Master's thesis

NTNU
Norwegian University of Science and Technology
Faculty of Economics and Management
Dept. of Industrial Economics and Technology Management

Anna Gravdal
Kristoffer By Vollset

Optimization and Machine Learning Methods for The Stochastic Joint Replenishment Problem under Seasonal Demand

Master's thesis in Industrial Economics and Technology Management
Supervisor: Anders Nordby Gullhav
Co-supervisor: Lars Peter Berling
July 2023

NTNU
Norwegian University of Science and Technology

Anna Gravdal
Kristoffer By Vollset

Optimization and Machine Learning Methods for The Stochastic Joint Replenishment Problem under Seasonal Demand

Master's thesis in Industrial Economics and Technology Management
Supervisor: Anders Nordby Gullhav
Co-supervisor: Lars Peter Berling
July 2023

Norwegian University of Science and Technology
Faculty of Economics and Management
Dept. of Industrial Economics and Technology Management

NTNU
Norwegian University of
Science and Technology

NTNU
Norwegian University of
Science and Technology

MASTER THESIS, 2023

Department of Industrial Economics and Technology Management

Optimization and Machine Learning Methods for The Stochastic Joint Replenishment Problem under Seasonal Demand

Authors:
Anna Gravdal
Kristoffer By Vollset

Supervisors:
Associate Prof. Anders Nordby Gullhav
Professor Lars Peter Berling

July 3, 2023

Preface

This master's thesis concludes our Master of Science in Industrial Economics and Technology Management at the Norwegian University of Science and Technology. The work in thesis is based on the work of our project theses in Vollset (2023) and Gravdal (2022).

We want to express our sincere gratitude to our supervisors Anders Nordby Gullhav and Lars Peter Berling for their guidance, interesting discussions and valuable feedback.

This thesis is written in collaboration with Visma Resolve. We would like to thank Anders Helgeland Vandvik and Thea Rønn for their guidance and insights towards realistic problems. They also provided us with real sales data to use in our computational study.

Trondheim, July 3, 2023

Anna Gravdal, Kristoffer By Vollset

Abstract

Optimizing the joint replenishment of multiple products under stochastic demand could contribute to major cost savings for businesses. This is investigated through the Stochastic Joint Replenishment Problem (SJRP).

This thesis investigates the SJRP under seasonal demand, which is relatively unexplored in current literature. The problem considers periodic review systems, meaning order placement is only possible at certain points in time. The objective is to minimize the total costs, composed of setup, holding and shortage costs while ensuring that a customer-specified service level is met.

We develop two approaches for solving the SJRP under seasonal demand: A Mixed Integer Programming (MIP) approach and a deep Reinforcement Learning (RL) approach. A comparative study, investigating the strengths and weaknesses of both approaches is conducted. Both methods are tested on the same test instances, which are generated based on real sales data.

Our MIP approach uses forecasting to predict future demand and utilize safety stocks to account for the uncertainty in the forecasts. An analysis of which forecasting method produces the most accurate forecasts is conducted. As a result of this analysis, we adopted the Holt-Winters forecasting method due to its superior accuracy. Many studies on inventory control assume that demand distribution is already known, which is a simplification. Other studies adopt simple forecasting approaches such as Auto-Regression or naïve methods, without conducting comparative analyses to identify the most effective method. Our results show that incorporating the Holt-Winters instead of a naïve method significantly reduces costs.

The RL approach developed employs a state-of-the-art Deep Deterministic Policy Gradient (DDPG) algorithm to produce robust ordering decisions. A key feature of our RL approach is the pre-training of the neural network using supervised learning. The neural network undergoes training based on state-action pairs derived from the MIP model, which is applied under deterministic demand conditions and without the constraint of safety stock. This pre-training enables the agent to learn more proficiently within the stochastic RL environment.

The results in this thesis indicate that the RL approach is superior to the MIP in terms of

total costs for two-product instances, except for one test instance with smooth demand where it performs equivalently well. This represents an advancement compared to the performance of RL solutions in existing literature addressing the SJRP. However, in the case of a four-product instance, the MIP approach supersedes the RL approach. This is mainly due to the exponentially growing state and action spaces, making the exploration of promising states for the RL agent more challenging.

Our main contributions are thus the development of a MIP approach and an RL approach designed to solve the SJRP with seasonal demand. In addition, our incorporation of a forecasting analysis to obtain a proper forecasting method represents a unique contribution within the context of SJRP literature.

Sammendrag

Optimering av felles bestilling av flere produkter under stokastisk etterspørsel kan bidra til betydelige kostnadsbesparelser for bedrifter. Dette undersøkes gjennom det stokastiske felles bestillingsproblemet (Stochastic Joint Replenishment Problem - SJRP).

Denne masteroppgaven undersøker SJRP under sesongvarierende etterspørsel, noe som er relativt uutforsket i nåværende litteratur. Problemet tar hensyn til periodiske bestillingssystemer, som betyr at bestillinger kun kan gjøres på bestemte tidspunkter. Målet er å minimere de totale kostnadene, bestående at oppsetts-, lager- og produktmangelkostnader, samtidig som et kundespesifisert servicenivå opprettholdes.

Vi utvikler to metoder for å løse SJRP med sesongvarierende etterspørsel: En blandet heltallsprogrammering (Mixed Integer Programming - MIP) metode og en dyp forsterkende læring (Reinforcement Learning- RL) metode. En komparativ studie som undersøker styrker og svakheter ved begge metodene, blir utført. Begge metodene testes på de samme testinstansene, som er generert basert på virkelige salgsdata.

MIP-metoden vår bruker prediksjoner for å forutsi fremtidig etterspørsel og benytter sikkerhetslagre for å ta hensyn til usikkerheten i disse prediksjonene. En analyse av hvilken metode som gir de mest nøyaktige prediksjonene er gjennomført. Som et resultat av denne analysen, brukte vi Holt-Winters metode til å utføre prediksjonene på grunn av dens overlegne presisjon. Mange studier innen lagerstyring antar at etterspørselsfordelingen allerede er kjent, noe som er en forenkling. Andre studier bruker enkle prediksjonsmetoder som auto-regressive (Auto-Regressive - AR) eller naïve metoder, uten å utføre komparative analyser for å identifisere den mest effektive metoden. Resultatene våre demonstrerer at bruken av Holt-Winters metode i stedet for en naïv metode reduserer kostnadene betydelig.

RL-metoden som er utviklet benytter en dyp deterministisk policygradient (Deep Deterministic Policy Gradient - DDPG) algoritme for å produsere robusere bestillingsbeslutninger. Et sentralt trekk ved vår RL-metode er forhåndstrening av det nevrale nettverket ved hjelp av overvåket læring (supervised learning). Det nevrale nettverket trenes basert på tilstand-handling (state-

action) par funnet ved hjelp av MIP-modellen, som brukes under deterministisk etterspørsel og uten begrensningen av sikkerhetslager. Denne forhåndstreningen gjør det mulig for RL-agenten å lære mer effektivt og bedre i det stokastiske RL-miljøet.

Resultatene i denne oppgaven indikerer at RL-metoden er signifikant bedre enn MIP-metoden når det gjelder totale kostnader for testinstanser bestående av to produkter, med unntak av en testinstans med jevn etterspørsel, hvor den presterer like bra. Dette representerer et fremskritt i forhold til RL-metoder i eksisterende litteratur som addresserer SJRP. Imidlertid, når problemstørrelsen økes til fire produkter er MIP-metoden signifikant bedre enn RL-metoden. Dette skyldes hovedsakelig de eksponensielt voksende tilstands- og handlingsrommene, noe som gjør utforskringen av lovende tilstander for RL-agenten mer utfordrende.

Våre hovedbidrag er dermed utviklingen av en MIP-metode og en RL-metode som er designet for å løse SJRP med sesongvarierende etterspørsel. I tillegg representerer vår analyse av prediksjonsmetoder for å velge en passenede prediksjonsmetode et unikt bidrag innenfor konteksten av SJRP-litteratur.

Contents

List of Figures

List of Tables

Abbreviations

AI	=	Artificial Intelligence
ANN	=	Artificial Neural Network
DDPG	=	Deep Deterministic Policy Gradient
DNN	=	Deep Neural Network
DP	=	Dynamic Programming
DQN	=	Deep Q-Network
EOQ	=	Economic Order Quantity
EWMA	=	Exponentially Weighted Moving Average
JRP	=	Joint Replenishment Problem
MAE	=	Mean Absolute Error
MIP	=	Mixed Integer Programming
RL	=	Reinforcement Learning
RMSE	=	Root Mean Square Error
SARIMA	=	Seasonal AutoRegressive Integrated Moving Average
SCM	=	Supply Chain Management
SJRP	=	Stochastic Joint Replenishment Problem

Chapter 1

Introduction

Effective inventory control is crucial for businesses to minimize costs. An important aspect within this domain is the Joint Replenishment Problem (JRP), which has been widely explored in the literature (Bastos et al., 2017). The JRP seeks to determine the optimal reorder points and quantities to minimize total costs. Alternatively, the focus can be shifted to maximizing the service level under the constraint of a predetermined budget for the associated costs. Both these approaches require insightful decision-making to balance cost efficiency and service level, particularly given the uncertainties of demand. By jointly considering products there is a possibility to share the fixed replenishment costs and thereby reduce the total costs (Goyal and Satir, 1989). This is the standard practice for most real-life procurement processes, as opposed to ordering products individually (Goyal and Satir, 1989).

This thesis investigates the Stochastic JRP (SJRP) with seasonal demand and develops two approaches to solve this problem: A deterministic mathematical Mixed Integer Programming (MIP) model and a deep Reinforcement Learning (RL) approach. The MIP formulation serves as a benchmark method for comparison. In the context of inventory control, the term stochastic refers to the consideration of uncertainty in demand. We conduct a computational study of the two approaches to test and compare their applicability to the SJRP. Both methods are tested using simulations based on real sales data.

