categorization. For this reason, this thesis aims to develop a method for product classification that is especially designed for multi-echelon networks. In such networks, we look at the supply chain holistically while keeping in mind that network inventory is about serving customers, but this involves interdependencies between inventories that are stored upstream and downstream in the network. Consequently, the first research question is as follows:
How to develop a product classification method that adapts to the underlying multi-echelon network but with minimal penalty?
In the research question we mentioned the word "penalty" since optimizing decision variables on product categories is already sub-optimal compared to the case where decision variables are optimized for each product individually.

After a clear classification of products is established, companies are often interested in finding the optimal level of inventories which guarantees high customer service while minimizing costs. Usually this is achieved by determining the correct size of the safety stock levels, which are used for reducing the risk of a stock-out. Each station in a multi-echelon network corresponds to a real location where stocks can be allocated. Part of this thesis is to optimally allocate safety stocks across the stations of the multi-echelon network in order to reach the target service level at the lowest costs. The policy used in this thesis to determine the right size of safety stock levels is in the following equation:

$$SS_{m,n} = k_{m,i} \cdot \sigma_{(L_{m,n}+R_{m,n})} \tag{1.1}$$

where $SS_{m,n}$ stands for safety stock level of product n at station m, $k_{m,i}$ is the safety factor of class i (to which product n belongs) at station m and $\sigma_{(L_{m,n}+R_{m,n})}$ is the standard deviation of demand

of product n during replenishment lead time (L) plus review period (R) at station m. Equation 1.1 also shows the importance of using product class—the optimal size of safety stocks of each product is set via the optimal value of safety factors, which are defined per class of products. The choice of defining safety factors per class of products reduces the number of decision variables drastically.

Multi-stage inventory optimization has proven to be challenging. To tackle this, van Liempd (2016) [34] proposed an interesting approach, and his method consists of using a simulation model in combination with the genetic algorithm (GA). The simulation model was used for evaluating the inventory levels of multi-echelon networks under different sets of safety stock levels. GA was used to determine the optimal set of safety stock levels that minimize the network inventory under service level constraints. Following the work of van Liempd, the GA is the tool used in this thesis to fine-tune the safety factors of equation 1.1 to optimal values.
One of the major difficulties in implementing the GA is that many decisions need to be made. For example, a crucial decision is regarding how to properly perform crossovers. Generally speaking, we want to make more informed decisions based on the given supply chain structure. For this reason, it is essential that we learn about the given supply chain structure while answering the first research question, and subsequently we would use what was learned to make more informed decisions for the GA. This point leads to the second research question of this thesis:
How can basic GA characteristics be improved by learning from the underlying supply chain structure, and how can this information be used to make more informed decisions?
In this research, we will focus on improving two GA characteristics: initial population and crossovers.

It is important to mention that on the one hand, companies should optimize the inventory levels to reduce holding costs, while on the other hand they should also aim to reach a high level of customer service. For this reason, this study adopts a two-step approach. Firstly, we optimize the safety factors of all the stations except the stations that are the closest to the customers— these are the downstream stations in the network. Secondly, we adjust the safety stocks of the downstream stations such that a certain target service level is reached. In this way, the safety factors of the downstream stations are not considered as decision variables.

The objective of this thesis is to develop a general method for the classification of products. The idea is to propose a classification method that suggests what are the most important product/network dimensions to use for categorization purposes. In the ABC categorization, the starting point is to use products' value as segmentation dimension. However, is value always the best product dimension to use when it comes to categorize products to solve a certain problem (e.g. minimizing inventory costs for multi-echelon networks)? Arguably not. This is why we propose a method that first understands what are the main dimensions that drive, for example, inventory costs and only afterwords we classify products based on those dimensions. To do this, we use machine learning techniques to build a flexible method that adapts to the underlying supply chain network and provides the main dimensions for categorization. The optimal inventory costs of the proposed method are compared with the optimal inventory costs based on the ABC–XYZ product categorization method [39]. While the multi-echelon network that we consider in this thesis is a distribution network, the proposed approach can be extended to production networks as well.

This research is conducted at EyeOn BV, a consulting firm with its headquarters in the Netherlands. EyeOn specializes in planning and forecasting; in the recent years, the company has invested much effort and interest in the topic of multi-echelon inventory optimization (MEIO). Indeed, this thesis can be considered as a continuation of the work done by van Liempd [34] at EyeOn in the field of MEIO.
This thesis is outlined as follows: Chapter 2 describes an approach for MEIO, and in Chapter 3 the problem is described in more details. Subsequently, Chapter 4 discusses the relevant literature, and the detailed approach and methodology are presented in Chapter 5. Results are shown in Chapter 6, and finally conclusions are made in Chapter 7.

2 | Multi-echelon inventory optimization

Multi-echelon networks are characterized by considering the dependency among stations in the supply chain network. In other words, the holistic approach of multi-echelon networks is to focus not on the performance of each single stocking point but on the overall performance of the supply chain network [1].
In the context of a multi-echelon network, each level of the network is referred as an *echelon*. The example in Figure 2.1 shows a three-echelon network. At Echelon 3, we have the central distributor, while at Echelon 2 there are the local distributors, and at the lowest level at Echelon 1 there are the retailers. Moving *upstream* in the network means moving from a lower to a higher echelon; likewise, moving from a higher to a lower echelon corresponds to moving *downstream* in the network.

Moreover, to be able to understand if a product categorization works better than others, we need an evaluation function that is able to test the network performance when safety factors are defined per product categories. Furthermore, to improve the GA, we need the evaluation function to determine if a given safety stocks allocation provides better results than different allocations. In this thesis, the evaluation function is represented by a simulation model. Moreover, since building a simulation model goes outside the scope of this thesis, we make use of the simulation model proposed by van Liempd. More precisely, the whole approach behind multi-echelon inventory optimization is inherited by the job of van Liempd. In this chapter, the multi-echelon inventory optimization approach used by van Liempd is summarized.

2.1 Multi-echelon inventory optimization

MEIO aims to minimize the inventory costs of the whole network while reaching high service levels at the customer level (i.e., retailer level). The general approach used in a multi-echelon is to optimize inventory control parameters by considering all stages in the supply chain network simultaneously [24]. The general multi-echelon approach is different from the single echelon approach where the inventory decision parameters are optimized at each stage individually and independently from other stages [30].
Van Liempd [34] suggested an interesting solution on how to deal with inventory optimization in a multi-echelon network. His approach in minimizing inventory costs was to optimally allocate safety stocks at all locations in the network. To do this, he used the GA to find the optimal allocation of safety stocks throughout the supply chain network. However, using the GA with decision variables that are represented by safety factors for each upstream product–location combination was not feasible, mostly due to a long computing time. The compromise used by van Liempd was to optimize safety stocks allocation via the use of safety factors, which were defined by each product group–location combination. This approach lead to the use of the safety stock policy that is described in equation 1.1.
In his study, van Liempd used an ABC–XYZ [39] categorization to form nine distinctive group of products for each upstream location in the multi-echelon network. This meant that for each upstream

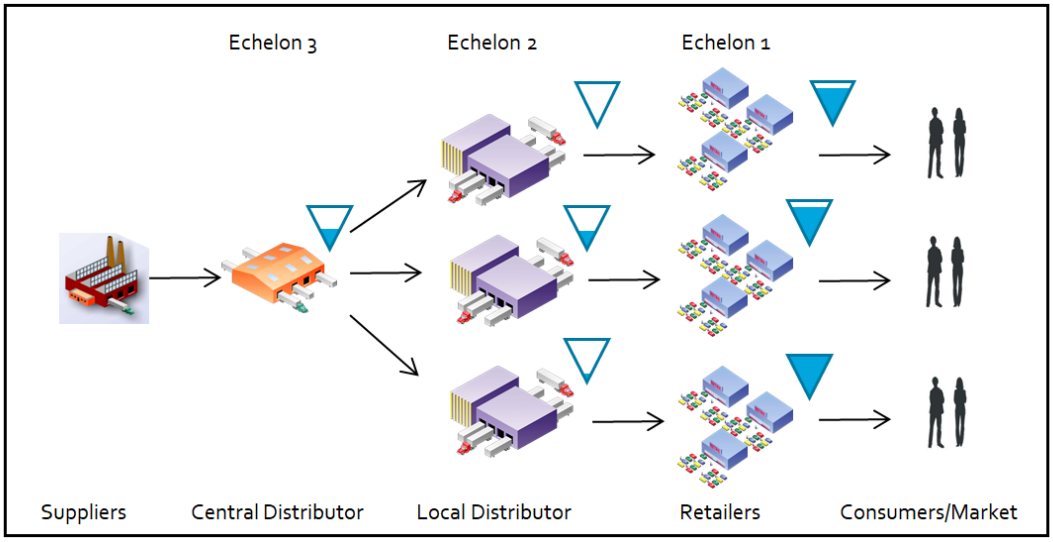


Figure 2.1: Three-echelons distribution network.

location in the network, there were only nine variables to be optimized, and these variables were in the form of safety factors.
Moreover, to make use of the GA, van Liempd needed an evaluation function that can quantify the inventory costs of the whole network corresponding to certain safety stocks allocation. The evaluation function used in his work was represented by a simulation model that was combined with a procedure called safety stock adjust procedure (SSAP). The simulation model and the SSAP will be discussed in detail in the next sections. The simulation model provides a cost solution to a given network scenario; a scenario involves allocating safety stocks in the network to certain values, and multiple scenarios are then simulated. Each scenario with its corresponding inventory costs is then sent to the GA, which generates a new set of "fitter" scenarios via proper crossovers. These fitter scenarios are then used as input scenarios for running new simulations. The procedure is repeated until an optimal solution is found.

2.2 Simulation model

Building a simulation model for multi-echelon networks is out of scope for this thesis. However, since the van Liempd's simulation model is extensively used in this thesis, it is worthwhile to present an overview of all the main characteristics of the model itself (please refer to van Liempd's work [34] for more details).

Simulation is a powerful tool that helps us to analyse the performance of a multi-echelon network for a given scenario. In our case, a scenario corresponds to the case where safety stocks are set to certain values across the network. The simulation model allows us to evaluate the network inventory costs that correspond to a given scenario.
The model proposed by van Liempd is a *discrete-event simulation*. Discrete-event simulation involves modelling a system in which the state variables change instantaneously at separate points in time [2]. An event can occur at a specific moment in time, and it can determine the change of the system's state. The assumption here is that between consecutive events, the state of the system cannot change. In our simulation model, we considered only three possible events, namely customer demand, replenishment order, and the fulfilment of a replenishment order. However, a time order of events should be given since some events trigger other events. For example, a replenishment order is placed if there are not enough stocks to satisfy demand directly. Similarly, a fulfilment of a

replenishment order can only take place after a replenishment order has been made. Moreover, in case of simultaneous events, we prioritize events has shown in Figure 2.2. We first work from the lowest echelon upstream having each location placing its orders, and then we work from the highest echelon downstream having each location fulfilling its orders.
Moreover, the simulation model uses a periodic review to monitor inventory levels. In this study, the periodic review frequencies of product–location combinations are given as input parameters.

Demand modelling

In the simulation model, we make a distinction between *dependent* and *independent* demand. Independent demand is customer demand which in this thesis is assumed to take place only at the downstream stations in the network. On the other hand, dependent demand depends on the replenishment orders of a downstream station to their supplying upstream station. In other words, the demand of the upstream stations depends on the replenishment orders of the connected downstream stations. Figure 2.1 shows clearly that only the retailers face customer demand; local distributors face the demand that depends on the retailers' replenishment orders, and the central distributor faces the demand that depends on the local distributors' replenishment orders.

Customer demand (or independent demand) is immediately fulfilled if there are enough stocks available, otherwise demand is placed in the backlog and a replenishment order will take place at the next review period. Moreover, if at a given station some backlog is still left remaining, this backlog is fulfilled first before any new demand. Customer demand is assumed to be normally distributed with a known mean and standard deviation.

All the upstream locations in the network face indirect demand in the form of replenishment orders. This means we assume that upstream locations do not face any direct customer demand. If a replenishment order from a downstream location d to an upstream location u takes place at time t, the order will be fulfilled at time $t + L_{d,u}$ only if there are enough stocks available at location u at time t where $L_{d,u}$ is the lead time between station d and u.. If at time t there are not enough stocks at location u, the order is then backlogged, and its fulfilment will happen at a later time than $t + L_{d,u}$.

In the simulation model developed by van Liempd, there is no rationing or partial fulfilment of demand. In other words, if there is not enough inventory to fulfil a certain demand, the demand is simply backlogged, and its fulfilment order will take place at a later time.

Moreover, the external supplier is assumed to always have stocks available. This means that once an order is placed, it is always delivered after a time that is equal to the lead time between the upstream location and the supplier location.

Ordering policy

The ordering policy used in the van Liempd simulation model is known as an (R, S) policy [3], where R is the review period and S is the order-up-to level. At a specific location, a replenishment order at period t is placed only if t is a review period and only if the inventory position of the previous period is less than the order-up-to level S plus all the known incoming demand at time t. More precisely, at review period t a replenishment order at location m for product n is made if the following condition is fulfilled:

$$IP_{m,n}(t-1) < S_{m,n} + D_{m,n}(t)$$

where $IP_{m,n}(t-1)$ is the inventory position at time $t-1$ of product n at location m, $S_{m,n}$ is the order-up-to level of product n at location m, and $D_{m,n}(t)$ is the known incoming demand of product n at location m at time t. If the above equation is satisfied, the order quantity of product n at location m at review period t is then equal to the following:

$$Q_{m,n}(t) = S_{m,n} + D_{m,n}(t) - IP_{m,n}(t-1)$$

In the (R, S) policy the order-up-to level S needs to be defined. The order-up-to level $S_{m,n}$ of product n at location m is directly related to the safety stock $SS_{m,n}$ according to the following formula:

$$S_{m,n} = SS_{m,n} + \mu_{L_{m,n}+R_{m,n}} \quad \forall n, m \tag{2.1}$$

where $\mu_{L_{m,n}+R_{m,n}}$ is the mean demand of product n and location m under the lead time (L) and review period (R).

Safety stock levels

The average customer demand of product n at downstream location m during the replenishment lead time and review period is defined as $\mu_{L_{m,n}+R_{m,n}}$, while the standard deviation of customer demand of product n at downstream location m during the replenishment lead time and review period is defined as $\sigma_{L_{m,n}+R_{m,n}}$. Given these statistical characteristics, we can fit a normal distribution and can thus generate customer demand paths using different seed numbers. By propagating these demand paths upstream in the network, we are able to determine the mean demand and standard deviation of the demand of product n at the upstream station m, denoted by $\mu^*_{L_{m,n}+R_{m,n}}$ and $\sigma^*_{L_{m,n}+R_{m,n}}$ respectively. Once we know the mean and standard deviation of the demand at the upstream and downstream stations, we can compute the safety stock level for each product–location combination. The safety stock for product n at upstream location m is defined as follows:

$$SS_{m,n} = k_{m,i}\sigma^*_{L_{m,n}+R_{m,n}} \quad \forall i; \forall n, m \in J_u \tag{2.2}$$

where $k_{m,i}$ is the safety factor at upstream location m for product group i (to which product n belongs), and J_u is the set containing all the upstream location indices.

The safety stocks at the downstream stations are adjusted by the SSAP, which will make sure to set the safety stock at a certain level such that a given target service level is reached. However, before performing the SSAP, it is necessary to initialize the safety stock levels such that the simulation model can be used. The safety stock of product n at downstream location m is initialized as follows:

$$SS_{m,n} = \Phi^{-1}(\gamma_{target_{m,n}})\sigma_{L_{m,n}+R_{m,n}} \quad \forall n, m \in J_d \tag{2.3}$$

where $\gamma_{target_{m,n}}$ is the target cycle service level of product n at downstream location m, $\Phi^{-1}(\gamma_{target_{m,n}})$ is the inverse of the standard cumulative distribution given probability $\gamma_{target_{m,n}}$, and J_d is the set containing all the downstream location indices.

Input parameters

The *warm-up* period and the *run length* are two input parameters for the simulation model. Since we are dealing with a non-terminating simulation, a clear stop criterion should be given by the run length. It has been said that non-terminating models converge to a steady state for a long run length [2]. Since we only want to start analysing the network when it reaches the steady state, we will use a warm-up period. This means that the system starts being analysed only after the warm-up period. In this way, we cause the system to be independent from certain starting configurations, such as initial stock levels.

Moreover, all the safety stock levels at all upstream locations should be given as input to the simulation model. Each scenario corresponds to a specific allocation of safety stock levels at upstream locations. As already stated, the safety stock levels of upstream locations depend on the safety factor variables as defined per product category. It would be up to the GA to find the optimal values of those variables that correspond to the optimal scenario for the given supply chain network.

Service level

The main objective of inventory optimization is to lower the inventory costs under the target customer levels constraints. In this research, the *cycle service level* [30]—also known as P_1—is used as the

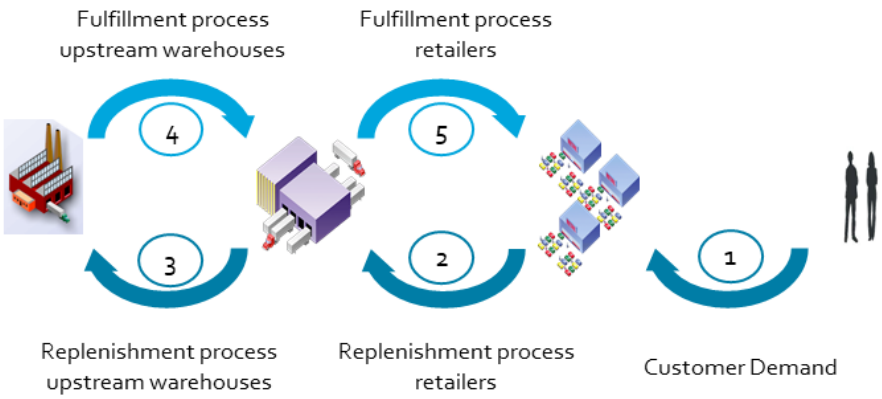


Figure 2.2: Prioritization of simultaneous events in the simulation model.

main metric to represent the customer service level. The cycle service level measures the probability of not having a stock-out during a replenishment cycle.
It is important to mention that only the downstream locations in the network face service level constraints since they are directly connected to customers' demand. This means that upstream locations are free from any service level constraints since they do not experience any direct customer demand.

Simulation output

Once a simulation run is performed, different statistics are given as output. The most important outputs for us are the realized service levels at the downstream locations and the actual inventory costs of the network.
Our objective is to have the realized service levels at downstream locations to be as close as possible to the target service level. At present, the simulation model itself does not guarantee a match between the realized service level and the target service level. For this reason, we perform the SSAP as an extra step in order to be able to reach the desired service level. The SSAP is described in the next section.

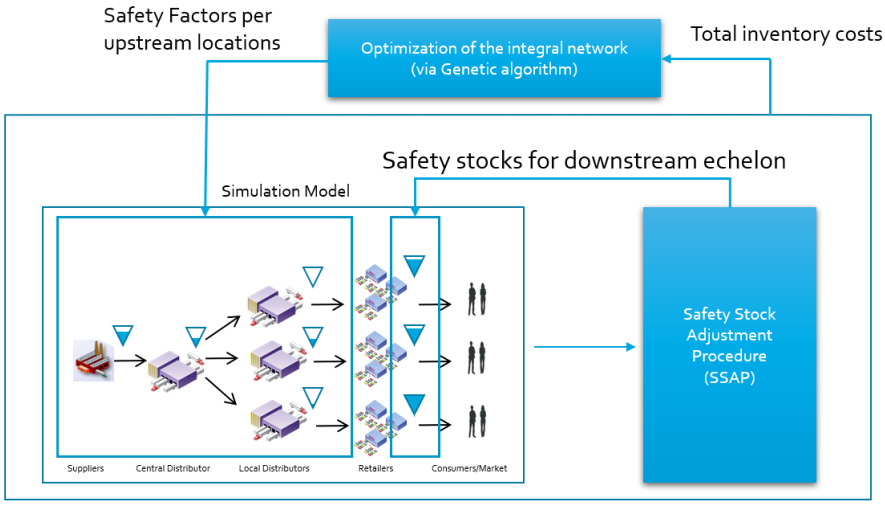


Figure 2.3: Inventory optimization scheme for multi-echelon networks.

2.3 Safety stock adjustment procedure

The SSAP [15] is a procedure needed to reach a certain target service level at the downstream locations in the network. If we would only use the GA in combination with the simulation model, we would find optimal solutions that may not guarantee the desired customer service level. For this reason, the SSAP is always performed after each simulation run.
In conclusion, the SSAP makes sure that the safety stock levels at the downstream locations are adjusted to the right level in order to reach the right target service level.

The overall procedure that involves the GA, the simulation model, and the SSAP can be summarized in the following steps:

- **Step 1:** The GA propose n random initial solution scenarios. An initial solution corresponds to a set of safety factors for the upstream stations.
- **Step 2:** Safety stock levels for upstream and downstream locations are determined as described by equations 2.2 and 2.3 respectively. A number of n simulations are performed, one for each scenario.
- **Step 3:** The SSAP is performed n times, one for each scenario. The safety stock levels at the downstream locations are adjusted to reach a target service level.
- **Step 4:** The simulation scenarios are performed again n times where the safety stock levels at the downstream locations this time are given by the SSAP but not by equation 2.3.
- **Step 5:** The inventory costs of each scenario are used for selecting a new set of n fitter scenarios.
- **Step 6:** Repeat Steps 2 to 5 until a stopping criterion is met.

The whole procedure is also schematically represented in Figure 2.3. The same procedure and assumptions described in this chapter will still apply for the remaining part of this thesis.

3 | Problem description

This thesis firstly shows a new approach on how to classify products in a multi-echelon network, and secondly it shows an improvement of an existing method on how to set safety stocks to optimal levels.

Product classification comes at the penalty of sub-optimality since policies are designed for groups of products and not for individual products. Nevertheless, categorization is needed for practical reasons. For example, optimization algorithm methods are generally sensitive to the number of decision variables, and having too many variables may lead to infinitely slow algorithms. In addition, companies prefer to use policies for a certain planning decision (e.g., replenishment or demand planning). A classification criterion needs to contain the dimensions that are most relevant for the planning decision. For example, ABC classification [13] focuses on dimensions such as sales volume, the value of products, or both; XYZ classification [10] focuses on dimensions regarding the forecastability of a product's demand. In certain situations it is clear what the relevant dimensions are for classification; however, product classification for replenishment policies in multi-echelon networks is far from trivial. For example, the best replenishment strategy probably does not only depend on the volume of products that flow through each station, but it also depends on the network structure. Identifying the most important features becomes challenging. One way to tackle this problem is to find which features determine the shape of the trade-off curve between the safety factor and total costs.
Figure 3.1 shows examples of trade-off curves between the total inventory and the safety stocks, the latter of which are linearly related to the safety factors according to equation 2.2. The figure shows two clear dynamics: for the red product, we see that the total network inventory grows by increasing the safety stocks at the central location, while for the blue product the total network inventory reduces as the safety stock at the central location is increased. The dynamic of the blue line appears to be counter-intuitive, and thus a question we can ask is, how is it possible that the total network inventory diminishes as the safety stocks increases centrally?
The blue line behaviour would be impossible in a single station situation, but this trend is possible in a multi-echelon network. Looking at the network example on the right of Figure 3.1, it is possible to eventually reduce the total network inventory if we increase the safety stocks at the central location and reduce the safety stocks at the retailers level.
In this thesis, we refer to the expression *product behaviour* in describing the scaling of a product's total network inventory with safety stocks (or safety factors). Figure 3.1 shows two examples of product behaviours.