The work in this thesis is carried out in dialogue with Visma, a leading provider of business software and services in Europe (Visma, 2018). Visma has offered valuable insights during the problem formulation phase, which helped ensure the problem's relevance and applicability in a realistic business context. They also provided data for our computational study. Vismas' current inventory control systems consider products individually, which is often unrealistic in practical settings. Joint consideration of products can lead to more accurate, efficient methods and further savings. The discussions with Visma led to the conclusion that the SJRP is an interesting problem to tackle. The solution methods developed in this thesis could be utilized by companies like Visma to enhance inventory optimization and reduce daily costs.

The SJRP investigated in this thesis is periodic, meaning that order placement is only possible at specific periods in time. The objective of this problem is to minimize the total costs, consisting of setup, holding, and shortage costs while ensuring that a specific service level is met.

The JRP is acknowledged to be NP-hard, even under deterministic demand (Bastos et al., 2017), making exact solutions challenging due to its combinatorial nature. Because savings can be created by purchasing items together, straightforward approaches, such as the Economic Order Quantity (EOQ) formula commonly used when considering single items or The Wagner-Whitin algorithm, are no longer applicable. Furthermore, common methods for the JRP, like the RAND method for the deterministic problem or can-order policies and P(s, S) policies for the stationary and stochastic problem, are not suitable because of the seasonal demand present in this thesis. Therefore, heuristic models and solution approaches are preferred to find near-optimal solutions, especially when handling large instances. Effective solutions may lead to large potential savings compared to solving problems for each product separately (Axsäter, 2015).

The MIP model developed in this thesis formulates the SJRP as a mathematical deterministic optimization problem that handles uncertainty in demand through service constraints using safety stocks. It aims to find the optimal actions in each time period based on updated demand information and inventory levels. This method is an extension of the previous work in Gravdal (2022), which used a similar MIP formulation, but under deterministic demand. In this thesis, the demand is assumed to be stochastic and seasonal. The MIP formulation tries to take uncertainty into account by looking ahead. It could be considered as a deterministic lookahead method. The demand is forecast using appropriate forecasting methods which account for seasonality.

As part of finding an efficient solution for the MIP approach, an initial analysis of relevant forecasting methods is conducted. The primary objective to is to minimize forecast errors and thus the uncertainty in the inventory control methods. Notably, this analysis distinguishes our research as many previous studies on inventory control simplify the problem by assuming that the demand distribution is already known. While some studies employ basic forecasting approaches like Auto-Regression or naïve methods, they lack comparative analyses to determine the most effective method. Our analysis showed that the Holt-Winters outperformed the other methods, and was therefore utilized in the MIP approach.

The RL approach utilizes a Deep Deterministic Policy Gradient (DDPG) algorithm to train the agent toward providing robust ordering decisions. With its use of deep neural networks (DNN) to approximate an optimal policy, the DDPG is able to capture the complexity of demand patterns.

A key aspect of our RL solution approach is the pre-training of the neural network using supervised learning. The network is trained on states and action pairs provided by the MIP model used on deterministic demand and without safety stock constraint. As will be demonstrated in our computational study, this pre-training stage is instrumental for the agent to learn pro-

ficiently within the stochastic RL environment. Introducing this step incentivizes the agent to explore more promising states and actions right from the start, enabling it to find more efficient solutions.

The distinct characteristics of the two solution approaches make them compelling subjects for comparison. The MIP approach is reliant on receiving precise forecasts through an effective forecasting method, as it depends on these forecasts for its computations. In contrast, the RL approach is trained on a series of many similar situations generated based on the real data, and can therefore be used independently of a forecasting method. This difference in dependency on forecasting techniques forms a significant contrast between the two methods.

The results of this thesis demonstrate that the developed RL approach is superior to the developed MIP approach in terms of total costs when applied to test instances consisting of two products. The achieved service level is however generally higher for the MIP model. When increasing the problem size to four products, the MIP approach outperforms the RL approach, as a result of the growing state and action spaces.

The academic contribution of this thesis is thus the development of a MIP approach and an RL approach for solving the SJRP under seasonal demand, considering the specific requirement and constraints faced by Visma and its customers. The MIP model developed differs from what is found in current literature. Furthermore, the forecasting analysis performed to find an appropriate forecasting method is also unique in the SJRP literature. The RL approach developed solves a less restricted problem than what has been done previously. Additionally, the combination of supervised learning and RL in this context is an innovative combination that has not been observed in the current literature.

This thesis is structured in the following way: Chapter 2 describes the relevant theory needed to understand the concepts and terminology used in the following chapters, while a literature review focusing on the JRP and researched solution methods for this problem is presented in Chapter 3. Chapter 4 gives a detailed description of the scope and the problem. The solution methods are then presented in Chapter 5. Chapter 6 describes the implementation of the methods, how the simulation is performed, and how the test instances are generated and determined. The tuning processes for enhancing the performance of the two approaches are described in Chapter 7. Chapter 8 presents the results of the computational study and compares the results from the two developed approaches. Finally, opportunities for future research are outlined in Chapter 9, and concluding remarks are given in Chapter 10.

Chapter 2

Relevant Theory

This chapter introduces theory relevant to understand the underlying concepts and methods of this thesis. Section 2.1 provides an introduction to forecasting. Section 2.2 introduces relevant terms and concepts of inventory control. Last, Section 2.3 gives a brief introduction to Reinforcement Learning (RL).

Vollset (2023) and Gravdal (2022) were written as part of the work leading up to this master thesis, and some sections of this thesis are therefore based on these. Section 2.1 contains some of the work presented in Vollset (2023), while Section 2.2 contains some of the work presented in Gravdal (2022).

2.1 An Introduction to Forecasting

In this section, we provide an overview of forecasting methods, discuss the importance of precise forecasts, and explore the categorization of demand and the selection of appropriate demand distributions.

Many studies on inventory control assume that demand data is already known, rather than obtained using a forecasting method (Tiacci and Saetta, 2009). Additionally, it is often assumed that the demand is adequately modeled (Tiacci and Saetta, 2009). This is, however, not always the case.

Forecasts are usually utilized as input when deciding upon how much and when to make replenishments (Thomopoulos and Thomopoulos, 2015). Precise forecasts are crucial to prevent excessive inventory and minimize costs (Kot and Grondys, 2011). It is, therefore, important to utilize appropriate forecasting methods that yield the most accurate forecasts.

2.1.1 Demand Categorization and Distribution

Categorization of demand is a way to sort products by their respective demand patterns and it facilitates the selection of a forecasting method (Syntetos et al., 2005b). Thus, before deciding upon which forecasting method to utilize, one should categorize the products by demand.

A traditional approach in the literature has been to more or less arbitrarily categorize demand patterns and then proceed to select an estimation procedure (Syntetos et al., 2005b). Syntetos et al. (2005b) suggest a more thorough process, which is to categorize the products based on two parameters, namely the inter-demand interval, denoted p, and the squared coefficient of variation of demand size denoted CV^2. Their approach is to compare forecasting methods directly based on the mean squared error and to use the quantified errors to define regions of superior performance. These regions, which are based on the inter-demand interval and the CV^2, are then used to categorize demand. Figure 2.1 shows the demand categories suggested by Syntetos et al. (2005b) and their respective cut-off values.

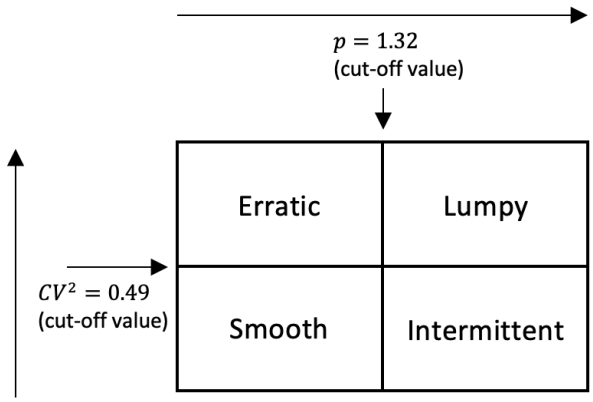


Figure 2.1: Demand categories with cut-off values.

Another property that should be decided upon prior to forecasting demand is the demand distribution. Silver et al. (1998) assume that the period demand is normally distributed for high-demand products. This is a common assumption in the literature (Huang, 2009). However, there are some situations in which the normality assumption could be less suitable. For instance, in computer simulation where one could risk randomly getting negative demand (Huang, 2009). Products with low demand are often assumed to follow a Poisson process (Cattani et al., 2011), which also mitigates the risk of negative demand. However, a normal approximation may reduce the computations substantially according to Axsäter (2013).

An important factor to be aware of is that real demands are integers and thus will not be correctly modelled by a continuous distribution. This is more serious for low demand (Axsäter, 2013). Other authors such as Willemain et al. (1994) and Huang (2009) assume log-normal distribution, which mitigates the non-negative constraint and provides right skewing.

In cases where the variance σ^2 of the demand is larger than the expected value μ ($\frac{\sigma^2}{\mu} > 1$) the

demand is sometimes approximated using a negative binomial distribution, as done in Berling and Marklund (2013).

2.1.2 Forecasting of Demand

Forecasting time series data is the process of predicting future values that a series is likely to take, based on historical data (Brooks, 2019). Forecasting finds utility in various applications, including predicting future weather, estimating future returns of a portfolio, or projecting future demand for specific products.