Since policies are set per group of products, it is preferable to classify together products that have similar behaviours in determining the total inventory. In other words, we prefer to group products together that have similar trade-off curves between inventory levels and safety factors. Looking at the example of Figure 3.1, the blue and the red products show completely opposite behaviours; hence, they should not be in the same category.
Moreover, in an actual multi-echelon network, the situation is not as simple as is shown in Figure 3.1. Figure 3.1 refers to the situation of a two-echelon network where there is only one upstream location. Safety factors at the retailer level do not represent any decision variable since at that level, the SSAP is used to set the safety stocks to the right levels such that a certain target service level is reached.

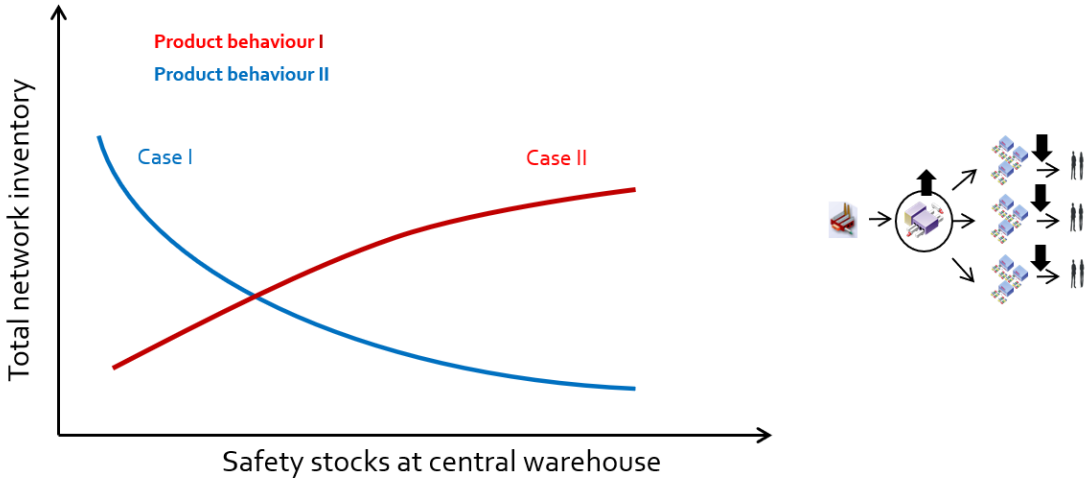


Figure 3.1: Product behaviours in two cases. The left side shows two potential dynamics of the total network inventory with safety stocks (or safety factors) for two different products. The right shows a two-echelon network example where the highlighted location represents the upstream location in the network.

Finding the trade-off curve for networks with more than two echelons becomes much more challenging. This is because for all upstream locations, decisions need to be made in terms of setting the optimal values of the safety factors. This means that the relation between network total inventory and safety factors is represented by a multidimensional curve. Products with similar multidimensional curves should be classified together as the similar curves imply similar behaviours.
As a second step, we aim to find out which dimensions play a role for the shape of the trade-off curves. First, we cluster together products with similar behaviours and we then try to identify which are the most relevant dimensions or features that explain the similarities or dissimilarities.
This thesis aims at developing a classification method that adapts to the underlying supply chain network. The goal is to create a new method that performs better than the commonly used methods such as the ABC classification. This leads to the first research question formulated in the introduction.

The second part of this thesis is focused on improving an existing method used to set safety stocks to optimal levels. Finding the optimal safety stock levels for multi-echelon networks can be done using exact mathematical programming (MP) formulations. Moreover, some quantities such as customers' demand can be uncertain, and stochastic programming methods should be used to solve the complete problem in an exact optimal way. However, for instances that involve a large number of decision variables, it would not be possible to solve MP problems in a reasonable time [16]. This makes MP attractive only for small-sized problems.

Analytical methods could also be used in the field of inventory optimization. However, although analytical models offer a very insightful way to show relations, they do have drawbacks. Among them, two can be mentioned:

1. Assumptions need to be made. For example, one assumption that can be made in analytical models is an infinite production capacity. Moreover, some important real scenarios are not always covered in analytical models; an example here is when shipment in full containers is required.

2. Analytical models can be used as an approximation of real scenarios. However, it becomes hard to estimate the error when analytical models deviate from real scenarios.

Generally speaking, there are often good analytical models for inventory optimization; however, we often cannot afford to make huge model adjustments for each new analysed instance.
To overcome the MP and analytical restrictions, simulation is used in this thesis. Simulation is considered an expensive way to calculate the outcome of a function; it can be time consuming, and model complexity increases according to the level of accuracy that need to be achieved. However, it is a widely used tool since it has several advantages. For example, it can deal with uncertainties, and it can approximate real scenarios at a high accuracy. Moreover, simulation models are usually validated. This means that before proceeding with parameters optimization, simulated outputs are compared to observable results. In this way, it is possible to know how reliable the simulation model is.

However, simulation model is about approximating real scenarios, and no parameters optimization is done with simulation. For this reason, in this thesis we make use of a model that combines simulation with an optimization algorithm. The optimization algorithm used is the GA [20], which is a widely used metaheuristic method that currently has many applications in the field of supply chain. The overall inventory optimization scheme is represented in Figure 2.3, and this scheme includes simulation, SSAP, and GA.
The drawback of the GA is that it explores the solutions field in a random way, and sound solutions are more likely to be selected in order to generate better possible solutions. However, while this method can lead to close to optimal solutions, there is a penalty that it may take a long time before reaching them. Optimal solutions for us are the optimal values of safety factors allocated to each product group and upstream location combination.

GA is the method used by van Liempd for safety stock optimization. It is considered effective but still not efficient yet, which limits the scope of problems that can be addressed. Efficiency can be improved in a few ways, such as by performing better crossovers by "learning" about the structure of the supply chain network. For example, in agreement with the *risk pooling effect* [31], a product's demand variability at the downstream locations of a multi-echelon network can be reduced by aggregating demands. Therefore, more stocks can be stored centrally in the supply chain network. On the other hand, if demand at the downstream locations is stable, more stocks can then be placed towards the downstream locations. By exploring the supply chain network, we could learn something about the structure of the network itself. For example, we could learn whether the optimal inventory levels lean more towards a centralized or decentralized stock system, and such information could be used to make more informed decisions in the GA. In this way, good results could be achieved in a shorter time, and larger instances could be addressed. This leads to the second research question formulated in the introduction.
Moreover, by answering the first research question, we can already capture some useful information about the supply chain network. We could then use this information to explore the second research question.

4 | Literature review

After formulating the research questions, it is worthwhile to go through similar research that have been formulated in the past.
Van Liempd [34] primarily focused on the use of the GA to find optimal safety stock levels for a multi-echelon network. While the proposed method is effective, it is still not efficient yet, and this inefficiency limits the scope of problems that can be addressed. On the other hand, little effort has been spent on product classification, and thus this thesis can be considered as an extension of van Liempd's work since more emphasis is placed on product classification for multi-echelon networks, and also an improved version of the GA is proposed to make the method more efficient.

4.1 Product Classification

In literature, no ad-hoc methods can be found regarding product classification for multi-echelon networks. De Kok et al. (2018) [8] and Simchi-Levi et al. (2011) [32] provide and extensive literature review of stochastic multi-echelon inventory systems. Furthermore, in both papers the topic of product classification is not covered since all the inventory control policies are not defined per classes of products. However, in this thesis we are going to use the policy described by equation 2.2 in which the safety factor variables are defined per class of products.
One of the most popular methods used in supply chain for the classification of products is the ABC analysis. The method was introduced for the first time by H. Ford Dickie [13]. The idea behind this method comes from the Pareto principle, which states that "for many events, roughly 80% of the effects come from 20% of the causes" [35]. In the ABC analysis, products are ranked in either categories A, B, or C based on their importance, with A-products being the most important and C-products being the least important. The classification of a product into the A, B, or C categories has generally been based on a single criterion, namely dollar value per unit multiplied by annual usage rate; this is commonly known as dollar usage [29].
However, more advanced multi-criteria ABC analysis have been proposed over time. In particular, Guvenir et al. (1996) [17] proposed a multi-criteria inventory classification using GA. Multi-criteria were considered, and their weights were optimized via GA. Nearly a decade later, Ramanathan (2004) [28] proposed an ABC inventory classification with multi-criteria using weighted linear optimization. Moreover, van Liempd used an ABC–XYZ [39] classification where products are divided into nine categories based on their demand's value and demand's volatility. While ABC classifies products according to their value, the XYZ method classifies products according to their forecastability [10]. A criterion to determine the forecastability of a product is to look at the coefficient of variation, which is the ratio between the standard deviation demand and mean demand. A product with a low coefficient of variation is generally characterized by a stable demand pattern. The way to discriminate between X, Y, and Z-products is by defining three coefficient of variation thresholds. Usually X-products are the easiest to forecast as they have a low coefficient of variation, while Z-products are the most volatile with a high coefficient of variation; Y-products are found between the two as a compromise between X-products and Z-products. It should be mentioned that the ABC analysis is widely used in practice mostly due to its simplicity. However, it does not provide good performance when applied to complex multi-echelon networks [33].

Multi-echelon networks are characterized by multiple stations. For each upstream station, there are decision variables, and these correspond to safety factors that need to be determined. In principle, we are able to determine the dynamic of a product in the network by studying how its corresponding total network inventory would scale with the safety factors of upstream stations. The way in which a product–network inventory responds to change in safety factors is described by its multidimensional curve. Each product has its corresponding curve, which has a dimension equal to the number of upstream stations plus one dimension corresponding to its total network inventory. Our approach is to classify and group products with similar multidimensional curves since we use a safety stock policy that is optimized per product category (see equation 2.2). In this regard, we use Euclidean distance as a similarity measure. The smaller the distance is between two multidimensional curves, the higher the likelihood is to have their corresponding products in the same category.

When it comes to inventory optimization, usually little effort is spent on product classification since existing techniques are used. In literature, many ABC categorization applications can be found. Apart from this, literature is generally lacking in the context of product categorization for multi-echelon networks. For this reason, this thesis proposes an innovative approach that uses machine learning techniques. Moreover, product similarities are established by comparing curves, which unfortunately are not displayable due to their high dimensionality, and for this reason we cannot rely on our visualization skills to make proper comparisons. The principle behind machine learning is that we enable computers to carry out a task which is not driven by hard programming but by examining data. The main idea is that data drives the computer while keeping the algorithm at a general level. Moreover, machine learning is becoming popular because of the ability of computers to "discover" and "learn" as well as being able to go through millions of records effortlessly.

For this research, two types of machine learning methods are of interest, namely unsupervised learning and supervised learning [12]. Unsupervised learning refers to cases when computers have no information about the given data but are still able to find any meaningful groupings and patterns within the data. This means that unsupervised learning algorithms have a descriptive task that is used to find similarities or patterns within the data that are not known beforehand. By contrast, supervised learning algorithms have predictive tasks, fitting rules are usually applied to known data. Subsequently, the algorithm learns those rules which can be applied to new data, and predictions can be made.
Common unsupervised learning algorithms are represented by clustering algorithms [12]. Clustering methods group similar data into the same cluster; more precisely, data in the same cluster are considered as similar, while data in different clusters can be considered as dissimilar. Along with some input parameters, a similarity metric should be provided to a clustering algorithm. One example is to use Euclidean distance as a similarity metric. The objective of clustering algorithms is to minimize distances between data in the same cluster and to maximize distances between data belonging to different clusters.
Clustering comes in handy for the purposes in this thesis, as product multidimensional curves are discretized in a finite set of points. Evaluating Euclidean distances between discrete multidimensional curves allows us to establish if the corresponding products show similar behaviours. More precisely, we aim to cluster together products that have their multidimensional curves relatively close to each other. Since products in the same cluster have similar behaviours, the same inventory policy should be applied to them. For this reason, products in a cluster should be seen as products belonging to the same category. This means that the total number of clusters will likely tell us the total number of categories that should be used.
Once clustering is performed, the next step is to train a supervised learning algorithm to learn about the clustering of the products and their features. In this way, we could learn the relevant features that determine the similarities and/or dissimilarities among products. By doing so, we can predict the category of new products only by looking at their features. This is what is done in the ABC analysis where a product is assigned to categories A, B or C based on its value feature.

The remaining tasks involve selecting the proper algorithms for clustering (unsupervised learn-

ing) and for supervised learning.

4.1.1 Clustering algorithm

Many clustering techniques are available in literature since they have been used in a lot of different areas, including marketing, engineering, design, and manufacturing. Jain (2010) [21] provided an overview of well-known clustering methods and underlined some emerging and useful clustering research directions. It should be mentioned that traces of clustering applications in the field of inventory management are more rarely seen but still present. Two examples can be found with Ernst et al. (1990) [9] and Srinivasan et al. (1999) [33]. Ernst proposed a mathematical formulation which requires finding a number of clusters such that a discriminant ration is maximized. The method proposed by Ernst was then used to develop an optimal inventory stocking policy in a single location distribution network. Srinivasan proposed a clustering method where products with a similar stock keeping unit (SKU) were clustered together. The similarity between two SKUs was established by a distance function that takes a number of factors into account, namely 1) the usage of the SKU with respect to the finished product, 2) the manufacturing lead time of the SKU, 3) the transit time of the SKU, and 4) the average cost. Hierarchical clustering with average linkage was then used to identify clusters of similar SKUs. Moreover, SKUs were defined for each product and for each station in the network.
Ernst's clustering approach was for a single location, while Srinivasan's approach applied to a supply chain network, but clusters of SKUs were formed at each station in the network. For these reasons, their methods are not entirely appropriate for our holistic multi-echelon approach.
The quality of our method heavily depends on how accurately product multidimensional curves are clustered. Unfortunately, clustering algorithms are problem dependent, and it is impossible to establish a universal clustering algorithms ranking. The best idea would be to try out different algorithms and then pick the method that performs the best.
Furthermore, we aim to use an existing clustering algorithm since the scope of this thesis is not on developing a new clustering method. As stated before, clustering algorithms are problem dependent since their accuracy depends on the structure of the data. In our case, we do not have any prior information on the data since we are dealing with simulated data. However, we do wish to use a clustering algorithm that does not capture any irregular patterns in the data because we want to compare two products based on the Euclidean distance between their corresponding multidimensional curves. Here, we consider a case where two product multidimensional curves belong to a certain irregular manifold but they are far from one another. For this case, we do not want to use a clustering algorithm that captures any irregular geometry or patterns in the data, since that could lead to clustering together products that have their representative multidimensional curve far apart. In other words, our focus is more on partition-based clustering algorithms [12] than on density-based algorithms [11]. To date, the most popular clustering partition method is *k-means* [21], which was introduced back in 1955 but is still widely used today. In this thesis, k-means is the method used for performing clustering analysis.

In this research, we will perform clustering on data points that have more than three dimensions. It should be mentioned that clustering algorithms usually do not work well in high dimension spaces with more than hundreds of dimensions. This is due to the fact that similarity metrics based on distances such as Euclidean distance become highly similar among a pair of multidimensional curves, and thus it becomes harder to discriminate. However, the problem can be contained by using principal component analysis (PCA) [22]. The idea of PCA is to remove dimensions that are not really informative in capturing the differences within the data. In high dimension spaces, it is possible that curves are very similar if projected along some dimensions. In these circumstances, PCA would remove these dimensions, and multidimensional curves would be projected in a lower dimension subspace where Euclidean distance becomes more effective.

4.1.2 Classification algorithm

Decision tree (DTree) [18] is selected as the supervised learning algorithm, and this choice is easily explainable for the following reasons. It is easy to understand conceptually, and it is also fast to train and test. In addition, it offers a good level of accuracy. Moreover, DTree algorithms show several advantages when compared with other well know supervised learning algorithms such as support vector machines (SVMs) and k-nearest neighbours (k-NN) [18]. Unlike SVMs, the accuracy of a DTree does not decrease when irrelevant data features are included. Furthermore, unlike k-NN, both the steps of training and predicting with a DTree are relatively fast operations.
There are three other important characteristics of a DTree model. First, it is invariant to monotonic features scaling and transformations. Second, a trained DTree model is readily human inspectable. Lastly, it is a probabilistic method in the sense that it does not merely assign labels to tested data, but it also indicates which probability these labels are assigned with.
Even though the DTree is considered one of the most popular classification methods, it does have a weakness. If a DTree is not carefully "pruned", it will deep-learn irregular patterns and data outliers, resulting in a model that overfits the training sample and can consequently show poor predictive abilities. Moreover, outliers and overfitting problems can be solved by extending DTree models to random forest models [18]. Moreover, we can control the depth and width of the DTree such that overfitting is avoided.

There are generally two types of DTrees, namely regression trees (which predict a value) and classification trees (which predict a class). In this thesis, we are interested in classification trees since we aim to predict product classes.

4.2 Multi-echelon inventory optimization

It is known that for single echelon systems, an (s, S) policy is optimal under certain conditions [1]. Optimal results have also been found in serial systems, and examples of this can be found in [6] and [7]. However, when it comes to multi-echelon networks, finding a general optimal policy is extremely complex [1]. This is because multi-echelon systems require the use of centralized policies that are extremely difficult to derive. For example, a replenishment order at one location may depend on the inventory status at all the other locations.
In his work, van Liempd [34] uses an *installation stock* policy over an *echelon stock* policy for a multi-echelon network. Using an installation stock policy means that all the decisions are made based on the inventory position of each installation in the network. The complexity of the problem then shifts to make sure that all the local decision rules are properly coordinated. This thesis uses the installation stock policy as well.

In Chapter 2, we explained the overall approach for inventory optimization in a multi-echelon network. The whole approach is based on the work by van Liempd, and it consists merely of three iterative steps—a simulation model, SSAP, and the GA.
In this thesis, the same simulation model used by van Liempd is adopted, together with the SSAP. However, the second research question of this thesis aims to improve the efficiency of van Liempd's GA, and thus we will not use the same GA used by van Liempd. In the next section, some relevant literature regarding the use of the GA in the field of supply chain is discussed.

4.2.1 Optimality

The GA is a heuristic method that has been used in the field of supply chain optimization quite extensively. Kannan et al. [23] analysed the performance of a close loop supply chain via the GA. In his work he also mentioned that one of the main advantages of the GA is that it can easily find a good solution without having to analyse all potential solutions.
Celebi [5] developed a solution on how to determine the optimal inventory levels in a spare parts distribution system using the GA. Celebi considered the GA to be one of the most favourable heuristic

method for solving MEIO problems. Among other reasons, Celebi mentioned that GAs are extremely flexible in addressing all sorts of different problems. For example, GA can deal with non-linear functions; moreover, the solution space can be either discrete or continuous, and the optimization problem can be either convex or non-convex.
In this thesis, a GA is used for finding the optimal values of a safety stock at upstream locations only. We could extend the use of the GA also to the downstream locations in the network. However, finding the optimal safety stock values downstream is too time consuming due to the customer service level constraints. For this reason, a more elegant and efficient procedure such as the SSAP is used for the safety stock levels at downstream locations.
In his thesis, van Liempd made use of the ABC–XYZ categorization [39]. This led to the creation of nine decision variables for each upstream location in the network. The total length of a GA's chromosome was then equal to nine times the number of upstream locations. For example, a chromosome for a multi-echelon network with three upstream locations would have a length equal to 27. Each element of a chromosome carried the safety factor value for a specific upstream location and for a specific product category. Once two chromosomes have been selected, a crossover was performed to create a new solution. The crossover method used by van Liempd is called the *parametrized uniform crossover* [25] in which each chromosome value can be interchanged with the corresponding value of a second chromosome. A second method for performing crossovers is called the *single point crossover* [25] in which a slice of a chromosome is interchanged with the corresponding slice of a second chromosome. In this thesis, we propose a crossover approach that resembles more the single point crossover; we believe that this method can preserve more the structure of the underlying supply chain network if the values of each chromosome are properly ordered.

5 | Methodology and approach

5.1 Network setup and policy definition

In this thesis, we consider a three-echelon network with 15 locations. This restriction is due the fact that the proposed method is tested on a real-life case of a pharmaceutical company that has a network setup consisting of three echelons and 15 locations (see Figure 5.1 for an illustration of the network). However, the methodology can be easily extended to any sort of multi-echelon network. Here, the goods move from upstream to downstream locations with the assumption that no lateral transshipment of goods take place among locations at the same echelon level. Each location has only one predecessor location and multiple successor locations. Only downstream locations are subject to direct customer demand while upstream locations face only the demand in the form of replenishment orders from downstream locations. Also, customer demand is assumed to be normally distributed. Moreover, demand in general can be backlogged, and any rationing or partial fulfilment of demand is not allowed.
We assume the network has a single supplier that always has stocks available to satisfy the replenishment orders coming from the upstream location. This means that replenishment orders coming from the central location are always fulfilled on time.
Part of this thesis is about optimally allocating safety stocks within the network's locations. However, a distinction is made between upstream and downstream locations, as shown in Figure 5.1. The safety stocks at upstream locations are set according to the following policy:

$$SS_{m,n} = k_{m,i} \cdot \sigma_{L_{m,n}+R_{m,n}} \tag{5.1}$$

where $SS_{m,n}$ is the safety stock of product n at upstream location m; $k_{m,i}$ is the safety factor of category i (at which product n belongs) at upstream location m, and $\sigma_{L_{m,n}+R_{m,n}}$ is the demand standard deviation of product n during lead time (L) plus review period (R) at upstream location m.
As mentioned in Chapter 2, the ordering policy considered in this thesis is an (R, S) policy, where R is the review period and S is the order-up-to level. The review period R was considered to be an input parameter for our simulation model, while the order-up-to level S was calculated based on equation 2.1. This means that we addressed the problem of inventory optimization only by looking at the safety stock levels and not in other ways, such as by looking into the optimal orders size.
Moreover, safety stocks at downstream locations were initially set via equation 2.3, and based on the realized service level, the downstream safety stocks were adjusted using the SSAP such that a given target service level is reached.

In this thesis, the task of optimally allocating safety stocks in the network is simplified into finding the optimal values of upstream safety factors that are defined for each class of products. The first step is to define classes of products, and it is only afterwards that we focus on finding the optimal values of safety factors via the GA.

5.2 Products classification

Before proposing a new method for product classification, it is necessary to recap the purpose of classification by answering the following two questions: why do we need to classify products, and

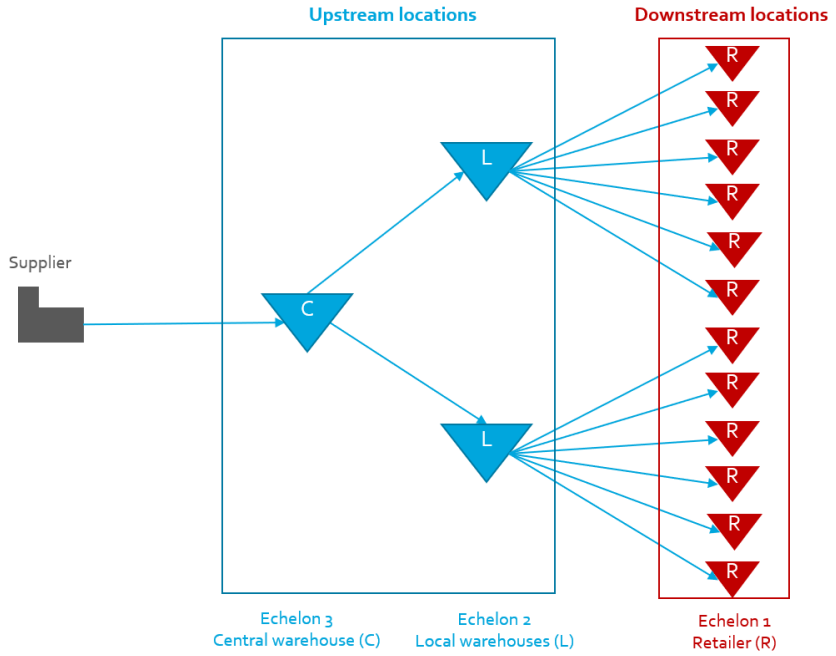


Figure 5.1: Three-echelon network with one central location, two local locations, and 12 retailers.

what defines a good classification?

To answer the first question, classification is needed for several reasons. First, it reduces the number of decision variables—such as from possible thousands to a dozen—and also, managers usually find it important to group products together when making policy and planning decisions. Secondly, classification is needed in order to use different policies among different classes. The ABC analysis represents a limited method since possible important network-dependent properties are not taken into account. Moreover, we should take into account that there are two types of penalties that increase when reducing the number of classes, namely 1) cost penalty for using polices per class and not per product and 2) the loss of discrimination. On the one hand, classification reduces the complexity of the problem, but on the other hand, the gain in simplification happens at the price of sub-optimal solutions. Thus, it is important to introduce a method that classifies products with minimal penalties. At the moment there are no custom product classification methods for multi-echelon networks, which is why this thesis intends to propose one.

To answer the second question, it is possible to intuitively say that a good classification method groups together products with similar properties. However, the definition of similar can be quite vague; particularly in the context of multi-echelon networks, it can be difficult to find similarities among products.
The planning decision variables in our case are the safety factors that are defined per upstream location and per product class. For each product, we are interested in knowing the variation of total inventory in the network with safety factors. In this sense, products that have similar variations have to be considered as similar and should be in the same class. Moreover, to be able to compare products' dynamics, we should introduce a normalization criterion since we expect that some products may have different total inventory magnitudes; this is due to the fact that some products might have much higher sales volume than others.

Once the purpose of using product classes is clear, we can move onto the methodology of the research. As mentioned, this thesis aims to find the relationship between product safety factors and their corresponding total network inventory. There are an infinite number of possible scenarios; for

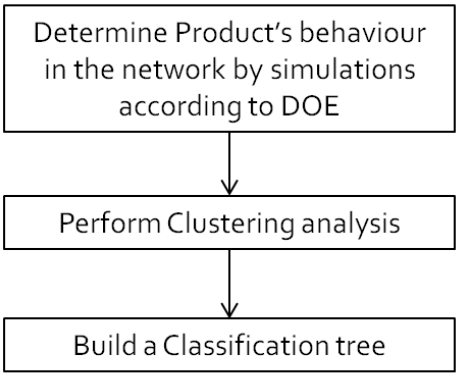
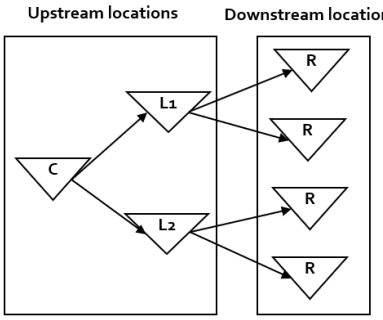


Figure 5.2: Three-step approach for product classification.

instance, one case may involve thousands of products with multiple stations in the network and multiple safety factor options for each upstream station. Given that endless scenarios are possible, there is the need for ad-hoc scenarios to be created in order to capture the overall relationships between products' safety factors and their corresponding total inventory. These scenarios are constructed according to the theory used in the design of experiment (DOE) [2], which is used to find cause-and-effect relationships. In our case, the cause and effect are represented by the products' safety factor and products' total network inventory respectively. In DOE terminology, a complete list of experiments corresponds to a *full factorial design*. However, the number of experiments of a full factorial design can be quite large and consequently time consuming. For this reason, *fractional factorial designs* [2] can be used, which is defined as "experimental designs consisting of a carefully chosen subset (fraction) of the experimental runs of a full factorial design. The subset is chosen so as to exploit the sparsity-of-effects principle to expose information about the most important features of the problem studied" [36].
The overall approach to build a classification method for multi-echelon networks can be summarized in three steps as shown in Figure 5.2. Firstly, by performing simulations according to the designed experiments (i.e., DOE), we are able to capture the behaviour of a product in the network. Each product has its own behaviour that is represented by a discretized multidimensional curve. Secondly, we can use clustering analysis to capture similarities among products—two products are similar if their corresponding discretized curves are relatively close to one another. Thirdly, a classification tree can be trained when it can know the cluster of a product and its features. Once the tree is completed, we can use it to predict the class of products only by looking at their features.

Step 1: Products' network behaviour via DOE

In the context of DOE, inputs parameters of a model are called *factors* while output parameters are called *responses*. If a model has a large number of potential factors, we could smartly select a subset of their combinations to obtain an overview of the response surface. In practice, each factor has limited *level* options because otherwise there would be an infinite number of possible values for each factor. For example, in a 2^k *factorial design* , each of the k-factors has only two level options, namely high and low. Possible factor combinations in a DOE are called *design points*, and all these together form a *design Matrix*.
In our case, the tool that allowed us to capture products' behaviour in the network consists of a simulation model and SSAP. The simulation model takes the upstream safety factors as input parameters; while, the SSAP adjusts the safety stocks at the downstream locations such that a given input target service level is reached. We can consider the combination of simulation and SSAP as a tool that takes safety factors and target service level as inputs and then gives the total network inventory for each product as output.



Design point	k_C	k_{L1}	k_{L2}	Response (tot. network inventory)
1	L	L	L	R_1
2	H	L	L	R_2
3	L	H	L	R_3
4	H	H	L	R_4
5	L	L	H	R_5
6	H	L	H	R_6
7	L	H	H	R_7
8	H	H	H	R_8

Figure 5.3: Network with three upstream locations. On the left is a three-echelon network layout, while on the right is an example of a possible experiment setup. Here, k_C stands for safety factor at the central station, k_{L1} represents the safety factor of Local Station 1 while k_{L2} is the safety factor of Local Station 2.

Product ID	*Response*1 (R_1)	*Response*2 (R_2)	*Response*3 (R_3)	*Response*4 (R_4)	*Response*5 (R_5)	*Response*6 (R_6)	*Response*7 (R_7)	*Response*8 (R_8)
Product 1								
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
Product n								

Table 5.1: Full design experiment corresponding to the network example of Figure 5.3

In agreement with DOE terminology, factors are represented by safety factors, and responses are represented by total network inventory.
Here, we assume that there is a network with three upstream locations—that is, one central location and two local locations. The example is represented on the left side of Figure 5.3. For each of the upstream location, we need to set a safety factor parameter. Since we cannot check all possible factors, we assume that for each upstream location, there are only two safety factor levels, namely low (L) and high (H). For each combination of safety factors, there is a response (i.e., total network inventory) value coming out of the simulation-SSAP model. The described scenario is summarized in Figure 5.3. The table corresponds to a full factorial design since all possible safety factor levels combinations of upstream locations are considered. The table in Figure 5.3 shows how the safety factors of the downstream locations do not play a role in designing the experiments since they do not represent decision variables for our approach. This is because we used the SSAP for downstream locations; it adjusted the safety stocks downstream to the right levels such that a certain target service level is reached.

According to the example of Figure 5.3, if we design a full design experiment for n products, we are in the situation described in Table 5.1. Each response column of Table 5.1 corresponds to a simulation-SSAP run where the safety factors inputs are set based on the full factorial design. For example, for the column *Response*2, the H–L–L combination of safety factors is used; here, the safety factors at the central location (C) are set to high (H), and the safety factors at the local locations (i.e., L_1 and L_2) are set to low (L). Once the safety factors are set, we run a simulation-SSAP, and we obtain the total network inventory of each product as a response with a guarantee target service level. This procedure is repeated until all design points are covered.

For each product, a vector of responses $[R_1, R_2, ...R_m]$ is obtained. These vectors represent the behaviour of each product in the network under different safety factor inputs. As a result, we can compare product behaviours by comparing their corresponding response vectors. However, it is possible for two products to have similar behaviours but different response magnitudes. For example,

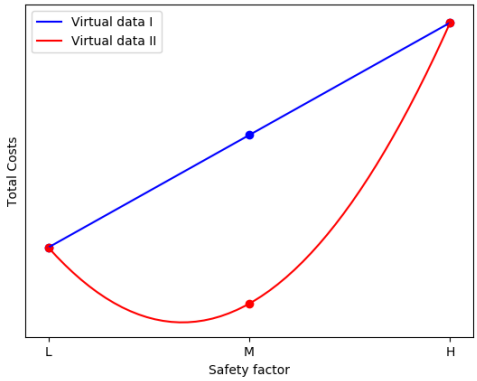


Figure 5.4: Example of using a middle level (M) to distinguish between two different dynamics within the network.

two products may react similarly to changes in safety factors, but one product may have much higher or lower inventory costs. In this case, a proper normalization of the response vectors needs to be performed in order for the products to be more comparable.
The following normalization criterion is used:

$$X_{scaled} = \frac{X - X_{min}}{X_{max} - X_{min}} \tag{5.2}$$

where X_{scaled} is the scaled response, X is the original response, X_{min} and X_{max} are the smallest and the largest original responses respectively. In this way the original responses are all converted to values ranging between 0 and 1.

According to 2^k factorial design theory, two levels are set for each factor, namely low and high. In our case, we used three levels for each safety factor: low (L), medium (M), and high (H). The additional level M increases the possible number of safety factor combinations, but it is necessary to be able to distinguish between different behaviours. An example is given in Figure 5.4 where without the use of a middle level we could not distinguish between the two different trends. Capturing the red trend is important since it suggests that there is a minimum inventory cost, which for optimization purposes is what we are aiming to identify.

To sum up, being able to construct a full factorial design requires z^k number of experiments, where z corresponds to the number of levels used for the safety factors and k is the number of upstream locations in the network.
To test our method, a three-echelon network with a total of three upstream locations is considered. This leads to a full factorial design consisting of 3^3 experiments, since we have three upstream locations and three levels of safety factors (i.e., L, M, and H).

Product ID	*Response* 1 (R_1)	*Response* 2 (R_2)	$\cdots$	*Response* m (R_m)
Product 1	R_1^1	R_2^1	$\cdots$	R_m^1
$\vdots$	$\vdots$	$\vdots$	$\cdots$	$\vdots$
Product n	R_1^n	R_2^n	$\cdots$	R_m^n

Table 5.2: Example of a full factorial design that consists of m design points and n products

Product ID	*Response* 1^* (R_1^*)	*Response* 2^* (R_2^*)	$\cdots$	*Response* m^* (R_m^*)
Product 1	R_1^{1*}	R_2^{1*}	$\cdots$	R_m^{1*}
$\vdots$	$\vdots$	$\vdots$	$\cdots$	$\vdots$
Product n	R_1^{n*}	R_2^{n*}	$\cdots$	R_m^{n*}

Table 5.3: : Example of a normalized full factorial design that consists of m design points and n products

Step 2: Clustering

A clustering analysis is used to find similarities among products. In Step 1, we explained how to capture product behaviour in multi-echelon networks by running ad-hoc simulations based on the full factorial design. Products behaviours are collected into vectors; in a discrete way, they summarize the dependency between the total network inventory and the upstream safety factors.
Here, we assume that we perform m simulation-SSAP runs for a total of n products. We also assume that the m runs correspond to a full factorial design where all possible upstream locations combinations of three levels of safety factors are exploited. At each simulation run, a response for each product is obtained. This leads to the results of Table 5.2.
However, before comparing products, all the results of the full factorial design should be normalized based on equation 5.2. After applying normalization, we end up with the normalized results of Table 5.3.

We assume here that we want to compare two products, namely product i and product j. For product i, we have the following vector of responses $V_i = [R_1^{i*}, R_2^{i*}, ..., R_m^{i*}]$ and for product j, we have the vector $V_j = [R_1^{j*}, R_2^{j*}, ..., R_m^{j*}]$. The similarity between product i and product j is measured by the euclidean distance between their corresponding vectors. The closer the distance, the more similar product i and j are. The euclidean distance between vectors V_i and V_j is expressed by D_{ij} which is computed as follows:

$$D_{ij} = \sqrt{(R_1^{i*} - R_1^{j*})^2 + (R_2^{i*} - R_2^{j*})^2 + ... + (R_m^{i*} - R_m^{j*})^2}$$

Once the similarity metric based on Euclidean distance is defined, we can compare products' behaviour using a clustering analysis. The clustering algorithm creates clusters of products such that products belonging to the same cluster will have their corresponding response vectors relatively close to each other. Products in the same cluster are then considered to be those that show similar inventory behaviours in the multi-echelon network.

Step 3: Classification tree

The clustering analysis leads to the formation of k clusters of products. Moreover, in addition to a cluster label, a product is characterized by its own features. This means that after Step 2, we are ready to train a classification tree.
The objective of using a classification tree is to use fitting rules to predict the cluster label of a product based on its features.

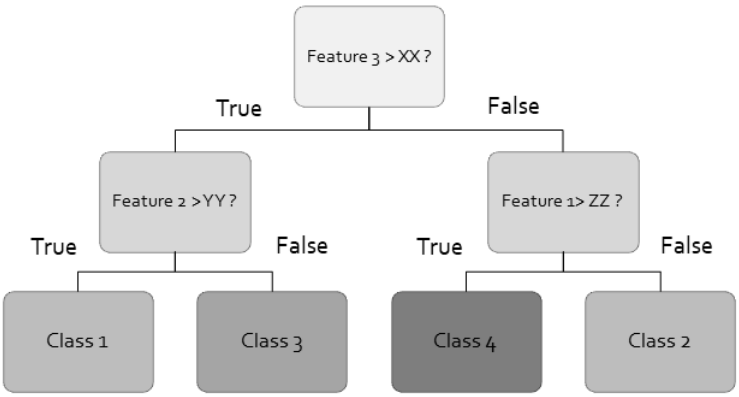


Figure 5.5: Example of classification tree.

Here, we assume that we are considering a MEIO problem with n products in total. Instead of going through Steps 1, 2, and 3 with all n products, we can use a sample of products, such as m products where $m < n$. Subsequently, we run Steps 1, 2, and 3 only using those m products. This means that we can train a classifier by using m training products, and afterwards we could use the classifier to assign a class to the remaining $n-m$ testing products only by looking at their corresponding features.

An example of a classification tree is shown in Figure 5.5. The product features that are considered when training a classification tree are described in the next session.

5.2.1 Product features

Some important features that characterize the products and the network need to be defined. The idea is to list all potential features, and we let the classifier establish which features are the most discriminative in determining the right class for a product.
The average demand of the products can be an example of a product feature, while the number of locations at each echelon level where products are stocked can be an example of feature that is both product and network dependent.
In this thesis, we distinguish between five types of features. This restriction is also partially due to the amount of real data information available. However, the list of features could be easily extended since DTrees are invariant with respect to the number of features used [18]. The type of features are defined in Table 5.4.

Feature type I

The notation for this feature type is: $\#Outlet_k$. Considering a product j stocked at location k, when looking downstream in the network we can count the number of outgoing outlets seeing by product j at location k. The number of outgoing outlets corresponds to the number of downstream locations connected to location k where product j is also stocked.

	Feature type I	Feature type II	Feature type III	Feature type IV	Feature type V
Description	Number of outgoing outlets seeing by product j at location k	Lead time between interconnected locations A and B	Fractional mean demand of product j at location k	Coefficient of variation of product j at location k	Pooling measurement between a centralize location k and its connected downstream locations
Notation	$\#Outlet_k$	Lt_{AB}	FMD_k	CV_k	$Pool_k$

Table 5.4: Product–network features definition

Feature type II

This feature type refers to the lead time between the interconnected locations in a multi-echelon network. The notation L_{AB} represents the lead time between locations A and B. We assume a fix lead time where lead time uncertainties are not taken into account.

Feature type III

The feature type III refers to the fraction mean demand of product j at location k where there are a total of m products stocked. This feature is defined as follow:

$$FMD_k^j = \frac{\mu_k^j}{\sum_{i=1}^{m} \mu_k^i}$$

where μ_k^i is the mean demand of product i at location k. This feature type reflects the dominance of a product, in terms of volume, at a given location k. Moreover, this feature type resembles the ABC categorization where products are split into three categories based on their values. In our case, we are using volume instead of value to compare product importance due to lack of price information in our real data. However, this feature type can be easily changed if price information is available.

Feature type IV

The coefficient of variation of product j at location k is defined as follows:

$$CV_k^j = \frac{\sigma_k^j}{\mu_k^j}$$

where μ_k^j and σ_k^j are respectively the mean and standard deviation of the demand of product j at location k. This feature is an indicator of product demand volatility. The smaller the CV, the more stable the demand pattern of a product. This feature is the same feature that is used in the XYZ categorization.

Feature type V

The feature type V, in combination with the feature type IV, takes into account the *risk pooling* effect. "Risk Pooling suggests that demand variability is reduced if one aggregates demand across locations" [31]. Demand pooling from a central location is beneficial if the location itself is connected to multiple downstream locations. Moreover, risk pooling is most effective when downstream demands have a high coefficient of variation and comparable standard deviations (and/or comparable means). In this research, we assume that customer demands are independent and normally distributed. In this way, we can also assume that the aggregated demands across locations are normally distributed. For example, the aggregated demand distribution of a given product across two locations n and m (at the same echelon level) is given by $N(\mu_n + \mu_m, \sqrt{\sigma_n^2 + \sigma_m^2})$.

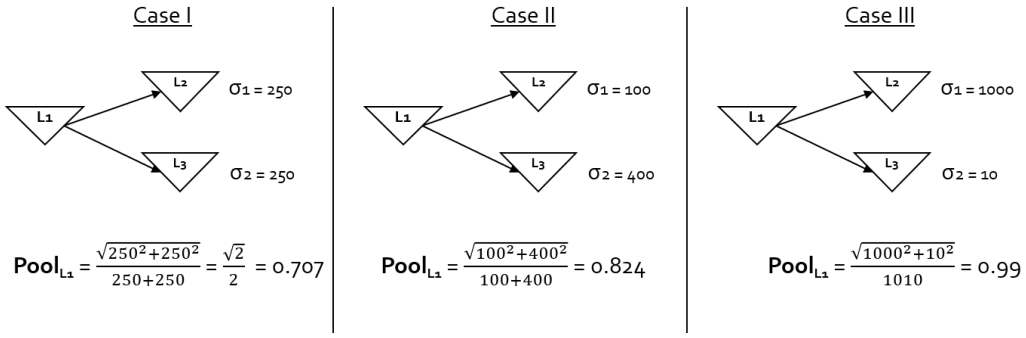


Figure 5.6: Three examples used to interpret different outcomes of the pooling feature.

Product ID	$Outlet_C$	Lt_{C-R_1}	Lt_{C-R_2}	Lt_{C-R_3}	FMD_C	FMD_{R1}	FMD_{R2}	FMD_{R3}	CV_C	CV_{R1}	CV_{R2}	CV_{R3}	$Pool_C$
Product i	2	Lt 1	Lt 2	NaN	$\frac{\mu_i^C}{\sum_{i=1}^{N}\mu_i^C}$	$\frac{\mu_i^{R1}}{\sum_{i=1}^{N}\mu_i^{R1}}$	$\frac{\mu_i^{R2}}{\sum_{i=1}^{N}\mu_i^{R2}}$	0	$\frac{\sigma_i^C}{\mu_i^C}$	$\frac{\sigma_i^{R1}}{\mu_i^{R1}}$	$\frac{\sigma_i^{R2}}{\mu_i^{R2}}$	NaN	$\frac{\sigma_i^C}{\sigma_i^{R1}+\sigma_i^{R2}}$
Product j	1	NaN	NaN	Lt 3	$\frac{\mu_j^C}{\sum_{i=1}^{N}\mu_i^C}$	0	0	$\frac{\mu_j^{R3}}{\sum_{i=1}^{N}\mu_i^{R3}}$	$\frac{\sigma_j^C}{\mu_j^C}$	NaN	NaN	$\frac{\sigma_j^C}{\mu_j^C}$	$\frac{\sigma_j^C}{\sigma_j^{R3}}$

Table 5.5: Features calculation based on the example of Figure 5.7

Let's assume that a product j is stocked at a central location k which supplies n downstream locations. Then, we define the feature as follows:

$$Pool_k^j = \frac{\sigma_j^k}{\sum_{i=1}^{n}\sigma_j^i} = \frac{\sqrt{\sum_{i=1}^{n}(\sigma_j^i)^2}}{\sum_{i=1}^{n}\sigma_j^i}$$

where σ_j^i is the standard deviation of demand of product j at location i. Moreover,the $Pool_k$ feature is bounded as follows:

$$\frac{\sqrt{n}}{n} \leq Pool_k \leq 1$$

where n is the number of downstream locations connected to the upstream location k. The $Pool_k$ assumes the lower bound value when the standard deviation at all the n locations is exactly the same. The $Pool_k$ assumes the upper bound value when the upstream location k supplies only one downstream location. A high value of $Pool_k^j$ suggests a high demand variability of product j at an echelon level that is lower than k. However, a value closer to 1 suggests that there is one dominant product demand downstream. Three examples of calculating the pooling feature can be seen in Figure 5.6. Case I shows the extreme case in which the standard deviations downstream are exactly the same, and the pooling feature takes the lower bound value. Case III is another extreme case where there is a dominant standard deviation value downstream, and the pooling feature takes a value closer to 1. Case II is the hybrid example between Cases I and III. Case III is not suited for demand pooling, while Cases I and II may be suited for demand pooling if they are combined with a relatively high value of the coefficient of variations.

For the sake of clarity, we can apply the defined five features type to the network example considered in Figure 5.7 for which we construct Table 5.5.

5.2.2 Clustering algorithm using k-means

K-means is the method used for the clustering analysis. K-means aims to partition n observations into k clusters in which each observation belongs to the cluster with the nearest centroid. However, k-means has three major drawbacks which can be summarized below.

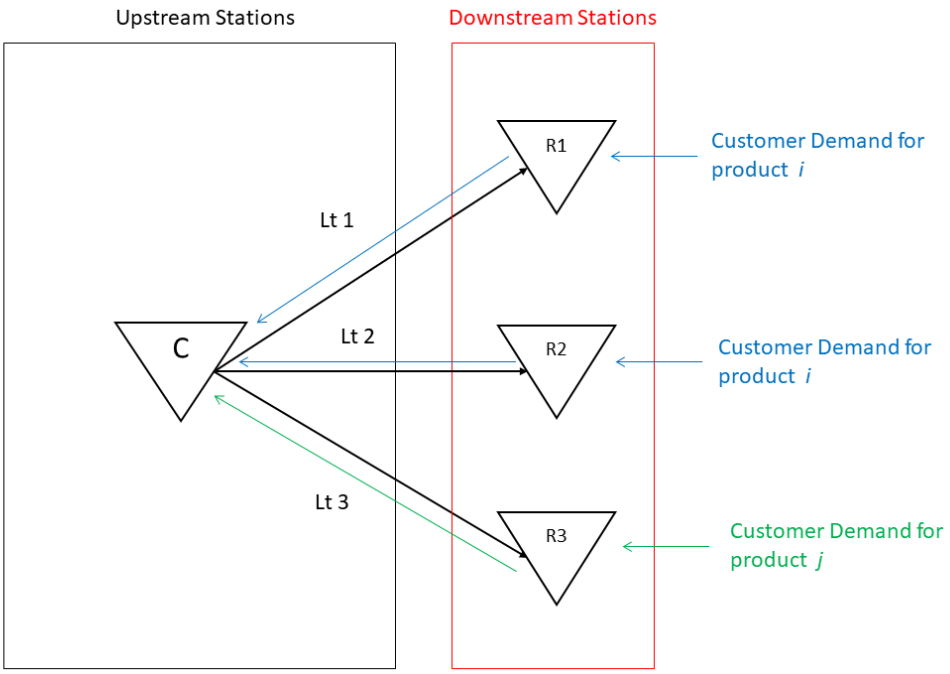


Figure 5.7: Network example. The blue and green lines track product i and j respectively from downstream to upstream locations. The lead times among the locations are represented by the notation Lt.