A distinction is made between in-sample and out-of-sample forecasts. In-sample forecasts utilize the same data set for both estimating the model's parameters and generating predictions. For this reason, in-sample forecasts are expected to be quite accurate (Brooks, 2019). On the other hand, out-of-sample forecasts use an independent data set, not involved in parameter estimation, for making projections. This is to better evaluate the model´s performance and generalizability to unseen data (Brooks, 2019).

A forecast could be performed using one-step-ahead forecasts or multi-step-ahead forecasts. One-step-ahead forecasts predict the immediate next observation, while multi-step-ahead forecasts project several steps into the future (Brooks, 2019). The choice of how many periods to predict is based on the desired forecasting horizon, and typically the uncertainty of a forecast increases with the horizon (Brooks, 2019). Evaluating the model's accuracy across varying forecasting horizons can be challenging. To address this, a recursive or rolling window approach can be employed, generating a series of forecasts for a specified number of steps ahead. Recursive forecasting models have a fixed initial estimation date and add observations to the estimation period incrementally. In contrast, rolling window models keep the in-sample period length constant, with both start and end dates advancing by one observation at a time (Brooks, 2019).

The following subsections discuss various time series forecasting techniques and their relevance when predicting seasonal demand.

Moving Average

The Moving Average (MA) is commonly used in practice for forecasting future values in time series. It is based on intuitive logic where the future values are forecast by simply taking the average of the N latest observation. The forecast is updated as soon as we have another observation.

Let s_t denote the MA estimate at time t, and let x_t denote the observed value at time t. The MA estimate s_t can be calculated as:

$$s_t = \frac{1}{N} \sum_{i=0}^{N-1} x_{t-i} \tag{2.1}$$

where N represents the number of periods in the MA window.

Exponentially Weighted Moving Average

Exponentially Weighted Moving Average (EWMA), often called exponential smoothing, is widely used in time-series forecasting because it effectively captures patterns in the data while placing greater emphasis on more recent observations (Carnot et al., 2005). Instead of giving all recent observations used for the forecast the same weight as in Equation (2.1), the weights now decay exponentially as the observation gets older. Let x_t denote the point estimates at time t. The exponential smoothing estimate s_t would in accordance with this method be estimated as:

$$s_t = \lambda x_t + (1 - \lambda) s_{t-1} \tag{2.2}$$

The smoothing parameter λ is the only parameter one needs to decide upon. Its value ranges between 0 and 1 and determines the rate at which the weights decay. It represents how important the observation at time t is compared to previous forecasts. The weights of the previous observations decay exponentially with time. The larger value of λ used, the more weight is put on recent observations. The same argument can be made in regards to the rolling window parameter N in the simple MA method described in Section 2.1.2. A smaller value of λ results in forecasts that are smoother and less influenced by noise.

EWMA is a useful method when limited knowledge is available regarding the factors driving the sample estimates and when the series trend appears to change rapidly (Carnot et al., 2005). The effect on the value of a single observation decays exponentially with time since the weights decay exponentially.

One disadvantage of EWMA is that it is unclear which value to use for λ. A solution is often to test various values for λ and evaluate which value yields more robust forecasts. Another limitation of the EWMA model is that it does not consider the long-run value as the prediction horizon increases. The model assumes that the one-period forecasts are serially uncorrelated, leading to an absence of mean reversion in the model (Brooks, 2019).

Holt-Winters Method

While the EWMA method is quite effective at capturing patterns in time-series forecasting, it does not account for seasonality or trend. The Holt-Winters method is an extension of

exponential smoothing, which addresses this limitation by incorporating seasonality and trend into the forecasting model.

Initially, Holt (1957) extended the simple exponential smoothing method by incorporating a trend component, resulting in the double exponential smoothing model. Later, Winters (1960) further enhanced the model by adding a seasonal component, thus creating the triple exponential smoothing method, which combines the ideas of both Holt and Winters.

The mathematical formula for the additive Holt-Winters method is given by Hyndman and Athanasopoulos (2018) as:

$$\begin{aligned} \hat{y}_{t+h|t} &= l_t + hb_t + s_{t-m(k+1)} \\ l_t &= \alpha(y_t - s_{t-m}) + (1-\alpha)(l_{t-1} + b_{t-1}) \\ b_t &= \beta^*(l_t - l_{t-1}) + (1-\beta^*)b_{t-1} \\ s_t &= \gamma(y_t - l_{t-1} - b_{t-1}) + (1-\gamma)s_{t-m} \end{aligned} \tag{2.3}$$

where

- l_t represents the smoothed level estimate at time t.
- b_t is the smoothed trend estimate at time t.
- s_t is the seasonal factor estimate at time t.
- $\hat{y}_{t+h|t}$ is the forecast for time $t+h$.
- y_t is the actual observation at time t.
- m is the seasonal period.
- h is the number of periods ahead for the forecast, and k is the integer part of $(h-1)/m$, which ensures that the seasonal indices used for forecasting come from the final year of the sample.
- The smoothing parameters are α, β, and γ, which all range from 0 to 1 (Hyndman and Athanasopoulos, 2018).

Equation (2.3) consists of three components: the average component, the trend component, and the seasonal component. The three components are expressed as exponential smoothing equations, which is why the Holt-Winters method is also referred to as the triple exponential smoothing method. Introducing the season- and trend equation allows for more complex patterns in the time-series data than the simple EWMA, while maintaining the benefits of exponential smoothing.

Autoregressive Moving Average

AutoRegressive Moving Average (ARMA) is commonly used in time series forecasting. The ARMA method combines two components, an Autoregressive (AR) part, and an MA part.

The MA part models the current error term as a linear combination of past error terms, which is assumed to follow a white noise process (Brooks, 2019). Let y_t denote the current value of the time series at time t, and let ε_t be the error term at time t. The MA(q) model can then be represented by the following equation:

$$y_t = \mu + \varepsilon_t + \theta_1\varepsilon_{t-1} + \theta_2\varepsilon_{t-2} + \ldots + \theta_q\varepsilon_{t-q} \tag{2.4}$$

where $\theta_1, \theta_2, \ldots, \theta_q$ is the parameters of the model and μ is the mean of the time series.

By defining $B^i y_t$ to indicate that y_t is lagged i times, we can rewrite Equation (2.4) using the lag operator B as follows:

$$y_t = \mu + \theta(B) * \varepsilon_t \tag{2.5}$$

where $\theta(B) = 1 + \theta_1 B + \theta_2 B^2 + ... + \theta_q B^q$

The parameter q represents the number of lagged forecast errors that are used to forecast the current value in the time series. It is usually set to be the lag at which the autocorrelation between the observation and its preceding lags drops to zero (Brooks, 2019). The AR part of the ARMA method models the current time series value as a linear combination of past values. The autoregressive model of order p, AR(p), can be written as:

$$y_t = C + \phi_1 y_{t-1} + \phi_2 y_{t-2} + ... + \phi_p y_{t-p} + \varepsilon_t \tag{2.6}$$

where $\phi_1, \phi_2, \ldots, \phi_q$ is the parameters of the model and C is a constant. Using the lag operator notation B, it can be expressed as:

$$\phi(B) y_t = C + \varepsilon_t \tag{2.7}$$

where $\phi(B) = 1 - \phi_1 B + \phi_2 B^2 + ... + \phi_q B^q$

For an AR(p) process, there is a partial autocorrelation between the current observation y_t and its lagged values y_{t-s} for all lags $s \leq p$, while no partial correlation exists for lags $s > p$ (Brooks, 2019). This partial correlation is defined as the direct correlation between y_t and y_{t-s}, after

removing the effects of all intermediary lags y_{t-s+1}, y_{t-s+2}, ... , y_{t-1} (Brooks, 2019).

The AR(p) model's stationarity is a crucial feature that can be evaluated by examining the roots of its characteristic equation:

$$1 - \phi_1 z^1 - \phi_2 z^2 - ... - \phi_p z^p = 0 \tag{2.8}$$

An AR(p) model is stationary if all roots of the equation lie outside the unit circle (Brooks, 2019).

This property of stationarity in an AR model is beneficial for a variety of reasons. An important concern associated with nonstationarity is the persistent, non-diminishing impact that past error terms may have on the current value of the dependent variable, y_t. This means that an error from many time steps ago can still significantly impact the forecast for the current time step, regardless of the elapsed time. This behavior could lead to unrealistic or unstable forecasts (Brooks, 2019).

An ARMA(p, q) model combines an MA(q) and AR(p) model. This model is useful if the time series value y_t is linearly dependent both on its own previous lags and its white noise error terms (Box et al., 2015). This model is expressed as:

$$\phi(B)y_t = C + \theta(B)\varepsilon_t \tag{2.9}$$

The model captures both abrupt changes in the data, which is handled by the MA part, and long-term trends, which is handled by the AR part.

The ARMA model assumes stationarity in the data, which makes it unsuitable for handling non-stationary data. If the data is non-stationary, differencing can be performed to transform the variables to make the data stationary. The ARMA model can then be applied to those transformed variables (Brooks, 2019). Consequently, an ARIMA(p, d, q) model is equivalent to an ARMA(p,q) model where the original variables are differenced d times (Brooks, 2019). If d= 1, a differencing is done by the following equation:

$$y'_t = y_t - y_{t-1} \tag{2.10}$$

Seasonal Autoregressive Integrated Moving Average

The SARIMA model is an extension of the ARIMA model, specifically designed to accommodate seasonal data. It does this by integrating both non-seasonal and seasonal components. The non-seasonal AR and MA components capture dependencies on past values and past forecast errors respectively, while the seasonal components do the same but at lags that are multiples of the

seasonal period S (Box et al., 2015).

For instance, the seasonal MA component, with a seasonal period S, can be expressed as:

$$y_t = \mu + \varepsilon_t + \theta_1 \varepsilon_{t-S} + \theta_2 \varepsilon_{t-2S} + \ldots + \theta_Q \varepsilon_{t-QS} \tag{2.11}$$

The same modification can similarly be applied to Equation (2.6) in order to derive the seasonal AR component.

To make the seasonal data non-stationary, a technique known as seasonal differencing is employed. Box et al. (2015) explain that for seasonal data, one would typically expect relationships to occur between the same season in successive years. Seasonal differencing is applied as follows:

$$(1 - B^s) y_t = y_t - y_s \tag{2.12}$$

In the presence of a trend, we can employ differencing (as in Equation (2.10)) to detrend the data, yielding:

$$(1 - B)(1 - B^s) y_t = y_t - y_1 - y_s - y_{s-1} \tag{2.13}$$

The Seasonal ARIMA (SARIMA) model is typically denoted as SARIMA $= \text{ARIMA}(p, d, q) \cdot (P, D, Q)S$ where q, d, and p represent the parameters of the non-seasonal ARIMA model, D indicates the order of seasonal differencing required to account for seasonal trends, while P and Q represent the orders of the seasonal AR (SAR) and MA (SMA) components respectively. S specifies the length of the seasonal cycle. The SARIMA model can be expressed as follows:

$$\Phi(B^S)\phi(B)(1 - B)^d (1 - B^S)^D y_t = C + \Theta(B^S)\theta(B)\varepsilon_t \tag{2.14}$$

This equation encapsulates all components of the SARIMA model. In particular:

- The $\Phi(B^S)$ operator represents the seasonal autoregressive part. For instance, a first order SAR(1) model can be written as $\Phi(B^S) = 1 - \Phi_1 \cdot B^S$.
- The $\phi(B)$ operator represents the non-seasonal autoregressive part. For an AR(1) model, it can be written as $\phi(B) = 1 - \phi_1 \cdot B$.
- The $(1 - B)^d (1 - B^S)^D$ term is the differencing part, which is done d times at the non-seasonal level and D times at the seasonal level to make the series stationary.
- The $\Theta(B^S)$ operator represents the seasonal moving average part. For an SMA(1) model, it can be written as $\Theta(B^S) = 1 + \Theta_1 \cdot B^S$.

- The $\theta(B)$ operator represents the non-seasonal moving average part. For an MA(1) model, it can be written as $\theta(B) = 1 + \theta_1 \cdot B$.
- Finally, ε_t is the error term and C is a constant.

With this setup, the SARIMA model can capture both trend and random fluctuation in data at both non-seasonal and seasonal frequencies (Box et al., 2015).

Croston's method

Certain items have infrequent demand occurrences, yet the quantity demanded can be relatively substantial. When using the exponential smoothing approach, the forecasts will decrease in periods with zero demand and increase in periods with positive demand. How fast it reacts depends on the λ parameter. Since demand occurs only sporadic, the exponential smoothing forecast might react too late since the demand in the next period can be zero. Croston (1972) states that using exponential smoothing to forecast sporadic demand in a control stock system almost always produces inappropriate stock levels unless the intervals between positive demand are known.

He suggests another approach to handle such situations; using separate estimates for the demand quantity and for the demand frequency (Croston, 1972). The forecast gets updated only when there is demand recorded for a certain period. When positive demand occurs, the estimates for the demand quantity and demand frequency are updated using exponential smoothing. This results in a more consistent forecast and enhances understanding of the demand structure (Axsäter, 2015).

Let x_t denote the demand for period t, and the following:

- p_t = The random number of periods that have passed since the last recorded positive demand
- $\hat{p}_t$ = the estimated mean number of periods occurring between two instances of positive demand, at the end of period t.
- $\hat{y}_t$ = The estimated mean magnitude of a positive demand, at the end of period t.
- $\hat{a}_t$ The expected demand per period at the end of period t.

If $\hat{p}_t$ and $\hat{y}_t$ are updated as follows:

$$\hat{p}_t = \begin{cases} \hat{p}_{t-1}, & \text{if } x_t = 0 \\ (1-\alpha)\hat{p}_{t-1} + \alpha p_t, & \text{otherwise} \end{cases} \tag{2.15}$$

$$\hat{y}_t = \begin{cases} \hat{y}_{t-1}, & \text{if } x_t = 0 \\ (1-\alpha)\hat{y}_{t-1} + \alpha y_t, & \text{otherwise} \end{cases} \tag{2.16}$$

where $0 \leq \alpha, \beta \leq 1$, the expected demand is given as:

$$\hat{a}_t = \frac{\hat{y}_t}{\hat{p}_t} \tag{2.17}$$

Despite its effectiveness for intermittent demand forecasting, Syntetos and Boylan (2001) argue that it possesses a positive bias. During periods of zero demand, the inter-demand intervals are not observed, leading to biased estimates. Syntetos and Boylan (2005) address this issue by multiplying the forecast of the demand per period with $1 - \alpha/2$, which yields:

$$\hat{a}_t = (1 - \frac{\alpha}{2})\frac{\hat{y}_t}{\hat{p}_t} \tag{2.18}$$

Teunter et al. (2011) introduce another modification of Croston's method. Instead of only updating the inter-demand interval only in periods where demand occurs, it updates the demand quantity each period regardless of whether there is a demand in that period or not. It also uses two different smoothing constants.

$$\hat{p}_t = \begin{cases} \hat{p}_{t-1} * \beta(-\hat{p}_{t-1}), & \text{if } x_t = 0 \\ \beta p_t + (1-\beta)\hat{p}_{t-1}, & \text{otherwise} \end{cases} \tag{2.19}$$

$$\hat{y}_t = \begin{cases} \hat{y}_{t-1}, & \text{if } x_t = 0 \\ (1-\alpha)\hat{y}_{t-1} + \alpha y_t, & \text{otherwise} \end{cases} \tag{2.20}$$

2.1.3 Calculating the Standard Deviation of Forecasting Errors

Due to the uncertainties in demand, there is often a need for safety stocks in inventory control (Axsäter, 2015). Axsäter (2015) defines safety stock as the average amount of inventory maintained to account for fluctuations in both demand and supply. To determine a proper safety stock level, it is crucial to quantify the uncertainty in the forecasts to account for possible forecasting errors (Axsäter, 2015).

A common way to describe the uncertainties in the forecasts is by the standard deviation of the forecast errors (Axsäter, 2015). Let μ be the mean of a random variable X. The standard deviation σ of X is defined as:

$$\sigma = \sqrt{(X - \mu)^2} \tag{2.21}$$

In order to incorporate σ in the determination of safety stock, it is necessary to develop a sequential updating method for σ based on new observations. This is because we do not know the variable X when forecasting the demand. A commonly employed approach to address this challenge is the use of exponential smoothing. Let $f_{t-1,t}$ be the forecast for period t at the end of period $t-1$. Although there often exists systematic errors in the forecast, $f_{t-1,t}$ can be treated as the mean at period $t-1$ (Axsäter, 2015). After receiving the actual demand d_t in period t, we can update the standard deviation of forecasting errors as follows:

$$\sigma = \lambda\sigma + (1 - \lambda)\sqrt{(d_t - f_{t-1,t})^2} \tag{2.22}$$

where λ is the smoothing parameter, as described in Section 2.1.2.

Equation (2.22) relies on the assumption that the standard deviation of forecasting errors is randomly deviating from the mean, at a slow pace. This is a common assumption in the literature (Axsäter, 2015).

When there is a limited amount of historical data per product available, the initial value of σ cannot be directly calculated. A common strategy to deal with this situation involves calculating σ as a function of the mean demand (Axsäter, 2015). This approach is based on the observation that forecast errors tend to increase with average demand. Let $\hat{d}$ denote average demand. A popular method for calculating σ can be represented as follows (Axsäter, 2015):

$$\sigma = a \cdot \hat{d}^b \tag{2.23}$$

The parameters a and b can be decided using linear regression:

$$\log \sigma = \log a + b \log \hat{d} \tag{2.24}$$

2.2 An Introduction to Inventory Control

Supply Chain Management (SCM) is defined as the control of the material flow from suppliers of raw materials to final customers (Axsäter, 2015). This is a crucial problem for most organizations across almost all sectors of the economy. Within SCM, inventory control plays an important role and has therefore been heavily researched (Jiang and Sheng, 2009). Optimal inventory control is among the most practical and challenging problems in SCM (Meisheri et al., 2022). Using scientific methods to manage the inventory control problem can give a significant competitive advantage (Axsäter, 2015). The purpose of these methods is to determine when and how much

to order (Axsäter, 2015). Doing this in an effective way can lead to improved customer service, reduced costs, and increased competitiveness for organizations.

Axsäter (2015) is along with e.g. Zipkin (2000) a commonly used reference in inventory control as it provides a wide range of both models and aspects of the inventory control problem. This is, therefore, used as a base to understand the inventory control problem and to find references to other relevant sources. There is a wide range of models and aspects that are not covered in this thesis. For the interested reader, we refer to Axsäter (2015) and the references therein.

The subsequent sections describe costs to consider in inventory control problems, the structure of inventory control problems, ordering policies, review policies, demand structures, some solution methods, and safety stocks.

2.2.1 Costs to Consider

The purpose of inventory control is to balance the inventory-related costs, and hereby minimize the expected costs per time unit. This involves balancing holding costs, setup costs, and shortage costs (Axsäter, 2015).