1. The quality of k-means depends on the selection of initial centroids.
2. K-means effectiveness depends on the data geometry.
3. The number of clusters k is an input for k-means.

To overcome the first issue, we run k-means multiple times with different random initial centroids, and the best solution is then kept. The best solution is the one with the highest similarity in the cluster and the lowest similarity between clusters.
For the second issue, we know that k-means works well for flat geometry data. Indeed, k-means is not able to capture non-globular data shapes as shown in the example of Figure 5.8. However, in the previous section, we showed that each product is characterized by a response vector; this vector captures the changes in inventory costs due to changes in upstream safety factors. Moreover, each vector of length l is represented by a point in an l-dimensional space. We aim to cluster the points together only based on their Euclidean distance without worrying about the presence of non-globular data shapes. This is because there may be points that belong to the same non-globular shape, and these points may still be far apart from each other. Here, we can conclude that data geometry does not represent an issue for k-means in our case.
Finally, to address the third point, three commonly used methods could be used to determine the right number of clusters k for k-means, namely sum square error, the Calinski–Harabasz method [4], and silhouette width [27]. The main idea behind these methods is to offer trade-off curves between complexity (i.e., the number of clusters k) and the accuracy of the clustering analysis. The higher the number of clusters, the more accurate the analysis. However, we have to consider whether the gain in accuracy is worth the gain in complexity. These methods offer trade-off curves that point out which value of k should be selected. A common method used in selecting the number of clusters k based on a trade-off curve is the *elbow rule* [37]. The position of the "elbow" in the plot indicates the number of clusters to use since further increases of k brings a marginal accuracy improvement.

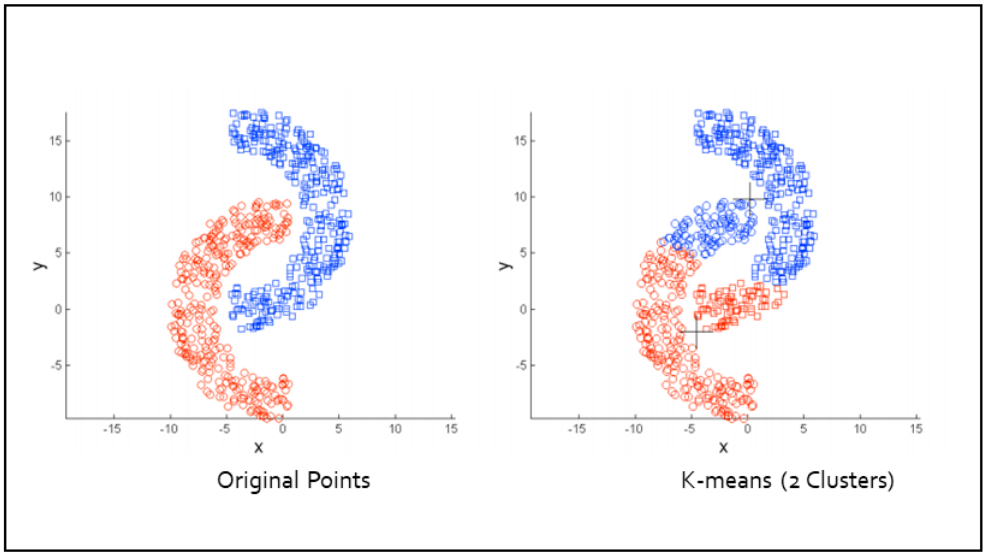


Figure 5.8: Example of bad clustering using k-means for non-globular shapes.

Moreover, it is important to remember that customer demand is assumed to be normally distributed and therefore not deterministic. Customer demand uncertainties ensure that different simulation runs would generate slightly different outputs under the same input conditions. This means that a product vector which lies relatively close to two cluster centroids can be quite sensitive to demand volatility. By way of example, suppose that we perform a full factorial design using a specific seed for the customer demand distribution. Then, the clustering analysis assigns product j to Cluster 1. Moreover, we assume that product j lies very close to the neighbourhood of Cluster 2, as shown in Figure 5.9. We assume also that we redo the clustering analysis based on a new full factorial design that uses a different seed for the customer demand distribution. The result of the new analysis can be that product j is now assigned to Cluster 2 and not to Cluster 1. The higher the number of clusters k, the higher the probability is for assigning the wrong cluster label to a product due to customer demand uncertainties.
However, we could select the right number of clusters k to contain or minimize the described issue. Our approach consists of performing an m number of independent full DOEs where we use m different seeds for the customer demand distribution. For each of the m DOEs, we perform a k-means analysis for a range of k values. Afterwards, we establish the "true" cluster label of a product based on the majority of occurrences out of the m clustering analysis. For example, $k = k^*$, product j would be assigned to cluster $k^* - z$ (with $0 \leq z \leq k^* - 1$) for the majority of the m cases, and the true cluster label of product j would then be $k^* - z$. By using this approach, we can construct trade-off curves where we measure the misclassification error by comparing the true cluster label versus the individual cluster labels of the m independent instances for a range of k values. An example of such trade-off curves can be seen in Figure 5.10. After the trade-off curve is constructed, we make use of the elbow rule to choose the number of clusters k for k-means. By applying the elbow rule to the example of Figure 5.10, we would select $k = 7$.

5.2.3 Classification algorithm using decision trees

A DTree is a method that allows us to predict a product class based on its features, and tree models where the target variables are represented by class variables are called classification trees. In our

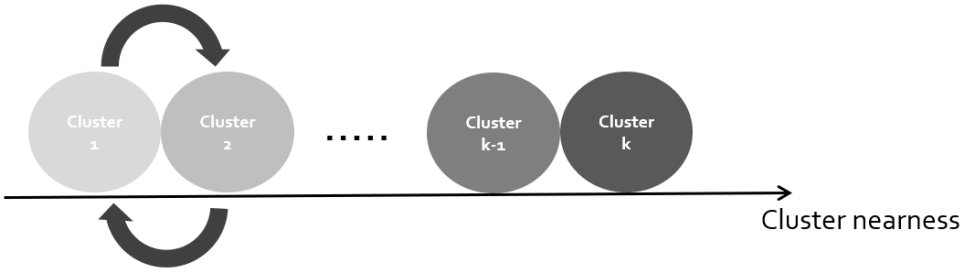


Figure 5.9: Example of a potential cluster label swap for products that are on the boarder of two clusters.

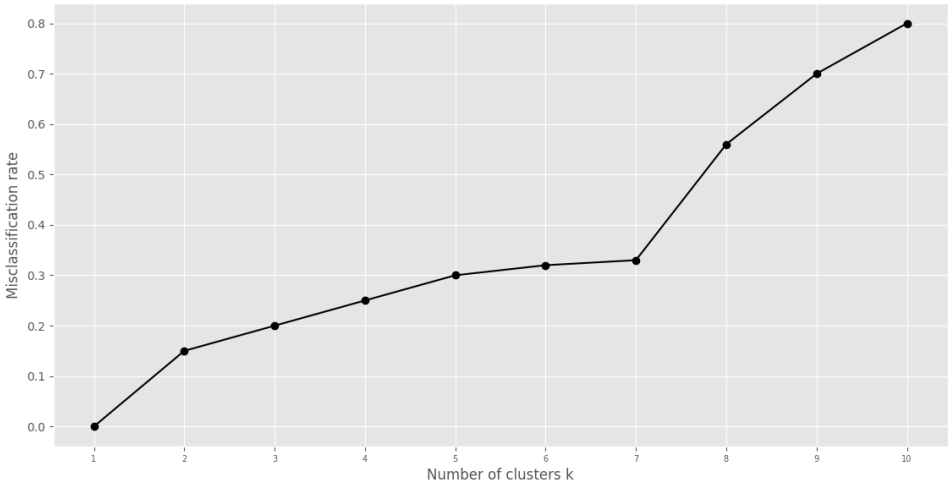


Figure 5.10: Example of a trade-off curve where the number of clusters k is plotted against the misclassification error. The elbow rule would suggest selecting the number of clusters k equal to 7.

case, the class variables are represented by the cluster labels obtained via k-means.

A tree consists of *nodes* and *leaves*. The first node is called the root while the final nodes are called leaves. In the DTree, we move from the upper to the lower part of a tree by applying binary splits. A binary split consists of splitting a node into two nodes, going from root nodes to leaf nodes. In this research, we consider only numerical product features; this means that a binary split is applied based on a feature threshold value. More precisely, we want to know whether a certain feature at a certain node is higher or lower than the corresponding threshold value at that node.
In general, a DTree goes from impure to pure when moving from root nodes to leaf nodes. Maximum impurity means an equal mixing of the classes, while minimum impurity means that each leaf corresponds to exactly one class. There are several measures for impurity, and among them two common methods are the Gini index and Entropy [19]. With these methods, we can determine which feature is used for a split when taken together with its associated threshold value. The higher the predicting power of a feature, the higher the position of the feature in the tree.

Building a DTree usually consists of two phases, namely building the tree and testing its performances. In addition, a data set is usually split into training and testing data; the training data are used to train a DTree, while the testing data are used to test the predictive power of the tree. The performance of a DTree model can be evaluated by considering the misclassification error on the training and testing data set. Simply adding nodes until all cases belong to the right class would lead to optimal results for the training set; however, the results will not be optimal for the test data since we are most likely overfitting the data. Big and complex trees are usually an indicator of overfitting the data. On the other hand, a tree with a few nodes would give a simple model that is useful in practice, but it may oversimplify the situation, thereby leading to many misclassified cases. Certain tree parameters such as *max depth*, *max leaf nodes* and *min samples split* can be fine-tuned such that a satisfactory accuracy on the testing data is reached and data overfitting is avoided. Figure 5.5 shows an example of a small classification tree where only three features are used to make predictions on four classes. In this research, we make use of the DTree algorithm available in the `Scikit-learn` Python package [38]. For the DTree, we optimize the following parameters such that the highest accuracy on the test set is reached:

1. *criterion*, which is either Gini or entropy;
2. *max_depth*, which determines the max depth of the tree;
3. *max_leaf_nodes*, which determines the max number of leaf nodes allowed such that there is control over the width of the tree;
4. *min_samples_split*, which determines the minimum number of samples in a node to be able to perform a binary split.

All the mentioned parameters are input parameters for the DTree algorithm available in the `Scikit-learn` Python package. In this research, we construct a grid search for which all the parameter combinations are tested. The combination with the highest accuracy on the test set is used to build the final classification tree.

5.3 Optimal safety stock levels

The simulation-SSAP model can be considered as an evaluation function used to measure the goodness of a solution in the form of safety stocks levels. However, the simulation-SSAP model is not sufficient to find the optimal safety stocks levels to the underlying multi-echelon network. In order to find the optimal safety stocks levels, we make use of the GA such that minimal inventory costs are reached under service level constraints. Upstream safety stocks are calculated according to the policy described by equation 5.1. Upstream safety stocks are optimally allocated by setting the corresponding safety factors to optimal values. However, safety factors are defined per product category and per upstream location.

5.3.1 Genetic algorithm

The GA is a dated heuristic method that was introduced by Holland [21] in 1975. The GA is inspired by the process of natural selection that drives biological evolution. The general GA principle is that by mixing good solutions, potential better solutions can be found.
First of all, some GA terminology needs to be introduced. A *chromosome* represents a solution to the problem that needs to be optimized. A chromosome consists of several *genes*, and each gene represents a decision variable of the problem. The length of a chromosome depends on the total number of genes. In our case, a gene corresponds to the safety factor value of a certain product category at a certain upstream location. This means that a chromosome for a multi-echelon problem with m upstream locations and k product categories have a length equal to $m \cdot k$. A chromosome represents a solution to the problem since it contains all the upstream safety factors for each product category.
In the GA, an *initial population* needs to be created. An initial population consists of creating different initial solutions to the problem in the form of chromosomes. For example, an initial population of length l corresponds to the creation of l different chromosomes. Moreover, in order to compare chromosomes and to establish which chromosomes perform better than others, we define a fitness value for each chromosome. For example, the total inventory costs associated to each chromosome could be used as a fitness metric to establish the quality of the solution itself.

Inventory costs can be divided into holding costs, ordering costs, and penalty costs [26]. In this research, we only consider holding costs. Moreover, since price information is not available for our real-life data, we take the total units of on-hand inventory as an approximation of the total holding costs. This means that the fitness of a chromosome is measured by the total units of on-hand inventory corresponding to its solution.

Genetic algorithm operators

The mains steps in the GA are called *operators* [25]. These are listed as follows::

- *Selection*: Selecting a chromosome out of a population is based on a probabilistic criterion. The fitter the chromosomes, the higher the probability of being selected. The selected chromosomes are called *parents*.
- *Crossover*: Genes belonging to two chromosomes are interchanged based on a probabilistic criterion such that two new chromosomes are formed. The new chromosomes are called *children*.
- *Mutation*: Chromosome values are changed based on a probabilistic criterion. Each gene of a chromosome could encounter a mutation. The probability of having a mutation is generally low [25] since a high value would destroy all the information encoded in a chromosome. For instance, a mutation rate equal to 1 corresponds to a random search since all genes are always mutated. Based on the work by van Liempd, when a mutation takes place, each gene of a chromosome can be adjusted by a randomly generated number in the interval $(-0.03, 0.03)$. For example, a safety factor value k can be mutated into a value belonging to the following interval $(k - 0.03, k + 0.03)$.

Once an initial population of chromosomes is formed, the *roulette wheel* [25] method is used to select two parents chromosomes. The principle of the roulette wheel is that the fitter the chromosome, the higher the probability of it being selected. Once two chromosomes are selected, their genes are interchanged via the crossover operator such that two new children chromosomes are created. After the crossover, the newly created chromosomes undergo some genes mutation based on a probabilistic mutation rate. The whole procedure is then repeated until a new population of chromosomes is created.

In this research, we propose a GA that mainly has two differences compared to the GA proposed by van Liempd [34]. The two differences concern the creation of the initial population and the crossover operator.

Initial population

Initial population consists of creating a certain number of initial chromosomes. We assume that the safety factors for each product class would range between the value of 0 and an upper value M. This means that each chromosome value also ranges between values 0 and M.
Van Liempd generated an initial population by drawing a random number from a uniform distribution $U(0, M)$. Assuming that each chromosome has length l, van Liempd was able to create an initial chromosome by randomly drawing l times out of a $U(0, M)$ distribution. The procedure was repeated according to the desired size of the initial population.
However, in this thesis, we can already be more specific about the value of certain genes in the early stage of creating an initial population. The reason is that before performing optimization via the GA, many simulations were already performed to build a classification tree. This means that some information about the structure of the network is available. The goal is then to translate this information into practical steps. In this thesis, a gene of an initial chromosome is created by drawing a random number from a uniform distribution $U(x, y)$ where $x \geq 0$ and $y \leq M$ (with $x < y$). In the clustering analysis, we performed a full factorial design based on three levels of safety factors for all upstream locations. This means that information about some good and bad performing combinations of safety factors is already available after clustering. By focusing on the good performing combinations, we are able to identify optimal areas of safety factors. By properly selecting x and y values, we can generate initial chromosomes by drawing from a uniform distribution $U(x, y)$ such that we can explore the optimal areas of safety factors more efficiently. Assuming a normally distributed demand, we have a max safety factor equal to 2 for an internal service level equal to 98%. This means that initial chromosomes are created by drawing from a uniform distribution $U(x, y)$ with $x \geq 0$ and $0 < y \leq 2$ (with $x < y$). The 98% service level is taken as a reference since it will be the service level target for all products at the retailers.

Crossovers

In this research, we used a class crossover approach that resembles the *single point crossover* [25]. This approach is slightly different from the *parameterized uniform crossover* [25] used by van Liempd [34].
All the information encoded in a chromosome can be structured in different ways. Here, we consider the multi-echelon problem where there are m upstream locations and k product categories per location. With the notation $k_{i,j}^{(p)}$, we denote the safety factor of product category j (with $j = 1, 2, .., k$) at upstream location i (with $i = 1, 2, ..m$) corresponding to chromosome p.
A chromosome can be structured in different ways. An example can be seen in Figure 5.11 where two chromosomes carry the exact same information but they are simply structured differently. The chromosome in Figure 5.11b propagates horizontally, while the chromosome in Figure 5.11a propagates both vertically and horizontally. Moreover, it should be noted that the chromosome of Figure 5.11a has its number of columns equal to the number of product categories k and the number of rows equal to the number of upstream locations m. In his work, van Liempd introduced a chromosome structure corresponding to Figure 5.11b. In this thesis however, we used a chromosome structure as described in Figure 5.11a.

Working with different chromosome structures can facilitate different types of crossovers. For example, the chromosome structure of Figure 5.11a can facilitate the class crossover as described in Figure 5.12. After two parents are selected, their genes are recombined such that two new children can be created. In the example of Figure 5.12, the safety factors of product category 1 are interchanged between Parent 1 and Parent 2. The same happens for the safety factors of product category k.
Moreover, if we would use the chromosome structure of Figure 5.11b, the parameterized uniform crossover would correspond to the example of Figure 5.13. The crossover approach summarized in Figure 5.13 is the crossover method used by van Liempd.

In this research, we used the *class crossover* that is schematically represented in Figure 5.12. We call the method a class crossover since all genes belonging to a product class are interchanged. The reason for this choice is that this kind of crossover preserves some structure of the underlying supply

$k_{1,1}^{(p)}$	$k_{1,2}^{(p)}$	$\cdots$	$k_{1,k}^{(p)}$
$k_{2,1}^{(p)}$	$k_{2,2}^{(p)}$	$\cdots$	$k_{2,k}^{(p)}$
$\vdots$	$\vdots$	$\ddots$	$\vdots$
$k_{m,1}^{(p)}$	$k_{m,2}^{(p)}$	$\cdots$	$k_{m,k}^{(p)}$

(a)

$k_{1,1}^{(p)}$	$k_{1,2}^{(p)}$	$\cdots$	$k_{1,k}^{(p)}$	$k_{2,1}^{(p)}$	$k_{2,2}^{(p)}$	$\cdots$	$k_{2,k}^{(p)}$	$\cdots$	$k_{m,1}^{(p)}$	$k_{m,2}^{(p)}$	$\cdots$	$k_{m,k}^{(p)}$

(b)

Figure 5.11: Two examples of chromosomes structure both encoding the same information.

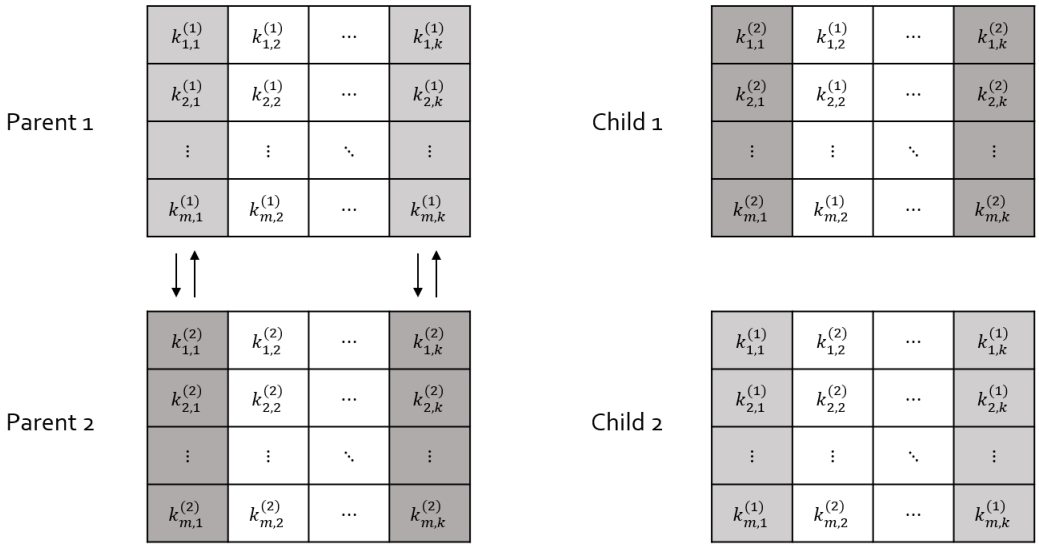


Figure 5.12: Example of class crossover.

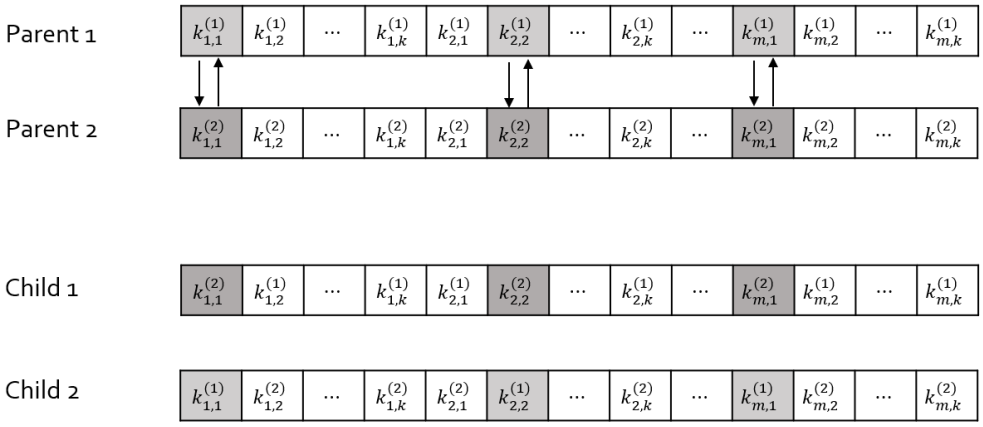


Figure 5.13: Example of parameterized uniform crossover.

chain network. For example, we assume we have a multi-echelon network consisting of three upstream locations, namely one central location and two local locations. We also assume that a chromosome carries a centralized setting of the network for a given product category k. The centralized setting corresponds to setting the safety factor to a high value (H) at the central location and low values (L) at the local locations for products belonging to category k. This corresponds to the H-L-L setting of the network. Moreover, we assume that a second chromosome carries a decentralized setting for product category k. The setting corresponds to a low value of safety factor (L) at the central locations and higher values of safety factors (H) at the local locations. This corresponds to the setting L-H-H. By using a class crossover, we are able to swap the centralized configuration H-L-L with the decentralized configuration L-H-H for products belonging to category k. Moreover, the probability of having a class crossover between two parents chromosomes is typically set between 0.5 and 0.8 [25]. In this thesis, we used a probability of 0.5; this means that there is a 50% chance that two selected chromosomes interchange their class genes. The sequence of operations performed in the GA are described in Figure 5.14.

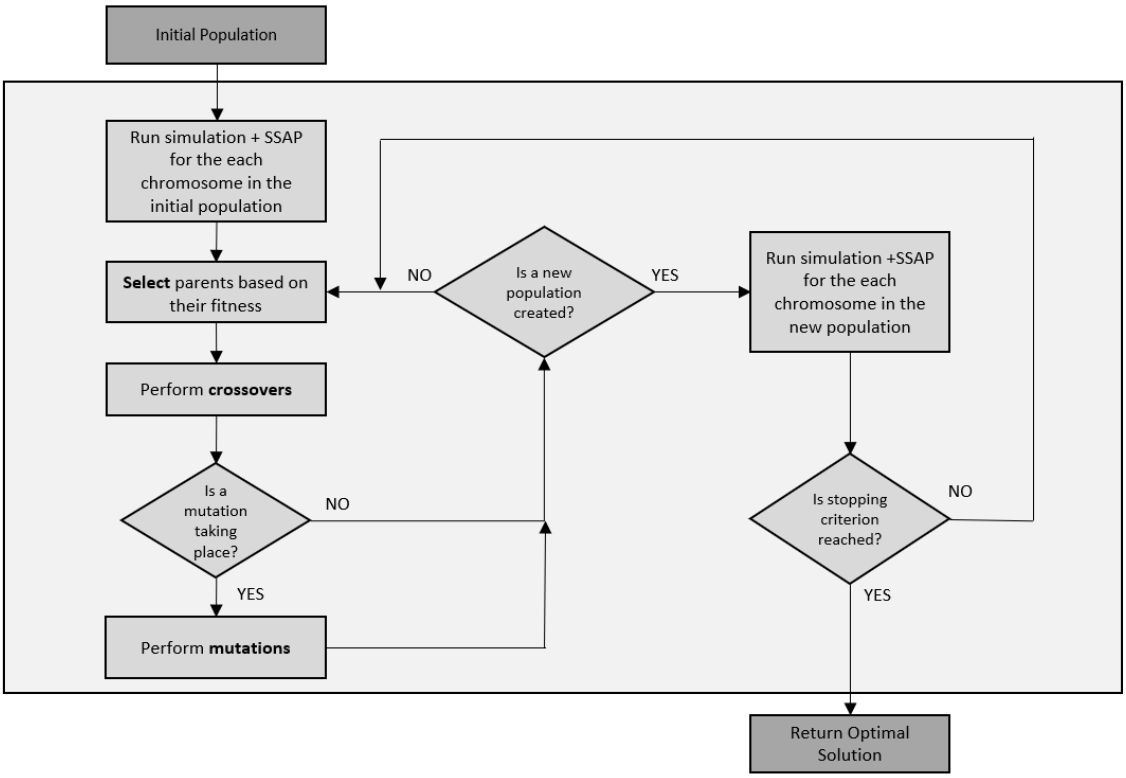


Figure 5.14: Overall steps of the genetic algorithm.

6 | Results

In this section, the proposed method is applied on a three-stage distribution network of a pharmaceutical company, and actual customer data is considered. The first step is to generate a product classification method based on machine learning techniques. Then, based on the new product classification, we use the GA to find the optimal allocation of safety stocks in the network. The optimal results are then compared to the optimal results based on the ABC–XYZ categorization.

In the research, we received one year of historical customer demand from the pharmaceutical company, and demand was provided in monthly buckets. Based on the historical data, we calculated the mean demand and standard deviation of demand. We then used the statistics to fit a normal distribution to simulate customer demand behaviour.

In addition, we used the programming language Python 3 to build the proposed method, and `Scikit-learn` is the python package used for clustering and classification.

6.1 Case introduction: Three-stage distribution network

The three-stage distribution network is shown in Figure 6.1. The network consists of one central distributor, two local distributors and 12 retailers.
We make the assumptions that the central distributor has a unique supplier with an infinite amount of stocks available and that the supplier always delivers on time. Moreover, a second assumption is that customer's demand takes place only at retailers, and each retailer is supplied by one distributor only. The replenishment lead time between locations is assumed to be deterministic. Table 6.1 gives a summary of the network's replenishment lead times. The total number of products considered for the distribution network is equal to 852.

6.2 Simulation model settings

The simulation model is a not terminating model, which means that a clear stop criterion needs to be given as input to be able to stop the simulation. The stopping criterion is given by the simulation run length. The longer the run length, the more stable the results. The idea is to use a run length such that less frequent events are also represented in a reasonable quantity. However, we want to analyse the network when it is in its steady state such that final results are, for example, independent from initial stock levels. In order to analyse the system in its steady state, we made use of the warm-up length, which is also an input for the simulation model. We set the simulation run length and warm-up length equal to 1000 and 100 periods respectively.
The virtual simulation period corresponds to one real month since customer demand is provided on monthly buckets. Moreover, replenishment lead times are provided in days, meaning that a criterion to convert days to months should be defined. We assume that one month consists of 22 working days, and we use the formula $Months = \lceil \frac{Days}{22} \rceil$ to convert days to months. For example, 23 days corresponds to 2 months, while 5 days corresponds to 1 month. This conversion criterion is quite conservative since the real lead time is always shorter than the corresponding approximated monthly

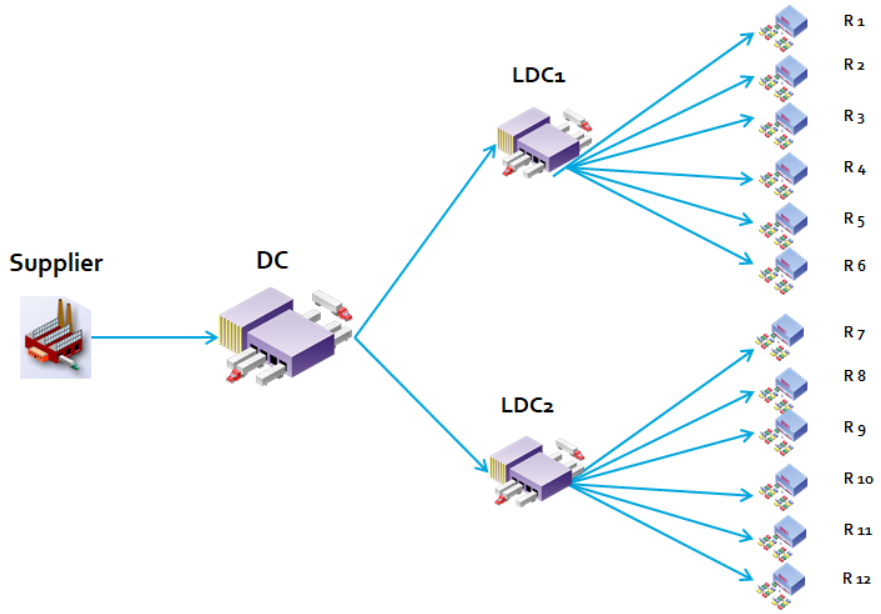


Figure 6.1: Three-echelon network with one central distributor, two local distributors and 12 retailers.

Downstream location	Upstream location	Real lead time (in days)	Simulation lead time (in Months)
Retailer 1	Local DC 1	8	1
Retailer 2	Local DC 1	3	1
Retailer 3	Local DC 1	3	1
Retailer 4	Local DC 1	3	1
Retailer 5	Local DC 1	3	1
Retailer 6	Local DC 1	9	1
Retailer 7	Local DC 2	6	1
Retailer 8	Local DC 2	9	1
Retailer 9	Local DC 2	10	1
Retailer 10	Local DC 2	4	1
Retailer 11	Local DC 2	9	1
Retailer 12	Local DC 2	12	1
Local DC 1	Central DC	6	1
Local DC 2	Central DC	30	2
Central DC	Supplier	10	1

Table 6.1: Replenishment lead times for the three-stage distribution network

lead time. The real and simulated lead times are shown in Table 6.1.

As described in previous sections, the simulation model is run together with the SSAP. For the SSAP, the target service level is an input parameter. In this thesis, the cycle service level is used as metric for the customer service level. We set the target service level to 98% for all products at the retailers. However, the whole methodology can be easily extended for diversified service levels.

Finally, in Chapter 2 we mentioned that the ordering policy used in the simulation model is an (R, S) policy, where R is the review period and S is the order-up-to level. In Chapter 2 we described how to calculate the order-up-to level S, while the review period R is set to one month in the simulation model for all product-location combinations. However, a diversified review period policy for different product-location combinations is fully supported in our methodology.

6.3 Product classification

The approach used for building a product classification consisted of three steps. Firstly, we used DOE to run a fixed amount of simulations that would give us a good overview of the products behaviour in the network. Secondly, we implemented an unsupervised learning technique via clustering k-means to find similarities among products based on the results of DOE. Thirdly, we made use of the clustering results to build a classification model using DTrees. The classification model predicts the cluster label of a product based on its features.

6.3.1 Full factorial experiment

In order to understand how products behave within the network a full factorial design of experiments (DOE) is used. The DOE consisted of running a finite number of simulations, each with a different setting; this was done in order to capture different products behaviour within the network. For each distribution center, three levels of safety factors were used. The three levels of safety factors are 0, 1, and 2; these correspond to the notation of L (low), M (medium) and H (high) respectively. We set the max safety factor to 2 based on the equation 2.3 which assigns the safety factor equal to 2 for a 98% service level. The low (L) and medium (M) safety factor values follow from an homogeneous split considering a max value equal to 2. Since there are three upstream locations corresponding to the distribution centers, we have a total of $3^3 = 27$ design points to simulate. We use a three-index notation X-X-X where the first index refers to the safety factor at the Central Distributor (DC), the second index refers to the safety factor at Local Distributor 1 (L_{DC_1}), and the last index refers to the safety factor at the Local Distributor 2 (L_{DC_2}). For example, the simulation corresponding to the design point L-M-H is the simulation where the safety factors are set to 0 at the central distributor for all the products; the safety factors are set to 1 for all the products at distributor L_{DC_1}, and the safety factors are set to 2 for all products at the distributor L_{DC_2}. For each product and for each design point, we had a response in terms of total units of inventory on-hand. Table 6.2 shows the results of the 27 design points for two exemplary products, which are called product i and product j for confidentiality reasons.
It has to be mentioned that the SSAP was applied to all simulated design points. In this way, we are able to compare design points that have the same realized service level of 98% at the retailers. The ordering of the design points in Table 6.2 is not relevant as long as the same design points are compared for the pair of products.

As we can see in Figure 6.2 and Figure 6.3, the two products i and j have different behaviours. First of all, product j barely reacts to safety factors changes. One reason could be that to reach a 98% service level, product j stocks are pushed at the retailers via the SSAP. In this way the total on-hand inventory is less sensitive to safety factors changes upstream. At a closer look, we note that product i reaches lower on-hand inventory for a configuration of the network where the safety factors at the central distributor are set to low (L). We will use this information later on to make better

Design point	Network config	Total on-hand inventory prod i	Total on-hand inventory prod j	Design point	Network config	Total on-hand inventory prod i	Total on-hand inventory prod j
1	L-L-L	2070	279	15	H-M-M	3027	275
2	M-L-L	2367	276	16	L-H-M	2629	271
3	H-L-L	2672	279	17	M-H-M	2934	269
4	L-M-L	2248	276	18	H-H-M	3239	274
5	M-M-L	2549	273	19	L-L-H	2459	276
6	H-M-L	2854	276	20	M-L-H	2759	275
7	L-H-L	2458	273	21	H-L-H	3064	282
8	M-H-L	2761	271	22	L-M-H	2637	273
9	H-H-L	3066	275	23	M-M-H	2941	272
10	L-L-M	2241	276	24	H-M-H	3246	278
11	M-L-M	2541	274	25	L-H-H	2847	270
12	H-L-M	2845	278	26	M-H-H	3152	271
13	L-M-M	2419	273	27	H-H-H	3457	277
14	M-M-M	2722	271				

Table 6.2: Full factorial experiment for products i and j with the corresponding total on-hand inventory at 98% service level

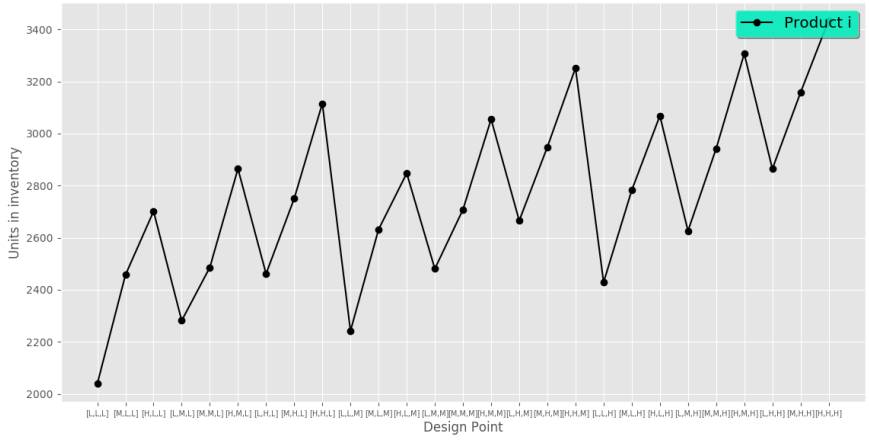


Figure 6.2: Product i behaviour based on 27 simulations according to the full factorial design.

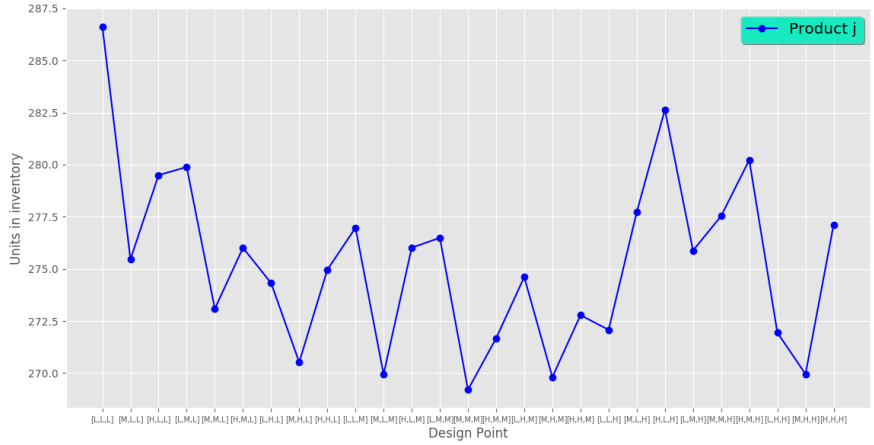


Figure 6.3: Product j behaviour based on 27 simulations according to the full factorial design.

decisions when setting the parameters for the GA.
The on-hand inventory volumes of products i and j have different magnitudes, and this difference makes them hard to compare. For this reason, the full factorial design results of products i and j are normalized according to equation 5.2. The normalization makes products i and j comparable for clustering purposes as shown in Figure 6.4.
Figure 6.5 shows the normalized results for two other products, which we refer to as products n and m. As we can see, products i and j show different behaviours, while products n and m shows more similarities in their behaviours.
For the sake of clarity, results in this section are shown only for four products. However, the illustrated procedure is applied to all the 852 products.

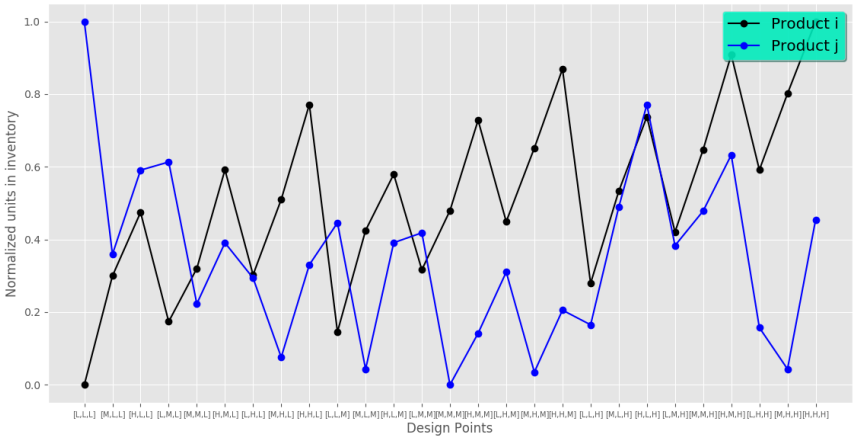


Figure 6.4: Products i and j showing normalized behaviour based on 27 simulations according to the full factorial design.

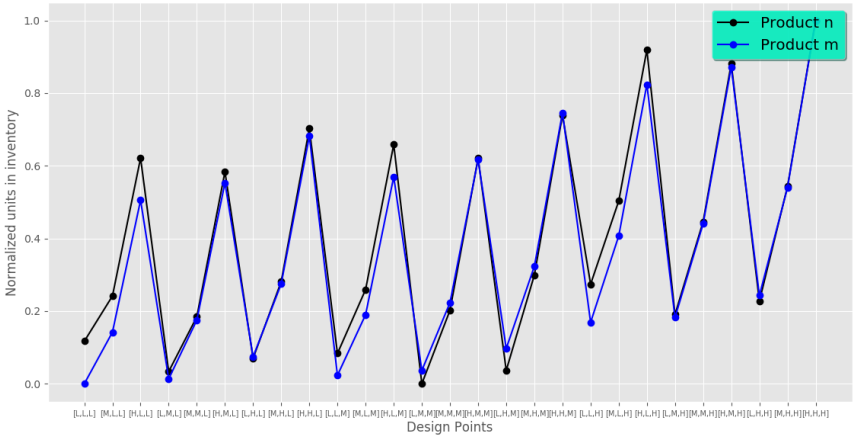


Figure 6.5: Products n and m showing normalized behaviour based on 27 simulations according to the full factorial design.

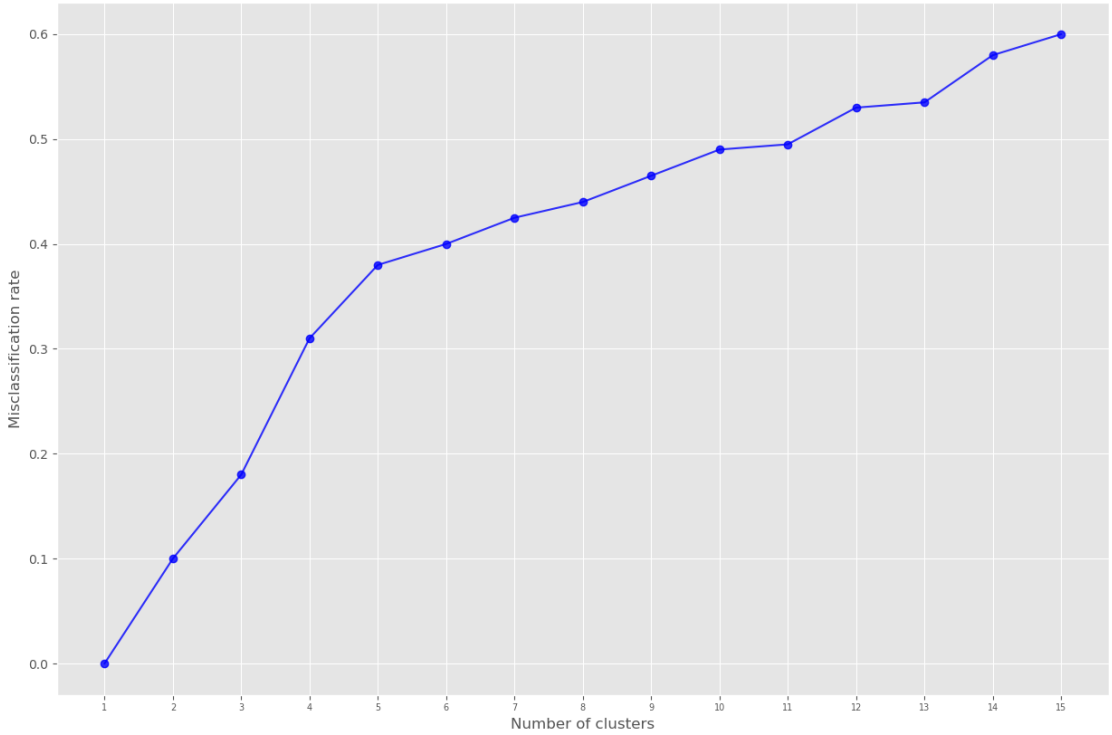


Figure 6.6: Trade-off curve that enables us to select the right number of clusters via the elbow rule.

6.3.2 Clustering via k-means

Based on the results of the full factorial experiment, a product is characterized by a vector of 27 settings that summarizes its behaviour in the network. After normalizing all the 852 vectors, we compared products' vectors via clustering k-means as described in Section 5.2.

The approach on how to select the number of clusters k is described in Section 5.2.2. We performed $m = 20$ independent full factorial experiments with different customer demand seed. We then performed a k-means analysis over m independent full factorial experiments for a range of k values to establish the true cluster label of a product. The true cluster label was then compared with the m individual cluster labels to determine the misclassification error. Plotting the misclassification error for a range of k values leads to the plot in Figure 6.6. Finally, using the elbow rule we were able to select the number of clusters $k = 5$ from Figure 6.6 .

After selecting k=5, we partitioned the 852 products into 5 clusters via k-means, as shown in Table 6.3. Table 6.3 shows that there are only 18 products in Cluster 3; these represent only 2.2% of the total products. In the remaining part of this thesis, products in Cluster 3 are included. However, a second approach could be to exclude those products from the analysis and to have a dedicated policy for each one of them.
Each product's vector has 27 components, which makes it hard to display in the 3-D space. However, we used PCA to reduce the 27 dimensions to three principal components that are representable in the 3-D space. By applying firstly k-means and then PCA, we were able to construct the plot in Figure 6.7; each dot in the figure represents a product. The numbers 1,2,3,4 and 5 represent the location of the cluster's centroids. The centroid of Cluster 3 is the most distant from the centroid of Cluster 2, which suggests that products belonging to Clusters 3 and 2 have quite different behaviours.
Considering the results of Figure 6.4, we concluded that products i and j show different behaviours. After performing k-means, we found that product i falls into Cluster 2 and product j into Cluster

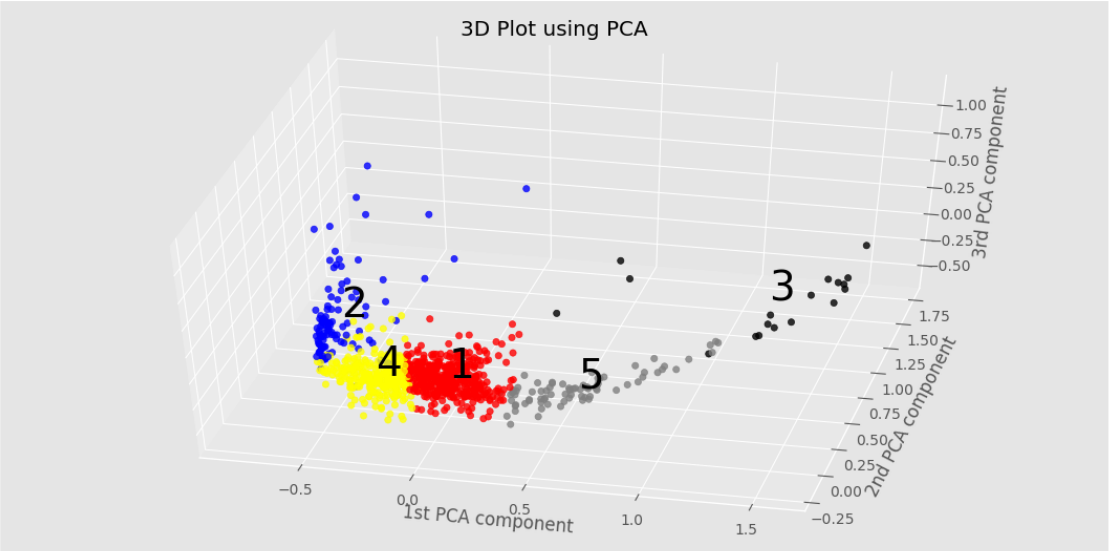


Figure 6.7: Results of k-means analysis with k = 5. Results are displayed in the 3-D space by using PCA.

3. On the other hand, based on Figure 6.5, we concluded that products n and m show similar behaviours. After k-means, we found that both product n and m belong to Cluster 4.