Holding costs usually consist of a cost per unit of time a product unit is kept as inventory and can encompass expenses like capital costs, material holding costs, storage costs, damage, insurance, and taxes (Axsäter, 2015). Setup costs are fixed costs associated with a replenishment (Axsäter, 2015). These might include costs of purchasing and filling out order forms, authorization, receiving, inspection, and handling of invoices from the supplier. These costs incur whenever an order is made. For the JRP, these costs are typically split into two parts: one major setup cost for the entire order and one minor setup cost for each item type replenished. This is explained in more detail in Section 3.2. Shortage costs arise when demand cannot be met, leading to backorders or lost sales (Axsäter, 2015). If the demand is backordered, the units demanded are delivered at a later point at a cost, while if the sale is lost, the demand is not met. These costs are difficult to estimate. Because of the difficulty in estimating shortage costs, it is common to replace them with service constraints. Service constraints are further described in Section 2.2.7.

The production purchase costs are often excluded from inventory control models, as they are not typically linked to inventory-related decisions.

2.2.2 Structure of the Problem

One important aspect of an inventory control problem is whether the system considered is single- or multi-echelon. Single-echelon systems contain one individual storage point within the supply chain, while multi-echelon systems consider several linked storage points, perhaps the entire supply chain (Ekanayake et al., 2016). This thesis focuses on a single-echelon system.

An additional aspect is whether the horizon is finite or infinite. In most problems, the real horizon is infinite (Axsäter, 2015). This is because most firms know that they will keep ordering in the same system for a long time ahead. However, one can argue that the horizon never truly is infinite. In finite-horizon problems, there is one point in time, which then is the end of the horizon, where one knows the optimal inventory level. This is typically zero. In infinite-horizon problems, on the other hand, no such knowledge exists. This is the case, for example, with the Economic Order Quantity (EOQ) formula, which is explained in Section 2.2.6.

The added restriction of a given optimal end of horizon often complicates the problem. One might for example not be able to use the simple EOQ formula for a problem with continuous demand. However, for some problems, for example, problems with time-varying demand, it simplifies the solution process. One thus often uses a more limited finite horizon for a problem as an approximation (Axsäter, 2015).

For problems with time-varying demand, it is also common to consider the problems using a periodic review rolling horizon approach (Axsäter, 2015). This means that the first period(s) are solved as deterministic problems using forecasts or customers' orders placed beforehand. After solving the problem for this time period, the decisions are implemented rolling through time. After the period, a new short-term plan is made using updated forecasts or customers' orders (Narayanan and Robinson, 2010). The real horizon is then infinite, but a finite horizon is considered an approximation.

A further aspect of an inventory control problem is whether a single item or multiple items are bought at once. In the simplest models, only one item is purchased at a time. The problem becomes more complicated when considering multiple items at once. However, this is often more realistic, because companies commonly order several items at once (Goyal and Satir, 1989). The problem of jointly replenishing items is discussed in Section 3.2.

Another aspect that is often mentioned in relation to inventory control is backorders. Backorders can be defined as the units that have been demanded but not delivered yet, meaning that they are ordered after they have been demanded (Axsäter, 2015). Backorders are mostly used in stochastic models but can also be used in deterministic models (Axsäter, 2015). As mentioned above, stochastic models either use backorders or model the demand as lost sales, if items are demanded but there is a shortage. If the customers are willing to wait, it makes sense to model the excess demand as backorders, while if customers would rather visit another store when a shortage occurs, it makes sense to model the excess demand as lost sales.

Lead time is another term that often is used in inventory control theory. Lead time is the time between ordering and delivery (Axsäter, 2015). When considering deterministic models and deterministic lead times, it is not necessary to include lead time in the inventory control model. The lead time can then be added to the original result. When assuming a stochastic lead time, however, the problem gets more complicated.

Another relevant term to define is inventory level. The inventory level is defined by Axsäter (2015) as follows:

$$Inventory\ Level = Stock\ on\ hand - Backorders \tag{2.25}$$

The stock on hand refers to the actual quantity of items physically present in the warehouse.

2.2.3 Review Policies

Axsäter (2015) describes two inventory review policies, a continuous review policy, and a periodic review policy. With continuous review systems, the inventory position is monitored continuously. An order is typically triggered when the inventory position reaches a critical level, meaning a lower level for when one needs to consider ordering. The order is delivered after a lead time, which is defined as the time from the order is placed until the ordered amount is in stock (Axsäter, 2015). A continuous review system will typically reduce the need for safety stocks, compared to periodic review systems, because the inventories are continuously monitored. The alternative to a continuous review policy is a periodic review policy. With a periodic review system, the inventory position is only considered at certain given points in time, usually with a constant time between them (Axsäter, 2015). This reduces the cost of the system since fewer resources are spent monitoring the inventory levels, but it increases the need for safety stock because inventories might reach lower levels between the reviews using this approach. In addition, it facilitates joint ordering, because several products might have reached lower levels at the time inventory levels are considered.

2.2.4 Demand Structures

An essential aspect of an inventory control model is the demand structure. One distinction that can be made, is the difference between problems with deterministic and stochastic demand. A deterministic demand means that the demand is known when doing the calculations (Axsäter, 2015). Axsäter (2015) also states that even though deterministic demand seems like an unrealistic assumption, the assumption is often quite reasonable because the demand often is fairly constant.

The demand, both deterministic and stochastic, can be constant, also referred to as stationary, or time-varying, also referred to as non-stationary or dynamic. The second is the focus of this thesis, also assuming stochastic demand. There can be several reasons for time-varying demand. For instance, some products are more requested in the winter time due to lower temperatures, or during special occasions like Christmas, which might cause an increase in demand for certain products. When dealing with time-varying demand in inventory control settings, one often assumes a finite number of time periods (Axsäter, 2015). The most common demand structure

for both deterministic and stochastic JRP models in the literature is a constant demand. This is less complex because one does not have to compute for all similar periods, but rather find a general rule, similar to the EOQ formula or a can-order policy.

In the case of stochastic demand, the demand is unknown. This is usually considered when determining reorder points (Axsäter, 2015). In this thesis, stochastic demand is assumed, and models assuming stochastic demand are therefore in focus. One last aspect of the demand that inventory models might consider is if the demand for items is correlated. This might have implications for how items are replenished. This is only relevant for problems with stochastic demand since the correlation is captured by the deterministic demand for problems with deterministic demand.

2.2.5 Ordering Policies

An ordering policy serves as a rule that determines when to place orders and how much to order. The two most common ordering policies are the (R, nQ) policy and the (s, S) policy (Axsäter, 2015). Using an (R, nQ) policy, a batch quantity of size Q, or an integer multiple of Q, is ordered when the inventory position is on or below a reorder point R (Axsäter, 2015). Using an (s, S) policy, one orders when the inventory position is on or below a lower level s and orders up to the maximum level S (Axsäter, 2015). These ordering policies are commonly applied to single-unit problems with stationary demand. For more complex problems involving joint replenishment and/or non-stationary demand, more advanced policies must be used.

2.2.6 Basic Solution Methods

In this section, some basic solution models for single-echelon models with deterministic demand and no joint replenishment are described. These are interesting because they give an overview of how simple inventory control problems are solved, which is a good stepping stone to developing exact models and heuristics for more complex problems.

Constant Demand Models - The Economic Order Quantity Formula

One of the most frequently used and most well-known inventory control models is the classical economic order quantity formula, referred to as the EOQ formula (Axsäter, 2015). This model was derived by Harris (1913). The EOQ model considers the replenishment of a single item in a single-echelon inventory system with stationary demand (Axsäter, 2015).

The assumptions made for this model are:

- Constant and continuous demand

- Constant setup costs
- Constant holding costs
- Order quantities can be non-integer
- The entire order is delivered simultaneously
- Shortages are not allowed

The following notation is used, defined as in Axsäter (2015):

- h = holding cost per unit per time unit
- A = setup cost
- d = demand per time unit
- Q = order quantity
- C = total costs per time unit

The optimal order quantity is then derived by deriving the cost expression for the problem (Axsäter, 2015).

The following expression of the cost per time unit, consisting of holding costs and setup costs, is given:

$$C = \frac{Q}{2} \cdot h + \frac{d}{Q} \cdot A \tag{2.26}$$

When differentiating the expression with respect to Q, and setting this equal to 0, one obtains:

$$\frac{dC}{dQ} = \frac{h}{2} - \frac{d}{Q^2} \cdot A = 0 \tag{2.27}$$

Differentiating this expression again provides the following:

$$\frac{d^2C}{dQ^2} = \frac{2d}{Q^3} \cdot A = 0 \tag{2.28}$$

For $Q > 0$ this is a convex function, and the cost is thus minimized.

Solving Equation (2.27) for Q, the following expression for Q^*, the economic order quantity, is obtained:

$$Q^* = \sqrt{\frac{2Ad}{h}} \tag{2.29}$$

Even though this model is fairly simple, it has had and still has a huge amount of practical applications (Axsäter, 2015). It can further be extended to take quantity discounts and backorders into account. When using this model, the solution obtained has equal setup cost and holding cost per time unit (Axsäter, 2015). This result is an important result, which is especially interesting when introducing heuristics.

Time-Varying Demand Models - The Wagner-Whitin Algorithm

When considering a single-echelon inventory system with time-varying demand, the common approach is to use dynamic programming (DP) (Axsäter, 2015). Wagner and Whitin (1958) were the first to suggest this. Their solution approach, denoted the Wagner-Whitin algorithm, is still widely used (Axsäter, 2015). The algorithm is, like most other algorithms with time-varying demand, applied to a finite horizon problem.

According to Axsäter (2015), an optimal solution obtained using The Wagner-Whitin algorithm, as well as other algorithms for systems with time-varying demand, has two important properties:

- Property 1: Each order must cover an integer number of consecutive periods' demand.
- Property 2: It should not happen for any time period that the holding costs exceed the ordering cost.

Time-Varying Demand Models - The Silver-Meal Heuristic

The Silver-Meal Heuristic is one of the best and most well-known methods for the single-echelon, single-item systems with time-varying demand (Silver and Meal, 1973). Even though Wagner-Whitin and other exact methods usually are computationally feasible for most instances, methods like the Silver-Meal heuristic are more common in practice. The Silver-Meal heuristic is a sequential method, in which the main idea is to choose to order when the first time average costs per period increase (Axsäter, 2015).

The cost increase is commonly only about 1 to 2% compared to an exact method, however, the relative error can become arbitrarily large (Axsäter, 2015). An advantage of this heuristic is that it can easily be extended to include quantity discounts and other extensions (Axsäter, 2015).

Heuristics Balancing Holding and Ordering Costs

Axsäter (2015) presents a heuristic that balances the holding and ordering costs of the problem. In the EOQ formula described earlier, the holding costs and ordering costs are equal in the optimal solution (Axsäter, 2015). Even though this is not true for the problem with time-varying demand, it is reasonable to expect that these are of similar size. This is the basis for a heuristic, which is a simplification of the method presented by DeMatteis and Mendoza (1968). The idea behind this heuristic is to order when the holding costs exceed the ordering costs (Axsäter, 2015), in the same way as described in the paragraph above. When to order next is determined in the same way, using *n+1* as the starting period. This heuristic seems to generally give poorer solutions than Silver-Meal, but the relative error is bounded by 100% (Axsäter, 2015). When the horizon, in reality, is infinite, even exact methods like Wagner-Within provide an approximation (Axsäter, 2015). In this case, one might see that heuristics are less sensitive to the chosen length of the horizon and therefore make good approximations. They might even outperform the exact method (Axsäter, 2015).

2.2.7 Safety Stocks

As mentioned previously, the proper safety stock can be determined by either a service constraint or a backorder cost. Generally, it is easier to determine a proper service level (Axsäter, 2015). Axsäter (2015) considers three definitions of service levels:

- S_1 represents the probability of no stockout per order cycle
- S_2 denotes the proportion of demand that can be immediately satisfied from stock on hand, referred to as "fill rate"
- S_3 indicates the proportion of time in which stock on hand is positive, referred to as "ready rate"

Axsäter (2015) further explains that service levels can be defined in several other ways as well. And often, it is suitable to not use the same service level for all items (Axsäter, 2015). In general, the service level used should be determined using the customers' expectations, the shortage costs, and the costs of maintaining high service levels (Axsäter, 2015).

The safety stock can then be determined using the specified service levels. Similar to service levels, there are many different methods used to determine the safety stocks. Gonçalves et al. (2020) present a review of papers published in the time frame from 1977 to 2019 on safety stock determination. According to Gonçalves et al. (2020), the appropriate sizes of safety stocks are affected by six components: service level, lead time, demand volatility, selected order policy, component commonality, and holding costs. Safety stocks should be small enough to minimize

holding costs while being large enough to ensure high service levels are maintained (Gonçalves et al., 2020).

When assuming that the lead time $L > 0$ is normally distributed with a mean μL and the standard deviation $\sigma_D\sqrt{L}$ and using a S_1 service constraint with target service level of α, the easiest approach for determining the correct safety stock is to use:

$$Safety\ stock = \phi^{-1}(\alpha)\sigma_D\sqrt{L} \tag{2.30}$$

where $\phi(\cdot)$ represents the standard normal cumulative distribution function, $\phi^{-1}(\alpha)$ represents the safety factor, and σ_D represents the standard deviation of demand per time unit (Gonçalves et al., 2020).

Another option is to determine the safety stocks based on the target service levels and the lead time demand forecast errors from the past (Gonçalves et al., 2020). This is a generalization of the previous measure, that can also handle correlation in demand. The following expression can then be used:

$$Safety\ stock = \phi^{-1}(\alpha)\sigma_F \tag{2.31}$$

where σ_F represents the standard deviation of the part forecasting errors for the lead time L (Gonçalves et al., 2020).

When the demand follows another distribution than the normal distribution, one can use a non-parametric forecasting approach, using the following expression:

$$Safety\ stock = Q_L(\alpha) \tag{2.32}$$

where $Q_L(\alpha)$ represents the lead time forecast error quantile at service level α (Gonçalves et al., 2020).

2.3 An Introduction to Reinforcement Learning

RL has seen significant advancements in recent decades and continues to be a key area of research in Artificial Intelligence (AI) (Sutton and Barto, 2018). It consists of a diverse collection of methods, several of which have resulted in major breakthroughs. Most recently, OpenAI implemented RL in the training of ChatGPT-4, which was a significant breakthrough in the field of AI. The model learned to generate more accurate responses over time by using feedback from its interactions with humans.

Deep RL includes neural networks and is employed when the environment is complex and dynamic, and there is no simple rule-based solution to the problem. Its utility spans a broad array of applications, among them inventory control (Boute et al., 2022). It is claimed that deep RL is promising to solve inventory control problems. However, there is a need for extensive hyperparameter tuning (Vanvuchelen et al., 2020).

Typically, problems solved by RL involve these three key characteristics: Active interaction between a decision-making agent and its environment, the agent seeking to achieve a goal, and uncertainty about the environment the agent operates in (Sutton and Barto, 2018).

An RL agent learns knowledge and behaviour through trial-and-error interactions with a dynamic environment (Kaelbling et al., 1996). The agent takes actions that modify the environment's state and receives occasional feedback from the environment in terms of a reward or penalty based on the achieved state and the agents' objective (Sutton and Barto, 2018). The feedback it receives is only an indication of whether the actions taken are good or bad, and the agent uses this indication to adjust its behaviour to maximize its cumulative reward over time (Sewak, 2019). Often, the effects of actions cannot be entirely predicted, and the agent, therefore, needs to regularly monitor the environment frequently and respond appropriately (Sutton and Barto, 2018).

During the training of the RL agent, the state represents a context presented by the environment to the agent (Sutton and Barto, 2018). The state is typically a simplified representation of the complex real-world scenarios that the agent will face in deployment, which include uncertainties and other challenges (Sewak, 2019).

2.3.1 Markov Decision Processes

RL processes employ the structured framework of Markov Decision Processes (MDPs) (Sutton and Barto, 2018). MDPs provide a mathematical formulation of RL problems, where learning is done by interaction with the environment to achieve a goal (Sutton and Barto, 2018). The agent of the problem is the decision maker and learner, while the environment is the thing it interacts with (Sutton and Barto, 2018). The interaction between the agent and the environment is illustrated in Figure 2.2.

MDPs represent a model of stochastic sequential decision-making, where actions have an impact on both the immediate and future rewards (Sutton and Barto, 2018). There is thus a trade-off between immediate and delayed reward. The MDP contains a set of actions, $\mathcal{A}$, a set of states $\mathcal{S}$, a set of rewards $\mathcal{R}$ and a set of discrete time steps $\mathcal{T}$. The agent makes an action A_t in each time step t based on the state at time step t, denoted S_t. The agent then receives a reward R_{t+1} next time step $t+1$, partly based on the previous action A_t. The new state is then S_{t+1}.

In a finite MIP, the set of actions, states and rewards are finite. One can calculate the probability

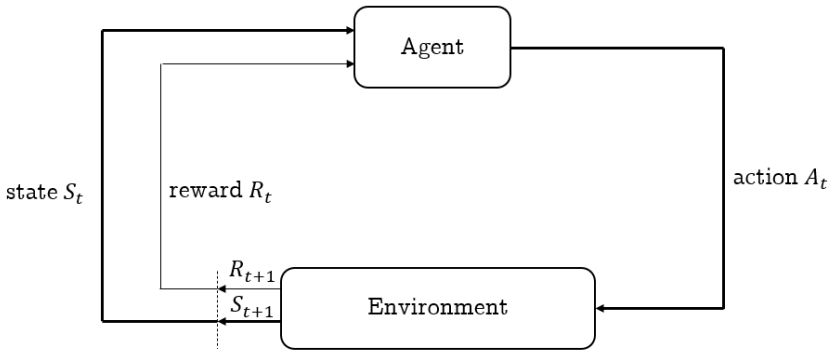


Figure 2.2: Illustration of interaction between agent and environment. Illustration inspired by Sutton and Barto (2018).

of the random variables s' and r occurring at time step t, given the values of state and action of the previous state as follows:

$$p(s', r|s, a) = \Pr\{S_t = s', R_t = r|S_{t-1} = s, A_{t-1} = a\} \tag{2.33}$$

for all $s', s \in \mathcal{S}$, $r \in \mathcal{R}$ and $a \in \mathcal{A}$ (Sutton and Barto, 2018). Thus, the probability of transitioning to any particular state depends solely on the current state and the decision made, not on the sequence of events that preceded it. This is called the Markov Property. This principle is applied to a succession of stochastic events. MDPs are applied in many areas, especially in queuing and inventory control (Puterman, 1990).

2.3.2 Elements of an RL System

In addition to the agent and the environment it operates in, four main subelements of an RL system can be identified: a *policy*, a *reward signal*, a *value function*, and, optionally, a *model* of the environment (Sutton and Barto, 2018).

A policy can be defined as a mapping from perceived states of the environment to probabilities of selecting each possible action in those states. The policy must balance the need to exploit existing knowledge for immediate rewards and the necessity to explore improved future actions. The agent needs to explore a diverse range of actions and gradually prioritize those that appear to be the best (Sutton and Barto, 2018).

Policy in RL can be divided into two categories: on-policy and off-policy. When using on-policy, the agent consistently explores while aiming to discover the optimal policy (Sutton and Barto, 2018). The off-policy approach uses two policies, namely a behaviour policy and a target policy.