Cluster number	Products in cluster	% of Products in cluster
Cluster 1	384	45%
Cluster 2	105	12.4%
Cluster 3	18	2.2%
Cluster 4	269	31.5%
Cluster 5	76	8.9%
Total	852	100%

Table 6.3: Partition of 852 products in five clusters

Feature type	# of Features
Type I	3
Type II	14
Type III	15
Type IV	15
Type V	3
Total	50

Table 6.4: Features distribution based on the three-stage network

Parameter type	Parameter values
`criterion`	{ gini, entropy }
`max_depth`	{2,3,...,10 }
`max_leaf_nodes`	{ 2,3,...,20 }
`min_sample_split`	{ 2,3,...,20 }

Table 6.5: Grid search for selecting the best combinations of parameters for the decision tree

6.3.3 Classification via decision trees

By applying the features type as described in Section 5.2.1 to the network of Figure 6.1, we defined a total of 50 features which are distributed as described in Table 6.4.
Each product is then characterized by 50 features and a cluster label l such that $l \in L = \{1, 2, 3, 4, 5\}$. Based on this information, we can train a DTree model to predict the cluster label of a product based on its input features.

First of all, the size of the training and testing set needs to be chosen. We use the 70-30 splitting rule which means that the original data are randomly split in a subset containing 70% of the data, which is then used for training the DTree. The remaining 30% of the data are used to test the classifier. However, the training set is chosen such that there is an acceptable number of products representing all the five clusters.

Moreover, the DTree algorithm that is available on the `Scikit-learn` Python library takes several parameters as input. Among them, we optimized four parameters: `max_depth`, `max_leaf_nodes`, `min_sample_split` and `criterion`.
Based on the four parameters, we constructed a grid search for which all combinations of parameters are tested. The score of each combination of parameters is based on the accuracy measured on the testing sample. The grid search with all the parameters combinations is shown in Table 6.5, and we selected the combination of parameters with the highest accuracy on the testing sample.
The optimal set of parameters is shown in Table 6.6 with a max accuracy score on the testing sample equal to 80%. The accuracy results over the testing and training set are shown in Table 6.7.

Based on the parameters of Table 6.6, we construct the DTree shown in Figure 6.8. The DTree

Parameter type	Optimal Parameter value
`criterion`	gini
`max_depth`	2
`max_leaf_nodes`	6
`min_sample_split`	5

Table 6.6: Best set of parameters with 80% accuracy on the testing sample

Data set	Accuracy
Training set	83%
Testing set	80%

Table 6.7: Accuracy results of the classification tree on the training and testing set

has been trained with a training sample corresponding to 70% of the original dataset. The tree in Figure 6.8 is the result of carefully selecting the DTree's parameters such that the accuracy over the testing set is maximized.

The tree in Figure 6.8 shows that among the 50 features available, only three features are used to diversify products into five classes. These features are $Pool_{LDC_1}$, $Pool_{LDC_2}$ and CV_{DC}. The sample value in the tree nodes indicates the number of training products. For example, the starting nodes contains 596 products, which correspond to 70% of the entire product selection. Moreover, each node contains information on how the training products are distributed within categories. For example, the first node contains 596 products, 280 of these belongs to Class 1, 72 to Class 2, 12 to Class 3, 187 to Class 4, and 45 to Class 5. This information helps us to quantify the misclassification error at each leaf. For example looking at Class 2 leaf, we have a misclassification error equal to $1 - \frac{70}{1+70+2+18+0} = 23\%$. If we look at the Class 3 leaf, the misclassification error is equal to zero since all the products in that node belong to Class 3. The leaf gets assigned the label that corresponds to the largest class in the node.
Moreover, there are two leaves corresponding to Class 1. As it can be seen, one Class 1 leaf is blue since the misclassification error is 50% and contains only a sample of seven Class 1 products. Moreover, the grey Class 1 leaf shows the highest number of Class 1 products in this class (237), which makes the blue leaf less reliable for drawing general conclusions about products in Class 1. However, given the 50 features and the optimized tree parameters, we reached a max accuracy on the testing sample equals to 80%. This means that we misclassify 20% of the products on average. However, by looking at Figure 6.7, the misclassification happens for classes that are neighbours in the clustering analysis. For example, the Class 4 leaf misclassifies 32 Class 1 products; however, Class 1 and Class 4 are neighbours in the clustering analysis.
In the last section, we combined the results of the DTree with the optimal safety factors results from the GA. In this way, we attempt to explain why the selected tree features are relevant for setting the optimal inventory policy that is defined for each product category.

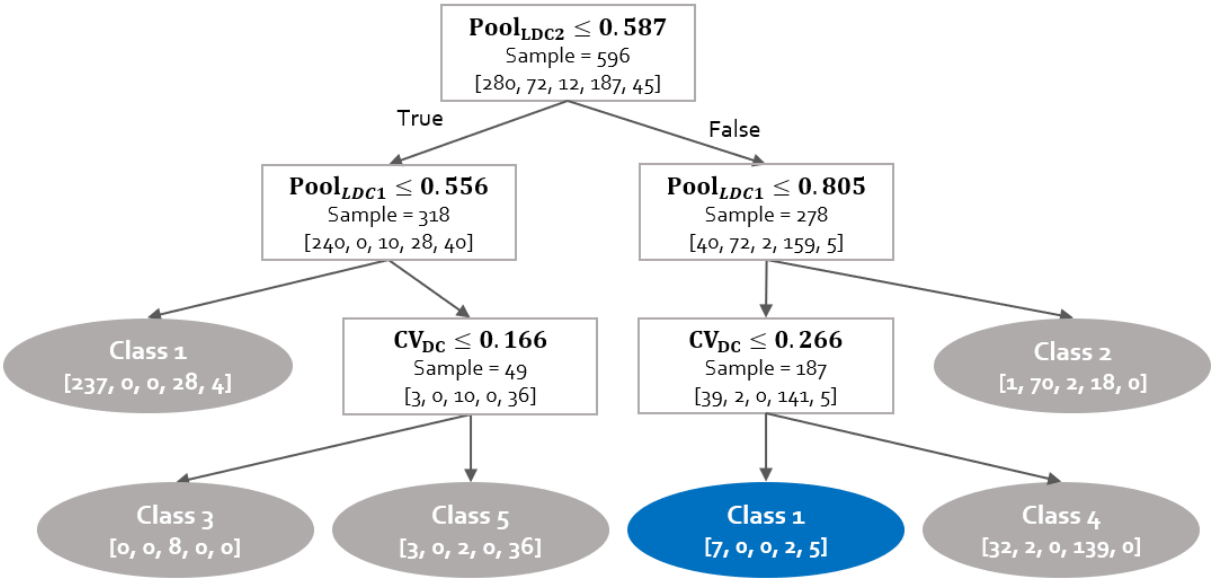


Figure 6.8: Classification tree based on the parameters in Table 6.6.

6.4 Three-echelon inventory optimization

The GA is the method that is used for finding the optimal values of safety factors at upstream locations. Three inputs for the GA needs to be defined, namely a stopping parameter, the initial population size, and the mutation rate.
The stopping criteria are given by three different conditions: 1) a max number of iterations of 50 is reached; 2) a convergence criterion is met; and 3) a no convergence criterion is met. The GA stops as soon as one of those conditions is met. For the convergence criterion, we stop the GA if the absolute difference between the new and the old solution is less than 0.1% for five consecutive iterations. For the non-convergence criterion, we stop the GA if no improved solutions are found after 10 consecutive iterations. The initial chromosomes population is set equal to 10 while the mutation rate is set equal to 0.1.

Based on the results of the DTree, we classify products into five categories. For each upstream location in the network, we define five safety factor variables for each product category. This means that a chromosome for the GA has the following structure:

$$\mathbf{k} = \begin{pmatrix} k_{DC,1} & k_{DC,2} & k_{DC,3} & k_{DC,4} & k_{DC,5} \\ k_{L_{DC_1},1} & k_{L_{DC_1},2} & k_{L_{DC_1},3} & k_{L_{DC_1},4} & k_{L_{DC_1},5} \\ k_{L_{DC_2},1} & k_{L_{DC_2},2} & k_{L_{DC_2},3} & k_{L_{DC_2},4} & k_{L_{DC_2},5} \end{pmatrix} \tag{6.1}$$

where $k_{i,j}$ represents the safety factor at location i for product category j with $i \in \{DC, L_{DC_1}, L_{DC_2}\}$ and $j \in \{1, 2, 3, 4, 5\}$.

6.4.1 Initial population for the genetic algorithm

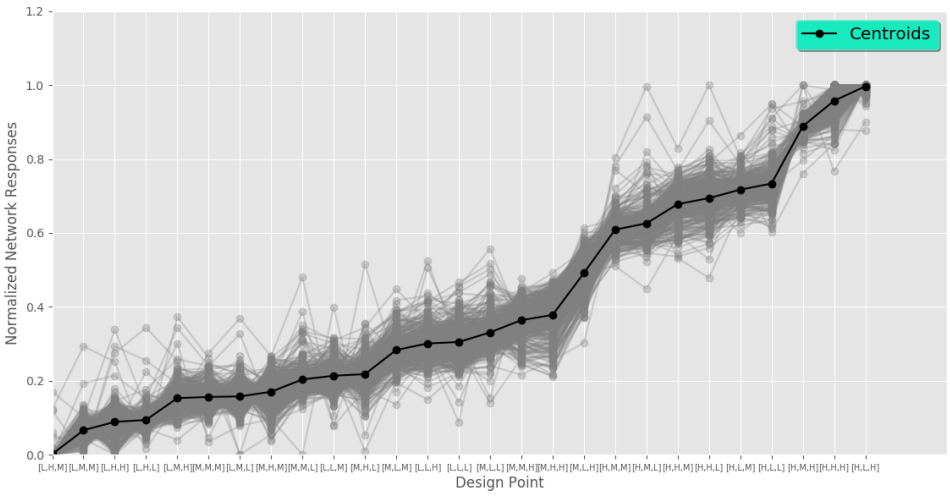
The first step of the GA is to generate an initial population in the form of safety factors at upstream locations. Following the work of van Liempd, each gene of a chromosome is generated by randomly drawing from a uniform distribution $U(0, 2)$. This implies that we allow a maximum safety factor value of two and a minimum value of 0.
However, we can use the results of the full factorial design and clustering to generate a better initial population. To have a representative behaviour of each cluster, we used the corresponding centroids. The five centroids consist of five vectors with a length of 27. Each of the 27 components corresponds to a design point of the full factorial design. If for each centroid we rearrange the vector elements in ascending order, we can construct plots as seen in Figures 6.9 and 6.10. Looking at Figure 6.9a we see a black line representing the centroid behaviour for Cluster 1 and the grey lines representing the behaviour of all the products belonging to Cluster 1. All the products' responses are ordered according to the indices of the centroid that are ordered in ascending order. This ordering is done because we are interested in finding the optimal safety factors configuration that corresponds to the lowest inventory level. However, picking the minimum value based on the centroid ordering could lead to completely wrong results due to deviation of single products behaviour from the centroid's behaviour. For this reason, we constructed the box plot of Figure 6.9b. On the x-axis, we used the already mentioned three-digit notation X-X-X where the first digit refers to the safety factor value at Central DC, the second digit refers to the safety factor value at Local DC_1, and the third digit refers to the safety factor value at Local DC_2. In reading the box plot in Figure 6.9b from left to right, we note that for the first 11 instances, the value of the safety factor at the central distributor is set either to L (k = 0) or M (k = 1), and that the box plot of the first configuration L-H-M does not overlap with the box plot of the 12th configuration M-L-M. This suggests that for products belonging to Category 1, it is highly unlikely that the optimal inventory setting corresponds to the case where at the central DC, the value of safety factor is set to H (k = 2). By keeping this in mind, we draw from a uniform distribution $U(0, 1.25)$ when generating the initial population of chromosomes for the gene corresponding to Category 1 at the central DC location.
However, Figure 6.10 shows that the situation for products in Category 3 is much more volatile compared to products in Category 1. Indeed, if we look at the box plot of Figure 6.10b from left to right, we note that for the first 8 configurations, the value of safety factor at the central DC is set to L, M, and H. This means that for products in Category 3 at the central DC, we cannot make

Network location	Class 1	Class 2	Class 3	Class 4	Class 5
DC	$U(0, 1.25)$	$U(0, 1.25)$	$U(0, 2)$	$U(0, 1.25)$	$U(0, 1.25)$
LDC1	$U(0.75, 2)$	$U(0, 2)$	$U(0, 2)$	$U(0, 2)$	$U(0.75, 2)$
LDC2	$U(0, 2)$	$U(0, 2)$	$U(0, 2)$	$U(0, 2)$	$U(0.75, 2)$

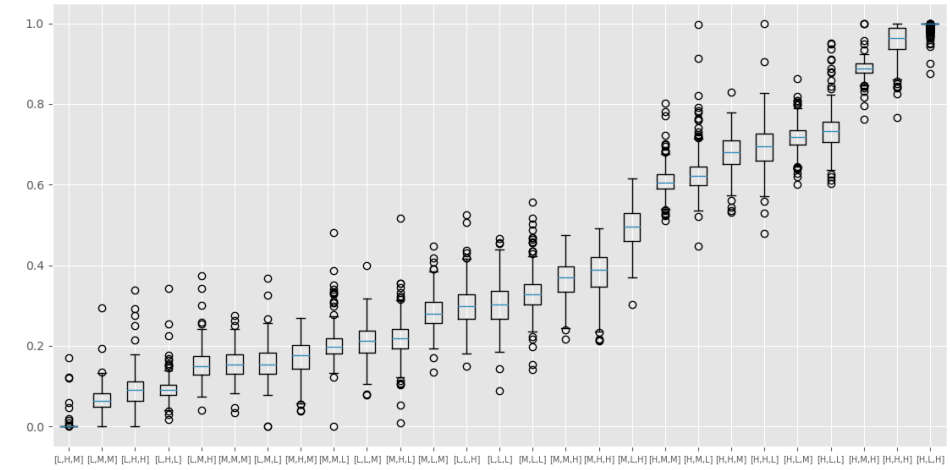
Table 6.8: Uniform distribution types for the creation of the initial population

clear assumptions about the optimal value of the safety factor. For this reason, we initialized the corresponding gene by drawing from a uniform distribution $U(0, 2)$.
The remaining plots for Clusters 2, 4 and 5 are shown in Appendix A.
Finally, we generated the initial population by properly drawing from the uniform distributions shown in Table 6.8. The initial chromosome $\mathbf{k}$ is generated as shown in equation 6.2. We used a population size of 10 chromosomes; this means that by drawing 10 times out of equation 6.2, we are able to generate the initial population for the GA.

$$\mathbf{k} = \begin{pmatrix} U(0, 1.25) & U(0, 1.25) & U(0, 2) & U(0, 1.25) & U(0, 1.25) \\ U(0.75, 2) & U(0, 2) & U(0, 2) & U(0, 2) & U(0.75, 2) \\ U(0, 2) & U(0, 2) & U(0, 2) & U(0, 2) & U(0.75, 2) \end{pmatrix} \tag{6.2}$$

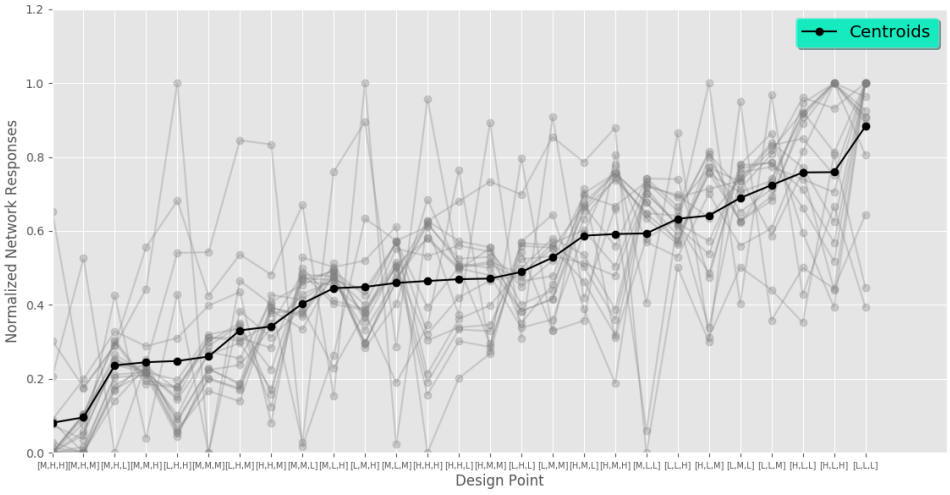


(a)

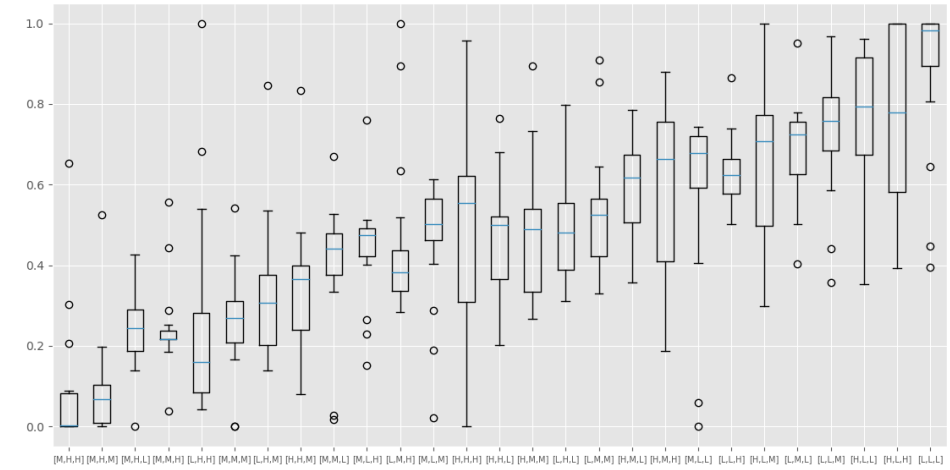


(b)

Figure 6.9: Cluster 1 graphs: (a) Behaviour of products in Cluster 1; (b) box plot for products in Cluster 1.



(a)



(b)

Figure 6.10: Cluster 3 graphs: (a) Behaviour of products in Cluster 3; (b) box plot for products in Cluster 3.

6.4.2 Benefits due to initial population and class crossovers

In this section we present the benefits of using an initial population based on drawing from uniform distributions, as described in equation 6.2 instead of drawing from a uniform distribution $U(0, 2)$ as done by van Liempd. The comparison of the different approaches can be seen in Figure 6.11, which shows the results of the GA; the figure shows the best results found by the GA at a given iteration. For the blue lines, the initial population is selected by randomly drawing from a uniform distribution $U(0, 2)$; for the black lines, the initial population is selected by drawing from uniform distribution $U(x, y)$ where x and y are carefully selected for each gene based on equation 6.2. For all the independent runs in Figure 6.11, the GA stopped after the convergence criterion is reached. We used a class crossover for the GA with a chromosome structure as described in equation 6.2 with five classes of products per upstream location. Figure 6.11 shows that for all the independent runs, the GA converges to a better solution using the differentiated drawing for the initial population. Moreover, on average, the GA converges faster to an optimal solution when the differentiated drawing is used.

The class crossover approach described in Section 5.3.1 is compared with the parameterized uniform crossover used by van Liempd. The comparison results are shown in Figure 6.12. The blue lines show the results of three independent runs of the GA using the parameterized uniform crossover, while the black lines show the results based on the class crossover. First of all, Figure 6.12 shows the best results found by the GA at each iteration. The main observation is that the solutions based on the parameterized uniform crossover show less convergence behaviours; on average, the best solutions are found in the early iterations, after which the solutions of the GA start to diverge. For all the parameterized uniform crossover solutions, the GA stops because the non-convergence stop criterion is met; in other words, this means that after the best solution is found, the next 10 iterations do not provide any improved solutions. On the other hand, the solutions based on the class crossover show a much better convergence behaviour, and the GA finds better solutions as the number of iterations increase. Moreover, for the class crossovers solutions, the GA stops because the convergence criterion is met.

6.4.3 Optimal k-factors

In this section, results are presented based on the GA version in which the selective initial population is used together with the class crossovers. Table 6.9 shows the optimal safety factors, which correspond to the optimal solution found by the GA. In the table, the optimal average safety factors, corresponding to three independent runs, are shown together with the 95% confidence intervals. For example, the optimal average safety factor at the central DC for products belonging to Class 1 is 0.16 with a 95% confidence interval given by $[0.11, 0.2]$. It is also worth mentioning that the results in Table 6.9 suggest using lower safety factors at the central DC compared to higher safety factors at the local DCs. The results obtained in Table 6.9 correspond to the scenario in which a target service level of 98% is reached at the retailers for each product.

Table 6.10 shows the total on-hand inventory and the percentage of the total inventory hold at different locations based on the optimal safety factors of Table 6.9. The results are provided for three different service levels. The obvious result is that by reducing the service level, the total on-hand inventory reduces. Moreover, it is interesting to note how the percentage of on-hand inventory hold upstream increases by reducing the target service level at the retailers. For example, 21.5% of the total on-hand inventory is held at the central DC for a target service level of 98% at the retailers, while 28.7% of inventory is kept at the central DC for a target service level of 85% at the retailers. This behaviour is somehow expected since more inventory is stocked closer to the customers in order to reach a high service level target. It should be mentioned that the whole procedure for building a classification method is optimized for a target service level of 98%. This means that in Table 6.10, we used the product categories based on a 98% service level to show the on-hand inventory results for different target service levels. Furthermore, we expect that the product categories are robust regarding small changes in the target service level. However, by looking at the results in Table

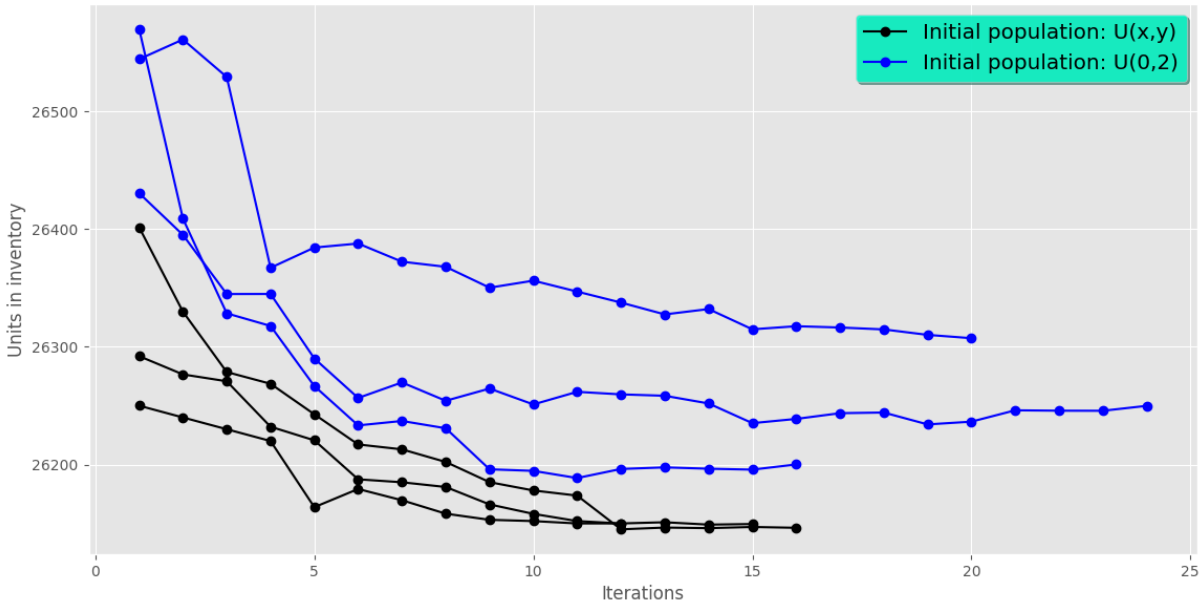


Figure 6.11: The blue lines show the GA iteration results for three independent runs obtained by randomly drawing initial populations from a uniform distribution $U(0, 2)$. The black lines show the GA iteration results for three independent runs; the initial population for this is obtained by drawing from the uniform distributions as described in Table 6.8.

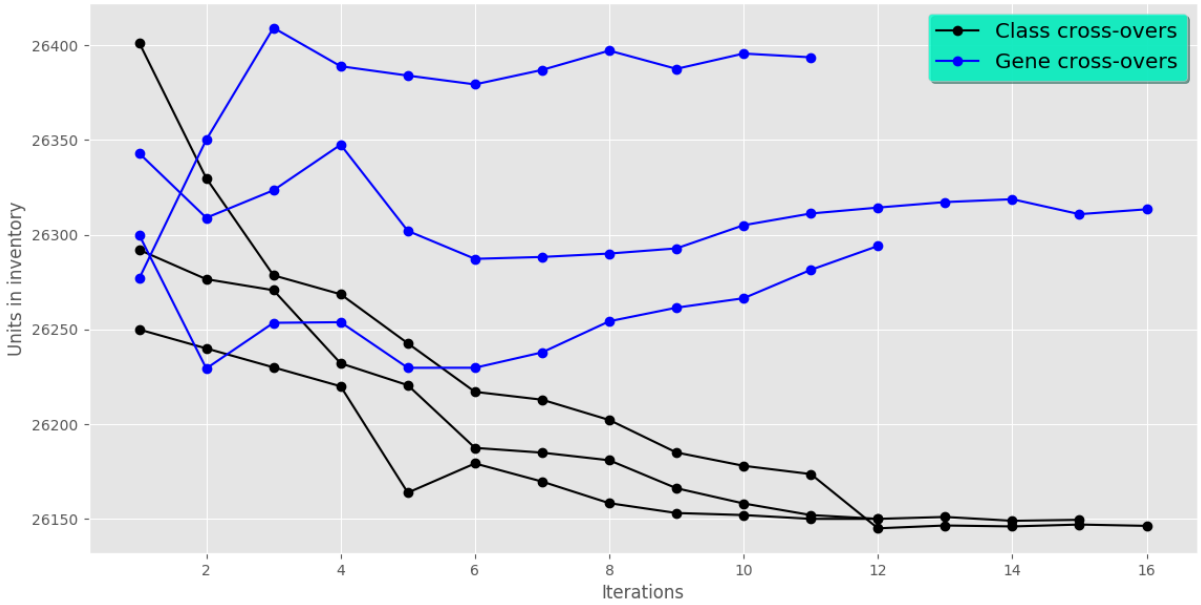


Figure 6.12: The blue lines represent the GA results obtained by performing single gene crossovers for three independent runs. The black lines represent the GA results obtained by performing class crossovers for three independent runs.

Central DC	Class 1	Class 2	Class 3	Class 4	Class 5
Safety factor	0.16	0.32	0.35	0.12	0.53
95% CI	[0.11,0.20]	[0.21,0.44]	[0.18,0.52]	[0.04,0.20]	[0.39,0.68]
Local DC1	Class 1	Class 2	Class 3	Class 4	Class 5
Safety factor	1.58	0.32	1.52	0.84	1.44
95% CI	[1.46,1.70]	[0.07,0.58]	[1.22,1.83]	[0.51,1.17]	[1.24,1.65]
Local DC2	Class 1	Class 2	Class 3	Class 4	Class 5
Safety factor	0.21	0.65	0.88	1.26	1.56
95% CI	[0.06,0.35]	[0.26,1.03]	[0.55,1.2]	[0.68,1.83]	[1.30,1.81]

Table 6.9: Optimal safety factors per product category and upstream location for a 98% service level at the retailers

Service level	98%	90%	85 %
Total on-hand inventory	26100	17400	14700
% of inventory units at Central DC	21.5%	26.8%	28.7%
% of inventory units at Local DC1	18.9%	26.1%	28.0%
% of inventory units at Local DC2	14.4%	19.5%	20.9%
% of inventory units at Retailer 1	1.4 %	0.9%	0.8%
% of inventory units at Retailer 2	4.6%	2.8%	2.2%
% of inventory units at Retailer 3	4.1%	2.48%	2.0%
% of inventory units at Retailer 4	6.9%	4.2%	3.4%
% of inventory units at Retailer 5	5.5%	3.2 %	2.6%
% of inventory units at Retailer 6	5.9%	4.1 %	3.5 %
% of inventory units at Retailer 7	1.7%	1.1%	0.96%
% of inventory units at Retailer 8	2.6%	1.5 %	1.2 %
% of inventory units at Retailer 9	3.2%	1.9 %	1.5 %
% of inventory units at Retailer 10	2.5 %	1.4 %	1.0%
% of inventory units at Retailer 11	2.2%	1.2%	0.9%
% of inventory units at Retailer 12	4.3 %	2.8%	2.3%

Table 6.10: Inventory results for three different service level

6.10, we see that a 8% reduction in service level generates almost 18% increase in the inventory hold upstream compared to the 98% service level case. This is an indication that the whole classification procedure should be repeated for achieving optimal results based on the 90% service level target at the retailers.

Based on the optimal k-factors of Table 6.9, we can see how the on-hand inventory is distributed among product classes-location combinations. Results are shown in Table 6.11. For example, we can see that on-hand inventory for products in Class 1 is almost equally split between upstream and downstream locations. Almost 49% of the inventory is kept upstream and 51% is at the retailers for Class 1 products. A different dynamic can be observed for Class 3 inventory for which 71.4% of on-hand inventory is stored upstream (DC, LDC_1, and LDC_2) and only 28% at the retailers.

Category	Class 1	Class 2	Class 3	Class 4	Class 5
Total on-hand inventory	11745	522	1827	5220	6786
% of inventory units at Central DC	19%	15.4%	30.8%	15.6 %	28.0 %
% of inventory units at Local DC1	19.7%	9.6 %	25.7%	13.5%	20.7%
% of inventory units at Local DC2	10.3%	11.4%	14.9%	15.7%	19.7%
% of inventory units at Retailer 1	2.0%	0.0 %	1.6%	0.5%	1.3%
% of inventory units at Retailer 2	5.5%	4.7%	3.2%	5.4%	3.0%
% of inventory units at Retailer 3	4.8%	10.1%	2.3%	5.7%	2.0%
% of inventory units at Retailer 4	7.7%	16.5%	3.8%	7.8%	5.0%
% of inventory units at Retailer 5	6.4%	5.6%	3.0%	6.8%	3.7 %
% of inventory units at Retailer 6	5.6%	5.8 %	3.6%	8.4%	5.4%
% of inventory units at Retailer 7	2.3%	0.0%	1.8%	0.7%	1.6 %
% of inventory units at Retailer 8	3.0%	2.4%	1.7%	3.0%	1.7%
% of inventory units at Retailer 9	3.7%	7.8 %	1.6%	4.4%	1.6%
% of inventory units at Retailer 10	2.8%	6.0%	1.4%	2.9%	1.9 %
% of inventory units at Retailer 11	2.6%	2.1%	1.3%	2.6%	1.6%
% of inventory units at Retailer 12	4.5%	2.6%	3.1%	6.7%	2.6 %

Table 6.11: Optimal inventory statistics per category groups for a 98% target service level

Central DC	AX	AY	AZ	BX	BY	BZ	CX	CY	CZ	Tot
# products	228	11	0	139	110	0	13	262	89	852
volume %	77.6 %	2.38 %	0 %	9.5%	5.5%	0%	0.31 %	4.2%	0.46%	100%
Local DC1	AX	AY	AZ	BX	BY	BZ	CX	CY	CZ	Tot
# products	103	136	0	3	244	2	0	130	234	852
volume %	48.6%	31.4%	0%	0.24%	14.7%	0.1%	0 %	2.8%	2.2%	100%
Local DC2	AX	AY	AZ	BX	BY	BZ	CX	CY	CZ	Tot
# products	201	38	0	93	154	1	4	246	115	852
volume %	70.5%	9.5%	0%	6.6%	8.3%	0.04%	0.1%	4.2%	0.7%	100%

Table 6.12: ABC-XYZ categorization of products per upstream location

6.4.4 Genetic algorithm comparison with ABC-XYZ

In this section, we compare the optimal results obtained in the previous section with the optimal results obtained by using the ABC-XYZ categorization.
For ABC, we categorized products according to their value. However, since we do not have any price information, we used product volume as a representation of value. For the ABC, we used an 80/95 threshold, which means that A-products are responsible for 80% of the total volume, B-products are responsible for 15% of the total volume, and C-products are responsible for the remaining 5% of the total volume. For the XYZ, we used a 0.4/1 coefficient of variation threshold. We labelled all the products with a coefficient of variation less than 0.4 as X-products, while Y-products are all those with a coefficient of variation between 0.4 and 1, and Z-products all have a coefficient of variation greater than 1. Moreover, the ABC–XYZ categorization is applied per location; for example, this means that a given product belongs to category AY at the local DC, and it may belong to category BX at the central DC. Table 6.12 shows how products are distributed within the ABC–XYZ categories. At the central DC there are relatively more X products than Y or Z products. This is a clear example of pooling effect in which the aggregated demand upstream is more stable than the disaggregated demand downstream.
In our case, we have a network consisting of three upstream locations; this means we defined ABC–XYZ categories at three different locations. The chromosome structure for the GA based on the ABC–XYZ categorization looks as follows:

$$\mathbf{k} = \begin{pmatrix} k_{DC,AX} & k_{DC,AY} & k_{DC,AZ} & k_{DC,BX} & k_{DC,BY} & k_{DC,BZ} & k_{DC,CX} & k_{DC,CY} & k_{DC,CZ} \\ k_{LDC_1,AX} & k_{LDC_1,AY} & k_{LDC_1,AZ} & k_{LDC_1,BX} & k_{LDC_1,BY} & k_{LDC_1,BZ} & k_{LDC_1,CX} & k_{LDC_1,CY} & k_{LDC_1,CZ} \\ k_{LDC_2,AX} & k_{LDC_2,AY} & k_{LDC_2,AZ} & k_{LDC_2,BX} & k_{LDC_2,BY} & k_{LDC_2,BZ} & k_{LDC_2,CX} & k_{LDC_2,CY} & k_{LDC_2,CZ} \end{pmatrix} \quad (6.3)$$

However, the variables $k_{DC,AZ}$,$k_{DC,BZ}$, $k_{LDC_1,AZ}$, $k_{LDC_1,CX}$ and $k_{LDC_2,AZ}$ are dummy variables since there are no products corresponding to these categories.

Figure 6.13 shows the GA iterations both for the ABC-XYZ categorization and for the previously described five-class categorization. For each categorization method, three independent GA runs are performed with a 98% service level target for all products at the retailers. The starting solutions based on the ABC-XYZ are always higher than the starting solutions based on the five-class method. This is because the initial populations for the two methods were generated differently. For the five-class solutions, we generated a sub-optimal initial population by randomly drawing from the distributions described in Table 6.8, while for the ABC-XYZ solutions, we generate the initial population by drawing from the uniform distribution $U(0, 2)$. The sub-optimal initial population cannot be applied to the ABC-XYZ solutions since it follows after the DOE and the clustering analysis. Despite the total on-hand inventory units being quite close among all the GA runs, the curves based on the five-class categorization are always below the ABC-XYZ curves. The results based on the five-class method are on average 1% better than the results based on the ABC-XYZ categorization. We could conclude that performance-wise, the two methods show almost identical results. However, it is remarkable that we were able to reach the same performance of the ABC-XYZ categorization by

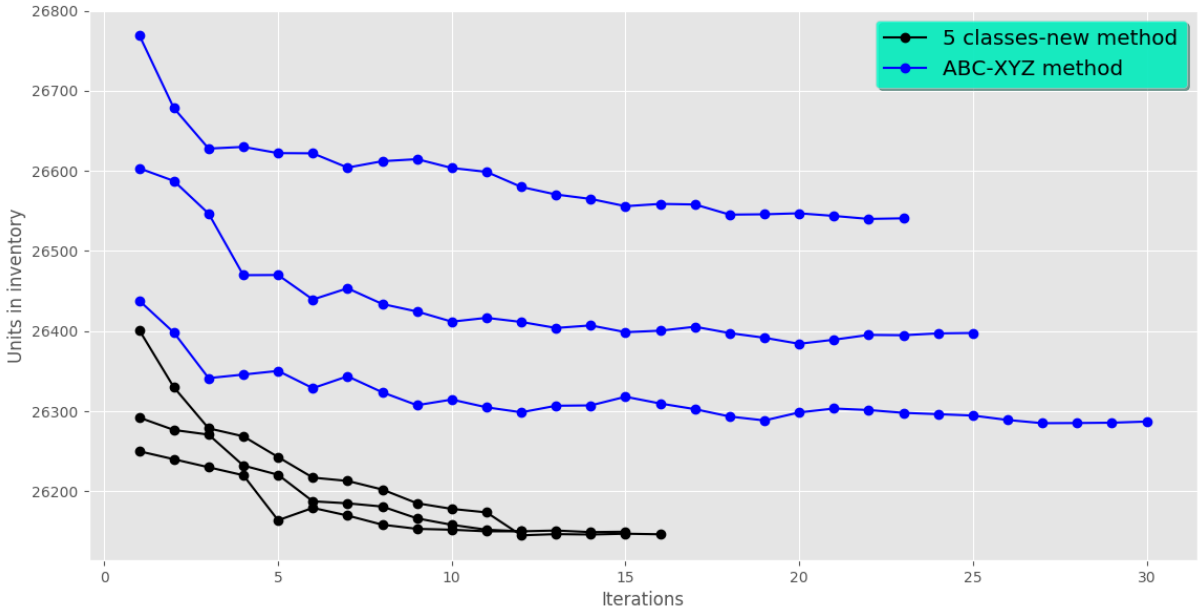


Figure 6.13: GA results based on ABC-XYZ categorization compared with the five-class categorization. Comparison is done by reaching the same 98% service level at retailers.

only using five product categories per location. We would expect that by increasing the number of categories we would have a better performance due to a higher policy diversification for products. This shows that with the five-class method, we classify on dimensions that are more relevant to the scoping of multi-echelon inventory reduction.

Table 6.13 shows the optimal average safety factors for the ABC–XYZ categorization logic. In addition to the average results, the 95% confidence interval for each safety factor is also shown. Moreover, there are no optimal safety factors for some categories since no products fall into those categories.

Central DC	AX	AY	AZ	BX	BY	BZ	CX	CY	CZ
Safety factor	0.14	0.8	-	0.28	0.4	-	0.29	0.14	0.6
95% CI	[0,0.24]	[0.65,1.03]	-	[0,0.4]	[0,0.74]	-	[0.1,0.5]	[0.1,0.23]	[0.0,1.2]
Local DC1	AX	AY	AZ	BX	BY	BZ	CX	CY	CZ
Safety factor	1.1	0.46	-	1.26	0.6	1.24	-	0.9	0.7
95% CI	[0.5,1.6]	[0.1,0.9]	-	[1.1,1.4]	[0,1.1]	[0.8,1.6]	-	[0.6,1.24]	[0.5,0.9]
Local DC2	AX	AY	AZ	BX	BY	BZ	CX	CY	CZ
Safety factor	0.58	0.8	-	1.33	0.6	1.27	0.63	1.11	0.55
95% CI	[0.2,0.9]	[0.3,1.3]	-	[0.8,1.9]	[0.2,1]	[0.8,1.7]	[0.3,1]	[0.9,1.3]	[0,1.1]

Table 6.13: Optimal safety factors based on the ABC-XYZ categorization, for which a 98% service level is reached

6.5 Overall approach

In this section, the overall approach is summarized. Here, we assume that we are a company interested in setting an optimal inventory policy per product category, since a policy for each individual product is too hard to maintain.
First of all, a classification tree is designed by following the tree steps approach that involves DOE, clustering, and DTrees. The clustering analysis points out how many product categories are needed, while the DTree shows which features need to be considered in order to assign products to classes.
The second step is to determine the optimal safety factor values defined per product category at upstream location. The optimal safety factors are determined via the GA described in Section 5.3.
The final step is to combine the classification tree results with the optimal safety factors results. The overall and final results can be seen in Figure 6.14. The DTree in Figure 6.14 is optimized for a 98% service level at the retailers since the clustering analysis is based on a DOE for which we set a 98% service level target at the retailers. This means that if the company wants to reduce or increase the service level then the overall procedure should be repeated. However, in practice, the service level constraints do not change overnight. This means that the frequency of updating the DTree of Figure 6.14 depends on the changes on the service level. Moreover, the DTree of Figure 6.14 should be updated in order to face customer demand changes as well.
However, the advantage of having a tree as in Figure 6.14 is that the optimal inventory parameters are easily accessible by only looking at few features. The complexity of the whole procedure pays off by delivering a tool that is easy to use.

Looking at Table 6.11, we can see that there is on average, a relatively high percentage of inventory held at the central DC. This may seem in contrast with the optimal safety factor values at the central DC that are set to relatively low numbers for all classes. However, the central DC is the only location directly connected to the external supplier. This means that all goods need to flow through the central DC before being distributed downstream in the supply chain. This effect, together with the use of a review period, makes sure that there is always an intrinsic relatively high percentage of inventory stocked at the central DC.
Some general characteristics of the five classes of products are listed:

- **Class 1:** Table 6.11 shows that the inventory is kept balanced between upstream and downstream locations with a percentage split of approximately 49% and 51% respectively. Looking at the classification tree of Figure 6.14, we see that there are two leaves labelled as Class 1. However, the blue leaf is characterized by a high misclassification error and with few Class 1 observations in the node. To draw general conclusions about Class 1 products, we look at the tree patch that leads to the grey Class 1 leaf. Class 1 products are characterized by a relatively low value of the $Pool_{LDC1}$ (≤ 0.556) and $Pool_{LDC2}$ (≤ 0.587) features which make them a good candidate for demand pooling. As stated in the features definition, the combination of the *Pool* feature and the *CV* feature should be enough to establish whether demand pooling can add any value. In investigating further, we note that products in Class 1 are characterized on average by a higher CV at Retailers 1 to 6 compared to Retailers 7 to 12. Moreover, in looking at Table 6.11, we note that most of the demand pooling happens between Local DC1 and Retailers 1 to 6. This effect is also confirmed by the optimal values of safety factors that are higher at the Local DC1 compared to the Local DC2.

- **Class 2:** Looking at Table 6.11, we see that approximately 36.4% of inventory is held upstream while the remaining 63.6% is pushed at the retailers. This is also confirmed by the value of the optimal safety factors, which are set to relatively low values at the upstream locations. Moreover, by looking at the DTree, we see that Class 2 products are characterized by a relatively high value of the $Pool_{LDC1}$ and $Pool_{LDC2}$. This is due to the fact that the products in Class 2 are characterized by low points of sales at the retailers. This means that customer demand for Class 2 products takes place at one or two retailers, and because of this there is no need to keep inventory centrally.

- **Class 3:** Table 6.11 suggests a pooling effect between the local DCs and the central DC and between Local DC1 and Retailers 1 to 6. We see that 71.4 % of inventory is kept upstream and only 28.6% at the retailers. The classification tree assigns products to Class 3 when the coefficient of variation at the central DC is quite small (≤ 0.166). This suggests that aggregating all the incoming demand at the central DC generates a stable demand pattern versus a potentially less stable demand at the retailers.

- **Class 4:** Table 6.11 shows that approximately 55% of the inventory is held at the retailers while the remaining 45% is almost equally split among the upstream locations. The classification tree assigns products to class 4 given a $Pool_{LDC2} > 0.587$, a $Pool_{LDC1} \leq 0.805$ and a $CV_{DC} > 0.266$.

- **Class 5:** 68.4% of on-hand inventory is held upstream and only 31.6% at the retailers. This is partially confirmed by the optimal value of safety factors that have a relatively high value at Local DC1 and DC2. Moreover, products in Class 5 are characterized by a value of the $Pool_{LDC2} \leq 0.587$, a $Pool_{LDC1} > 0.556$ and a $CV_{DC} > 0.166$.

The DTree offers some intuitive results for some products classes (e.g., Classes 1, 2, and 3) while for some other classes, results are harder to interpret. This is mostly due to the fact that some features are correlated to one another. For example, we defined 15 coefficient of variation features that correspond to the 15 locations in the network. However, based on the tree results, we see that only the CV feature at the central DC is selected. This may be because the CV at the central DC is related to the demand volatility at the local DCs, and the CV is also related to the demand volatility at the retailers. For example, we prefer to see the CV features at the retailers to have a direct interpretation of the demand volatility downstream. For example, the $CV_{DC} \leq 0.166$ for products in Class 3 tells us that aggregating all the normally distributed demands coming from the 12 retailers generates a stable demand at the central DC. However, does this mean that the demand downstream was much more unstable or that it was already stable? Unfortunately, we are not able to answer the question by looking at the DTree results in which only the CV_{DC} is used.

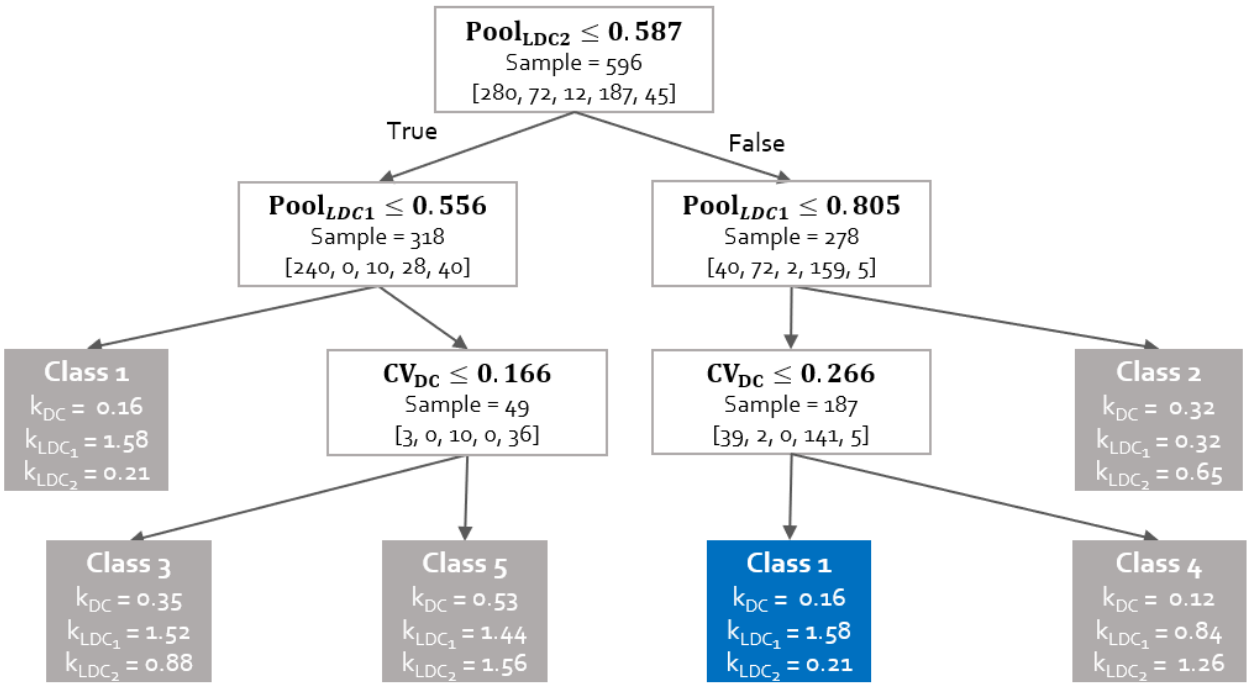


Figure 6.14: Classification tree with optimal safety factors for a 98% service level at the retailers for each product.