The behaviour policy is the exploratory policy used to generate actions, while the target policy is the policy that we aim to improve and optimize (Sutton and Barto, 2018). Common off-policy learning methods are Q-learning and R-learning, while common on-policy methods are Sarsa and actor-critic methods (Sutton and Barto, 2018).

The reward signal describes the goal of the problem (Sutton and Barto, 2018). In most RL systems, the environment gives a reward to the agent at each time step, indicating how good the current state and action were, and the goal of the agent is to maximize the cumulative reward (Sutton and Barto, 2018). To define this formally, let R_{t+n} denote the reward received at time $t+n$ and G_t be the cumulative reward from time t until the termination time T. The cumulative reward is decided as a function of the reward sequence, and the simplest way to define G_t is by:

$$G_t = R_{t+1} + R_{t+2} + R_{t+3} ... + R_T \tag{2.34}$$

Future rewards are normally discounted by a factor, γ. Due to the stochastic nature of many environments, the farther in the future a reward is, the less certain we are about its value. By discounting rewards, immediate rewards are explicitly preferred over distant, uncertain ones. Also, many problems do not have a terminal state, but an infinite time horizon. Using Equation (2.34) would be problematic since the cumulative reward G_t could become infinite. Discounting future rewards ensures that the sum of the rewards will converge to a finite value (Sutton and Barto, 2018). The cumulative discounted reward becomes:

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} ... = \sum_{n=0}^{\infty} \gamma^n R_{t+n+1} = R_{t+1} + \gamma G_{t+1} \tag{2.35}$$

where γ is the discount rate with a value in the range $0 \leq \gamma \leq 1$.

The value function defines what is good in the long run and yields the expected cumulative reward G_t (Sutton and Barto, 2018). The state-value function $V_\pi(s)$ is the value function for a state s following policy π and is given by:

$$V_\pi(s) = \mathbb{E}[G_t | S_t = s] = \left[\sum_{n=0}^{\infty} \gamma^k R_{t+n+1} | S_t = s \right], \text{for all s} \in \text{S} \tag{2.36}$$

Using Equation (2.35), the value function can be rewritten as follows:

$$\begin{aligned} V_\pi(s) &= \mathbb{E}[G_t | S_t = s] = \mathbb{E}[R_{t+1} + \gamma G_{t+1} | S_t = s] \\ &= \sum_a \pi(a|s) \sum_{s'} \sum_r p(s', r|s, a) \left[r + \gamma \mathbb{E}[G_{t+1} | S_{t+1} = s'] \right] \\ &= \sum_a \pi(a|s) \sum_{s', r} p(s', r|s, a) \left[r + \gamma V_\pi(s') \right], \text{for all s} \in \text{S} \end{aligned} \tag{2.37}$$

The resulting Equation (2.37) is known as the Bellman equation. This equation articulates the relationship between the value of a given state and the values of the successor states of it (Sutton and Barto, 2018).

The action-value function $Q_\pi(s, a)$ is the value of taking action a while in state s, under a specific policy π, and is given by:

$$Q_\pi(s, a) = \mathbb{E}[G_t|S_t = s, A_t = a] = \left[\sum_{n=0}^{\infty} \gamma R_{t+n+1}|S_t = s, A_t = a\right] \tag{2.38}$$

2.3.3 Common RL Methods

There exist various key categories of techniques to address RL problems, including DP, Monte Carlo, and temporal-difference (TD) learning (Sutton and Barto, 2018). DP methods solve RL problems by solving the Bellman Equation (2.37). These methods are able to compute optimal policies given a complete and precise representation of the environment. The need for complete representation, in combination with the fact that it is computationally intensive, makes these methods less favourable. Monte Carlo methods have the advantage of being model-free and relatively simple to use. They learn by using past experiences only. When it comes to handling incremental computations that need step-by-step solutions, they fall short. TD learning methods combine ideas from Monte Carlo and DP (Sutton and Barto, 2018). Similarly to Monte Carlo methods, TD learning also has the benefit of being model-free. In addition to this, TD methods are also able to handle incremental computation. These are the most widely used methods according to Sutton and Barto (2018).

Q-learning is a popular off-policy TD control algorithm defined by Watkins (1989). The algorithm is defined as:

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha[R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - QS_t, A_t)] \tag{2.39}$$

where Q represents the learned action-value function.

2.3.4 Approximate Solution Methods

In problems with smaller action spaces, the action-value function can be represented as a look-up table (Sutton and Barto, 2018). In larger more complex problems, it is not possible to find the optimal policy within a reasonable time using look-up tables. A key issue using these methods is generalization. It is important that the methods are able to generalize from previous examples to similar situations. This generalization can be done with the use of complex nonlinear

approximators, such as Artificial Neural Networks (ANNs), thus replacing the lookup table used in simpler RL agents (Sutton and Barto, 2018). ANNs are denoted deep ANNs when they have many layers.

2.3.5 Artificial Neural Networks

ANNs are popular to use for nonlinear function approximation in various areas (Sutton and Barto, 2018). ANNs are composed of artificial neurons, which are designed to mimic the human brain. A neuron takes a set of input values, computes the weighted sum of these, and applies an activation function to produce its output signal (Sutton and Barto, 2018). This is illustrated by Figure 2.3. The figure shows a perceptron, which is the simplest form of neural network (Aggarwal et al., 2018). The activation function is typically an S-shaped or sigmoid function (Sutton and Barto, 2018). An ANN may consist of thousands of neurons, arranged in layers.

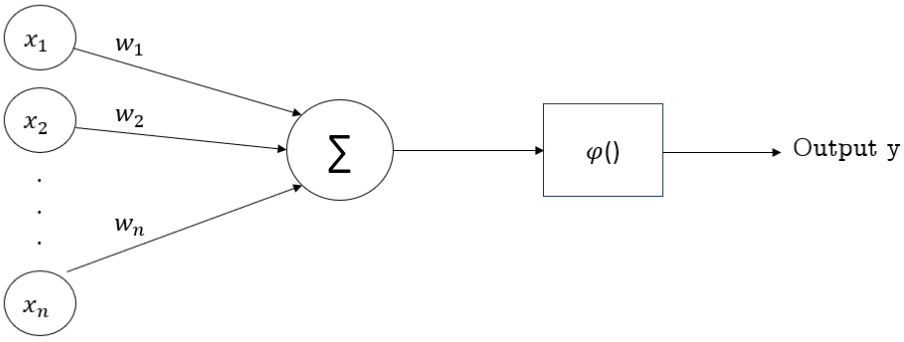


Figure 2.3: Illustration of an artificial neuron with n input values. $\varphi()$ is the activation function.

The standard ANN consists of an input layer, a number of hidden layers, and an output layer. This is also referred to as a feed-forward network (Aggarwal et al., 2018). Each layer consists of a number of neurons, which are connected to a given number of neurons in the following layer, except for the last layer, the output layer. The connections each have a weight associated with them to do the computations of each neuron. The input layer takes in the information given as input to the network. The number of neurons in the input layer corresponds to the dimension of the input data. The hidden layers are denoted hidden because they cannot be accessed directly, and their computations are hidden from the user (Aggarwal et al., 2018). The output layer translates the values outputted from the hidden layers into output values. The number of neurons in the output layer corresponds to the dimension of the possible output. Figure 2.4 shows an ANN with three input values in the input layer, two hidden layers, and two output values in the output layer.

The weights of the connected neurons are updated during the training of the network. Updating of the weights is done using the *backpropagation* algorithm in combination with an optimization

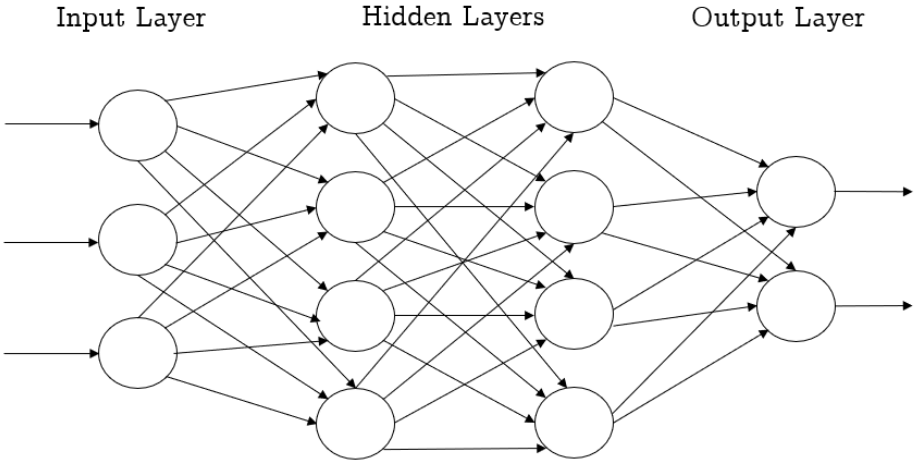


Figure 2.4: Illustration of an ANN with three input values, two hidden layers, and two output values.

algorithm that minimizes the *loss function*, usually some form of gradient descent. The loss function measures the ANN's performance and can for instance by the mean squared error. The backpropagation contains two main phases, the forward phase, and the backward phase. In the forward phase, a training instance is given as input to the network. Using the current weights of the network, an output is produced. The loss, which is the difference between the real value and the output, and the derivative of this loss is then computed. In the backward phase, the chain rule is applied to learn the gradient of the loss function with respect to the weights in the network (Aggarwal et al., 2018). The gradients are then used to update the weights in the direction that decreases the loss function.

The adjustment of the weights is determined by the *learning rate* (Aggarwal et al., 2018). A higher learning rate means that the weights are more adjusted based on the gradients, while a learning rate of zero equals no adjustment. One iteration of sending a training example through the network, and updating the weights based on the gradients is called one *epoch* (Aggarwal et al., 2018).