7 | Conclusion

In the first part of this thesis, we presented a product classification method which adapts to the underlying supply chain network and suggested which are the most valuable features of the network. The identified features were then used to establish which product belongs to which category. The second part of the thesis focused on finding the optimal safety stocks levels at all upstream locations in order to deliver the desired end-customer service levels at minimum network inventory. The results were tested on a three-stage distribution network with a total of 852 products.
When given known safety stock levels at each location in the network, the discrete-event simulation tool allows us to measure the total network inventory under uncertain demand. However, the simulation tool by itself does not guarantee that a certain target service level is reached at the retailers. For this reason, a two-step approach was used: firstly, we measured the realized service levels at the retailers once the simulation had been completed, and secondly we adjusted the safety stocks at the retailers level (via the SSAP) such that a given target service level can always be reached. The discrete-event simulation together with the SSAP represent the tool box that allows us to evaluate the inventory costs of the three-stage network for for a given service level and a given set of safety factors at upstream locations. The safety stock levels at each upstream location were determined via the policy given by equation 5.1.

We found product categories by studying how the inventory costs curve changed with the safety factors, and this was done for each product. Consequently, products with a similar inventory cost curve are likely to be placed in the same product category. This means that the proposed categorization method is tailor-made for situations where the main cost function is represented by inventory costs. However, the approach could be easily extended for new metrics used for measuring similarities among products. In the research, the inventory cost curve of each product were constructed by using the full factorial experimental design. More precisely, each experiment consisted of running both the discrete-event simulation and SSAP for a given service level and a given set of safety factors at upstream locations. For each experiment, the total on-hand inventory was then measured. By running a finite number of experiments, the full factorial design allows us to capture how the inventory cost curve changes with the safety factor values for a given target service level. After all the simulations are performed based on the full factorial design, we made use of k-means clustering technique to capture similarities among products. The k-means analysis resulted in partitioning the products into five clusters. The five-cluster result from k-means suggests that products should be divided into five groups. However, we do not have any prior information regarding the relevant dimensions or features that lead a product to belong in one of the five clusters. In order to understand which are the relevant dimensions or features for the given network, we made use of a classification DTree. The DTree was trained using 50 product features and the clustering results. The results of the trained tree indicate that three out of 50 features are relevant for making a clear distinction among the five product clusters. Moreover, the power of such a DTree is that it is able to assign a category to a product based on only its features. This means that once a new product is introduced, there would not be any need to rerun the whole process; instead, we could simply look at its features in order to assign a category to it by using the DTree. The accuracy of the DTree on the testing set is 80%. For the inventory optimization section, we assigned each product to a class based on the Dtree prediction.

Once the products were divided into five categories, we moved to the second topic of the thesis, which is based on finding the optimal safety stock levels for the upstream locations via the GA. One of the first steps of the GA is to generate an initial population of chromosomes. Moreover, based on the full factorial design results, we were able to generate the already sub-optimal initial population. Using sub-optimal initial populations instead of a random initial populations, two benefits can be seen, namely that the GA converge to optimal solutions more quickly, and the GA was able to reach lower inventory costs on average. One of the main operators for the GA is represented by the crossover. This operator is responsible for combining parents' gene to generate a new offspring. Compared to the parametrized uniform crossover approach by van Liempd, we used a class crossover approach instead, which shows a much more stable convergence behaviour. One reason to explain why the class crossovers seems to work better is attributed to the fact that it accounts for a potential correlation of safety factors at different locations. For example, if the optimal inventory costs for a certain product category is to have high safety stocks at the central DC and low safety stocks at the local DCs with class crossovers, we are able to preserve this information once it is available in the chromosome. It is also interesting to see how the amount of inventory hold at upstream locations changes with the service level. For a high service level, more inventory is held at the retailers. Indeed, to fulfil a high service level, more stocks are kept closer to the customers' demand. On the other hand, when we reduce the service level, we note that more inventory is held upstream in the supply chain.
The GA provided the optimal set of safety factors per each of the five product classes at the upstream locations. Then, we integrated the Dtree results with the GA results. This led to a Dtree that not only predicts the class of a product but it also predicts the optimal safety factor values at upstream locations to be able to reach a given target customer service level.
The power of such a DTree is that it assigns a class to a product together with the optimal inventory settings by only looking at few features. This means that if new products are introduced, we are able to predict their classes and their optimal inventory settings by simply using their corresponding features to access the DTree. In conclusion, the lengthy approach delivers an easy to use tool that requires sporadic maintenance to face, for example, big changes in customers' demand and service level constraints.

Finally, we compared the optimal GA solutions based on 5-classes approach with the ABC-XYZ categorization approach. For the ABC-XYZ scenario, we had chromosomes consisting of 27 genes, while for the 5-categories we had chromosomes formed by 15 genes. Firstly, we saw that, based on tree independent runs, the total on-hand inventory based on the 5-classes approach was always lower than the ABC-XYZ case. Secondly, the GA for the 5-class approach converged much quicker to an optimal solution than the ABC-XYZ approach. The GA for the ABC-XYZ case took, on average, 26 iterations to converge to an optimal solution while for the 5-classes approach it took, on average, 15 iterations. The difference in convergence speed could be explained by the fact that for the ABC-XYZ case we have longer chromosomes, hence, more gene combinations to explore. Moreover, the starting solutions for the 5-classes approach were always lower than the ABC-XYZ case. This was due to the fact that, for the 5-classes case, we used sub-optimal initial populations. Furthermore, with the ABC-XYZ categorization nine product categories were formed at each upstream location versus five product categories based on the proposed method. Conceptually, we would have expected that a higher product diversification would lead to better results in terms of inventory costs reduction. However, this was not the case since the optimal results based on the 5-classes approach were always better than the ABC-XYZ approach even if the improvement was, on average, only 1%.
This indicates that a higher product diversification if not properly done may not automatically lead to better results. The proposed 5-classes method not only performed slightly better than the ABC-XYZ, but it also reduced the complexity of the product portfolio since less categories were formed compared to ABC-XYZ. The reduction in complexity also brought some benefits for the GA, that converged to optimal solutions in shorter times since there were less genes combinations to explore.
The ABC-XYZ categorizes products based on two dimensions that might not be optimal for the replenishment policy based on product categories. For the proposed method, however, we used an opposite approach. First, we study the inventory cost curve of each product and then we captured

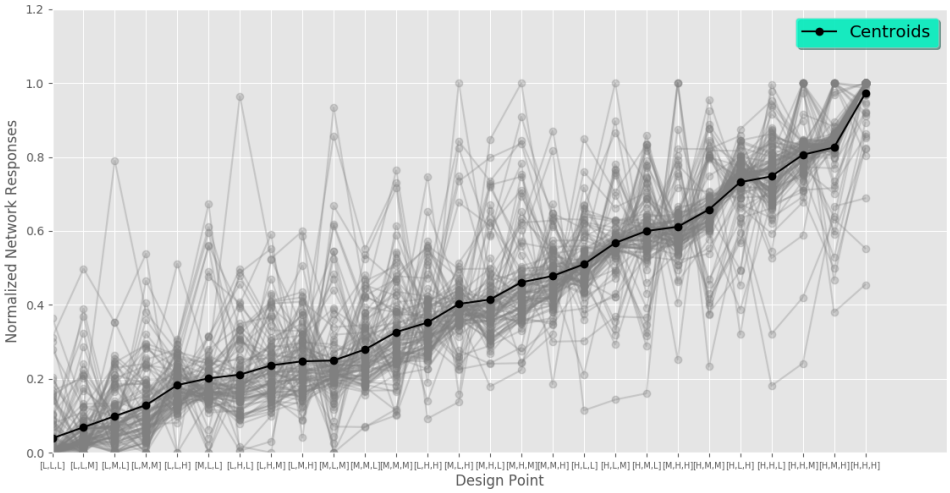
the relevant dimensions that should be used for categorization.

Looking at the limitations of this research, computing time represents one of the biggest limitations. We run the analysis on a intel core i7-3630QM with 8GB RAM machine and it took approx. 0.06 s per product to perform one simulation run together with the SSAP. This means that for 852 products the simulation run + SSAP took around 51 seconds. The full factorial design consists of 27 independent simulation runs which corresponds to a total time of approx. 23 minutes (51 seconds x 27). To find the optimal solutions via the GA with an initial population of 10 and a total of 20 GA iterations it took approximately $51s \cdot 10 \cdot 20 = 2.8h$.
Full factorial design is used to understand how products behave within the network. In this case, the full design consists of 3^3 experiments since we have three upstream locations and three levels of safety factors. To be able to perform 27 simulations it took approximately 23 minutes based on 852 products. However, for bigger and more complex problems the computing time can grow quite rapidly. Let's for example consider a problem with 2000 products and with five upstream locations network. To be able to run a full design we will need $3^5 = 243$ experiments. Considering the running time of 0.06 per product per experiment, it will take approx $0.06s \cdot 2000 \cdot 243 = 8.1h$ to be able to perform the full design of experiments. A way to partially contain the high computing time would be to consider fractional factorial design instead of full factorial design. Fractional experimental design consists of carefully choosing a subset (fraction) of the experimental runs of a full factorial design. However, since independent experiments are carried out, another potential improvement would be to use parallel computing in which multiple experiments are performed simultaneously.

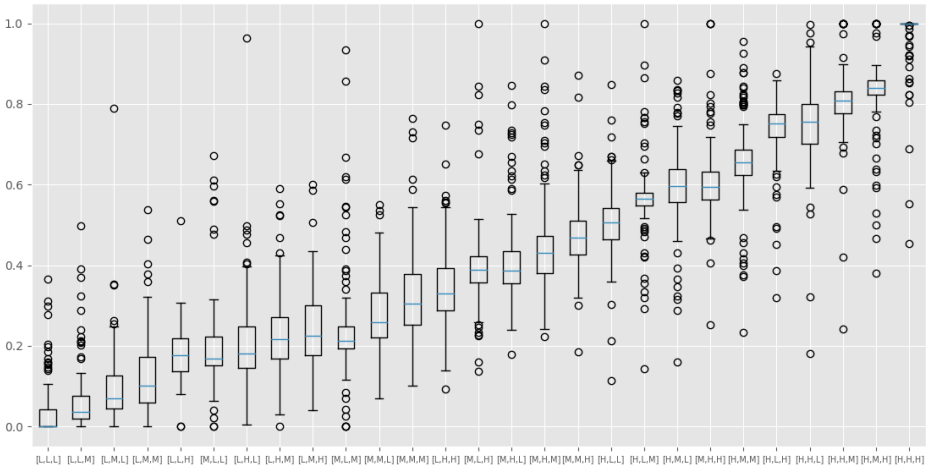
Moreover, another limitation is represented by the features used for training the DTree. To be able to train the decision tree, all product features should be defined beforehand. For a specific problem it may be trivial to retrieve all the most important features. However, in a multi-echelon context, it's far from trivial to think beforehand to all the dimensions/features that might have an important role in determining the optimal inventory settings. The risk is then to leave out some important features that won't be used to train the decision tree. The approach used in this research was to simply list all the features that we could think of based on the data available. Moreover, an important feature to take into account in future work is the Minimum Order Quantity (MOQ) coverage. A high MOQ coverage plays already a safety role against inventory stock-out. This means, for example, that we would expect a low safety factor value for product categories with a high MOQ coverage.

Appendices

A | Genetic algorithm initial population

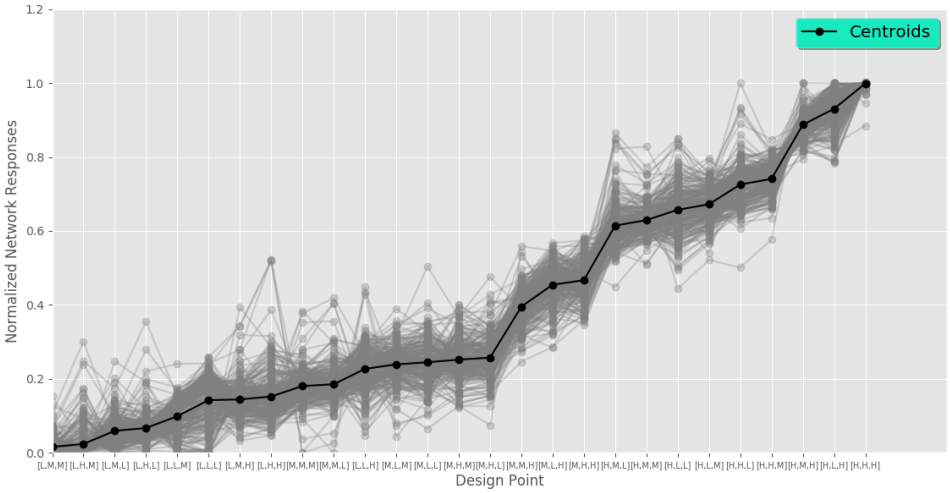


(a)

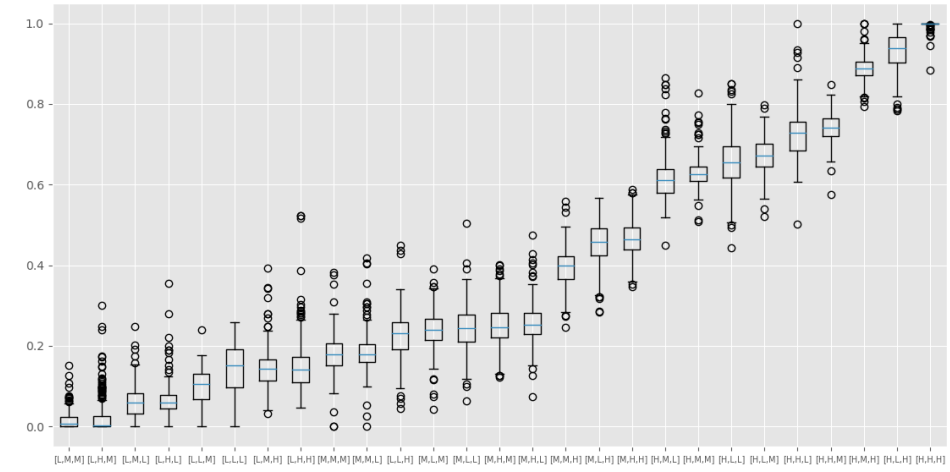


(b)

Figure A.1: (a) Behaviour of products in category 2; (b) box plot for products in category 2.

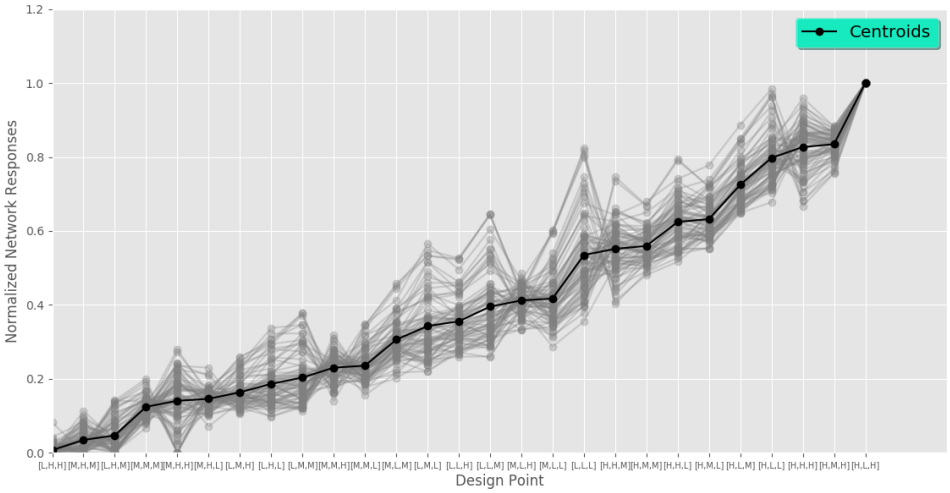


(a)

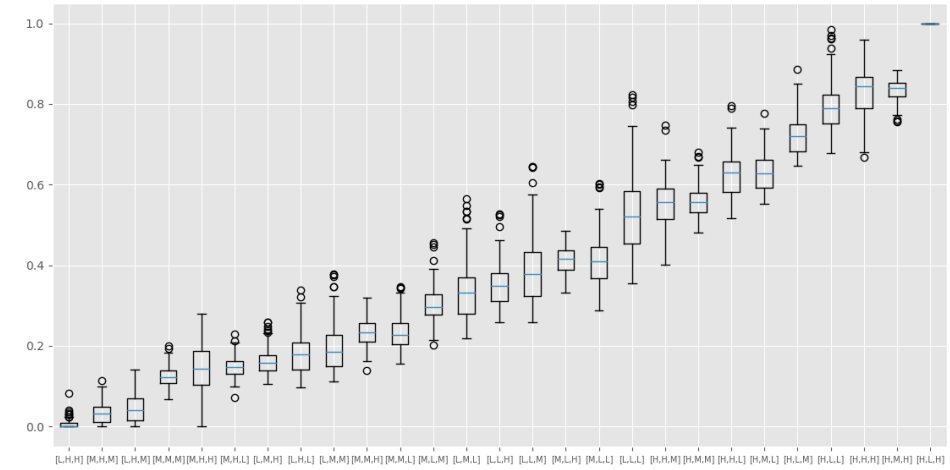


(b)

Figure A.2: (a) Behaviour of products in category 4; (b) box plot for products in category 4.



(a)



(b)

Figure A.3: (a) Behaviour of products in category 5; (b) box plot for products in category 5.

Bibliography

[1] Axsäter, S. (2000). Inventory Control, Springer Verlag, 3rd edition, ISBN 978-3-319-15729-0.

[2] Averill, M. Law. (2007). Simulation modeling & analysis, 4th edition, ISBN 978-007-125519-6.

[3] Brekelmans, R., C., M. (2013). Lecture Notes (35V5A8) Inventory Management, Tilburg University.

[4] Calinski, T., Harabasz J. (1974). A dendrite method for cluster analysis. Communications in Statistics, Vol 3, 1–27.

[5] Celebi, D. (2015). Inventory control in centralized distribution network using Genetic Algorithms: a case study. Computers Industrial Engineering, Vol 87, 532-539.

[6] Chen, F., R. Samroengraja (2000). The Stationary Beer Game. Production and Inventory Management, Vol 9, 19-30.

[7] Clark, A. J., H. Scarf (1960). Optimal Policies for a Multi-Echelon Inventory Problem. Management Science, Vol 5, 475-490.

[8] De Kok, A. G., Grob, C., Laumanns, M., Minner, S., Rambau, J., Schade, K. (2018). A typology and literature review on stochastic multi-echelon inventory models. European Journal of Operational Research, Vol 269(3), 955-983.

[9] Ernst, R.,Cohen, M.A. (1990).Operations Related Groups (ORGs): A Clustering Procedure for Production/Inventory Systems. Journal of Operations Management, Vol 9, 574-598.

[10] Errasti, A.,Chackelson, C.,Poler, R. (2010). An Expert System for Inventory Replenishment Optimization. International Conference on Information Technology for Balanced Automation Systems, 129-136.

[11] Ester, M., H. P. Kriegel, J. Sander, and X. Xu (1996). A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise. In Proceedings of the 2nd International Conference on Knowledge Discovery and Data Mining, 226–231.

[12] Fokoue, E., Clarke, B., Zhang, H. H. (2009). Principles and Theory for Data Mining and Machine Learning, ISBN-13: 978-0387981345.

[13] Ford Dickie H. (1951). ABC Inventory Analysis Shoots for Dollars, Not Pennies. Factory Management and Maintenance, Vol 109, 92-94.

[14] Friedman J.(2001). Greedy function approximation: a gradient boosting machine. Annals of Statistics, Vol 29, 1189–1232.

[15] Gudum,C.K., de Kok, T.G.(2002). A safety stock adjustment procedure to enable target service levels in simulation of generic inventory systems (No.1). Copenhagen Business School, Department of Management Science and Statistics.

[16] Gunther, H.O., Meyr, H.(2009). Supply Chain Planning, Springer Verlag, 1st edition, ISBN 978-3-540-93774-6.

[17] Guvenir,H.A., Erel, E. (1996). Multicriteria inventory classification using a genetic algorithm. European journal of operational research, Vol 105, 29-37.

[18] Hastie, T., Tibshirani, R., Friedman, J. (2008). The Elements of Statistical Learning, 2nd edition (Springer), ISBN-13: 978-0387848570.

[19] James, G, Witten, D., Hastie, T., Tibshirani, R. (2017). Tree-Based Methods. An Introduction to Statistical Learning: with Applications in R. New York: Springer, 303–336.

[20] Holland, J.H.(1975). Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control and artificial intelligence. ISBN: 978-0-262-27555-2.

[21] Jain, A.K. (2010). Data clustering: 50 years beyond K-means. Pattern Recognition Letters, Vol 31, 651–666.

[22] Jolliffe, I.T. (2002). Principal Component Analysis, 2nd edition (Springer), ISBN-13: 978-0387954424.

[23] Kannan, G., Sasikumar, P., Devika, K. (2010). A genetic algorithm approach for solving a closed loop supply chain model: A case of battery recycling. Applied Mathematical Modelling, Vol 34, 655-670.

[24] Klosterhalfen S. (2010). Multiple sourcinging Single-and Multi-echelon Inventory Systems.Ph.D.thesis.University of Mannheim,Mannheim,Germany.

[25] Mitchell, M. (1998). An introduction to genetic algorithm. MIT press.

[26] Nahmias, S. Production & operations analysis, 6th edition, ISBN 978-0-0771-5900-9.

[27] Peter J. R. (1987). Silhouettes: a Graphical Aid to the Interpretation and Validation of Cluster Analysis. Computational and Applied Mathematics, Vol 20, 53–65.

[28] Ramanathan, R. (2004). ABC inventory classification with multiple-criteria using weighted linear optimization. Computers and operations research, Vol 33, 695-700.

[29] Silver, E.A.,Peterson, R.(1985). Decision Systems for Inventory Management and Production Planning, 2nd edition, Wiley, New York, NY.

[30] Silver, E.A., Peterson, R. (1998). Inventory management and production planning and scheduling.

[31] Simchi-Levi, D., Kaminsky, P.,Simchi-Levi E. (2002). Designing & Managing the Supply Chain, Second Edition (66). ISBN-13:978-0071214049.

[32] Simchi-Levi, D., Zhao, Y. (2011). Performance Evaluation of Stochastic Multi-Echelon Inventory System: A survey. Advances in Operations Research, Vol 2012, article ID 126254.

[33] Srinivasan, M., Moon, Y.B. (1999). A comprehensive clustering algorithm for strategic analysis of supply chain network. Computers & Industrial Engineering, Vol 36, 615-633.

[34] Van Liempd, M. (2016). Stochastic optimization of safety stock levels in multi-echelon networks. MSc Thesis, Tilburg University.

[35] `https://en.wikipedia.org/wiki/Pareto_principle` [visited 28 April 2019].

[36] `https://en.wikipedia.org/wiki/Fractional_factorial_design` [visited 28 April 2019].

[37] `https://en.wikipedia.org/wiki/Elbow_method_(clustering)` [visited 28 April 2019].

[38] `https://scikit-learn.org/stable/modules/tree.html` [visited 28 April 2019].

[39] `https://help.sap.com/doc/saphelp_scm700_ehp02/7.0.2/en-US/4d/33d92edb9e00d3e10000000a42189b/content.htm?no_cache=true` [visited 28 April 2019].