2.4 The Deep Deterministic Policy Gradient

Lillicrap et al. (2015) present a new method called the Deep Deterministic Policy Gradient (DDPG) algorithm. This is an approximate RL method. The approach is a model-free, off-policy RL method that extends the ideas of Deep Q-networks (DQNs) to continuous action spaces. Lillicrap et al. (2015) demonstrate that their approach can be used to solve problems with continuous action spaces across several domains. A key aspect of this method is its simplicity, and the experiments done by Lillicrap et al. (2015) show that their method uses fewer steps than what is commonly done with DQN algorithms. The DDPG is a powerful algorithm that effectively handles high-dimensional continuous action spaces.

The DDPG is an actor-critic method. Actor-critic methods consist of two ANNs: an actor-network that learns the policies and a critic network that learns to evaluate and update these policies (Sutton and Barto, 2018). In these methods, the actor is used when selecting actions, while the critic estimates the value of those actions, based on the expected cumulative reward.

In each step of the DDPG algorithm, the actor-network selects an action based on the current state. A small amount of noise is added to this action to encourage exploration. Exploration is often a challenge when dealing with continuous action spaces, and is there applied here (Lillicrap et al., 2015). The agent then executes this action in the environment and receives a reward and the next state. The current state, the action, the reward, and the next state are all stored in a replay buffer. The replay buffer is used to address the issues concerned with the dependency of the samples when generated from sequential exploring in the environment (Lillicrap et al., 2015). The state, action, reward, and next state are randomly sampled during learning, breaking the correlation between sequential experiences, and leading to better stability.

The critic network is trained by minimizing the loss function, consisting of the difference between its current value estimates and the target values, which are computed based on the rewards and the next state values estimated by a separate target network. The target networks are copies of the actor and critic networks used to calculate the target values. These networks are updated by slowly tracking the learned network. While this slows down the learning process, it also greatly improves the stability of learning (Lillicrap et al., 2015).

The actor-network is trained to maximize the expected return from each state, as estimated by the critic network. The actor-network is updated using the policy gradient method (Lillicrap et al., 2015). In practice, this is implemented by computing the gradients of the critic's value estimates with respect to the actions and using these gradients to update the actor's parameters.

In addition to this, the DDPG algorithm applies batch normalization to address the issue of learning from low dimensional input data with different physical units (Lillicrap et al., 2015).

Algorithm 1 summarizes the training procedure of the DDPG presented by Lillicrap et al. (2015) at a high level. The parameters in the algorithm are defined as follows:

- **Actor network ($A(s|\theta^A)$):** The actor network serves as the agent's policy function. Given the current state (s), it outputs the action to be taken. θ^A denotes the weights of the actor network.
- **Critic network ($Q(s, a|\theta^Q)$):** This network assesses the value of executing an action (a) at a given state (s). It predicts the expected return of an action, where θ^Q represents the weights of the critic network.
- **Replay buffer (R):** This component is utilized for experience replay, which improves the stability of the learning process by storing and reusing past experiences.

- **Batch size (N):** This parameter defines the number of past experiences sampled from the replay buffer to update the networks.
- **Discount factor (γ):** The discount factor is used to diminish the value of future rewards, emphasizing the importance of immediate rewards.
- **Soft update rate (τ):** This parameter controls the rate at which the weights of the target networks are updated towards the weights of the main networks, thereby aiding stability.
- **Target networks ($A'(s|\theta^{A'})$, $Q'(s,a|\theta^{Q'})$):** These networks provide stable targets for learning, thus mitigating harmful correlations. The target networks are gradually updated to the main networks' weights.
- **Exploration noise (N_t):** This noise is added to the actor network's output to encourage exploration of the environment.

Algorithm 1 DDPG Training Process

Require: Actor network $A(s|\theta^A)$, Critic network $Q(s,a|\theta^Q)$, Replay buffer R, Batch size N, Discount factor γ, Soft update rate τ
Initialize target networks $A'(s|\theta^{A'}) \leftarrow A(s|\theta^A)$, $Q'(s,a|\theta^{Q'}) \leftarrow Q(s,a|\theta^Q)$
for episode $= 1, M$ **do**
Initialize a random process N for action exploration
Receive initial state s_1
for $t = 1, T$ **do**
Select action $a_t = A(s_t|\theta^A) + N_t$ according to the current policy and exploration noise
Execute action a_t and observe reward r_t and observe new state s_{t+1}
Store transition (s_t, a_t, r_t, s_{t+1}) in R
Sample a random minibatch of N transitions (s_i, a_i, r_i, s_{i+1}) from R
Set $y_i = r_i + \gamma Q'(s_{i+1}, A'(s_{i+1}|\theta^{A'})|\theta^{Q'})$
Update critic by minimizing the loss: $L = \frac{1}{N}\sum_i (y_i - Q(s_i, a_i|\theta^Q))^2$
Update the actor policy using the sampled policy gradient:

$$\nabla_{\theta^A} J \approx \frac{1}{N} \sum_i \nabla_a Q(s,a|\theta^Q)|_{s=s_i, a=A(s_i)} \nabla_{\theta^A} A(s|\theta^A)|_{s_i}$$

Update the target networks:

$$\theta^{Q'} \leftarrow \tau\theta^Q + (1-\tau)\theta^{Q'}, \quad \theta^{A'} \leftarrow \tau\theta^A + (1-\tau)\theta^{A'}$$

end for
end for

Chapter 3

Literature Review of The Joint Replenishment Problem

This chapter introduces relevant literature to the Stochastic Joint Replenishment Problem (SJRP) with non-stationary demand. Section 3.1 first presents an overview of how the literary search has been conducted. Section 3.2 presents several versions of the JRP with varying assumptions, and discusses several solution methods to these problems. Section 3.3 presents how Reinforcement Learning (RL) is used to solve the JRP. Finally, Section 3.5 summarizes the literature review, and the theoretical motivation for this thesis is given. Section 3.2 is based on the preliminary work presented in Gravdal (2022).

In the following sections, it is assumed that the reader is familiar with basic terminology within Operations Research and Finance.

3.1 Search Strategy

To find articles on the SJRP with non-stationary demand, a systematic literature review is conducted. The articles were mainly accessed using Google Scholar and Oria, which provided access to most literature found. Some of the search words used were: *the joint replenishment problem*, *the joint replenishment problem with stochastic demand*, *the joint replenishment problem stochastic and non-stationary demand*, *the stochastic joint replenishment problem*, *stochastic dynamic joint replenishment*, *stochastic demand environments joint replenishment*, *joint replenishment using forecasting*, *the stochastic joint replenishment problem reinforcement learning*, *reinforcement learning inventory control*, *joint replenishment reinforcement learning*, *reinforcement learning joint inventory control*, and *safety stock in joint replenishment*.

During the search, we found around 100 articles. The relevance of the articles was then deter-

mined based on the title, abstract, and in some cases more reading.

3.2 The Joint Replenishment Problem

The Joint Replenishment Problem (JRP) is the problem of jointly replenishing a group of products from the same supplier when there exist fixed costs that can be shared (Goyal and Satir, 1989). For most businesses, this is the common practice as opposed to replenishing single products independently (Goyal and Satir, 1989). Arkin et al. (1989) proved that the JRP is NP-hard. It is, therefore, unlikely that an exact algorithm that solves the problem in polynomial time exists. Because of this, it makes sense to consider solving the problem with heuristics, especially for larger instances of the problem.

The objective of JRP models is usually to minimize the total costs while satisfying demand (Khouja and Goyal, 2008), just as for the EOQ Formula described in Section 2.2.6 and other previously mentioned methods. The total costs are commonly composed of two parts, setup costs and holding costs. The setup costs are here composed of two components; a major setup cost, which is independent of which and how many items are ordered, and a minor setup cost for each item ordered. The latter might vary between the items. The holding costs are modelled in the same way as in the single item models as described in Section 2.2.1. Due to the major setup costs, significant savings can be made by group replenishment (Khouja and Goyal, 2008). When a shortage is allowed shortage costs may also be included. These can be calculated based on the customer-specified service level.

The JRP has been heavily researched since the 1960s (Khouja and Goyal, 2008). There are written several reviews on the literature involving this problem. Among these are Goyal and Satir (1989), which gives a review of the literature that was published up until the late 1980s, Khouja and Goyal (2008), which provides a review of the JRP literature from 1989 to 2005, and Peng et al. (2022) which presents a review of the literature from 2006 to 2022. In these reviews, both deterministic and stochastic models are presented. In this thesis, stochastic models are in focus, and these are therefore mainly presented here. Furthermore, these discuss dynamic models, which are also relevant to this thesis. For the interested reader, we refer to the reviews and the references therein.

There exist two different strategies for solving JRPs; a direct grouping strategy (DGS) and an indirect grouping strategy (IGS) (Peng et al., 2022). Using a DGS, the items replenished are divided into sets before solving the model, and the items of each set are jointly replenished with the same ordering schedule. Using IGS, groups are indirectly formed by items having the same ordering times (Khouja and Goyal, 2008). Since an IGS is employed in the problem of this thesis, the literature review concentrates more on methods that employ this strategy.

3.2.1 The Classic JRP

Khouja and Goyal (2008) present the classic JRP problem, which uses similar assumptions as the EOQ formula described in Section 2.2.6. The assumptions include deterministic and uniform demand, no shortages, no quantity discounts, and linear holding costs. Goyal and Satir (1989) recite five methods for the classic problem. First, an enumerative optimal solution procedure by Goyal (1974b) is presented. This method gives upper and lower bounds for the frequency of replenishment, which reduces the computational effort of the problem. Goyal and Satir (1989) then present several heuristic algorithms for solving the problem. These are also mentioned in Khouja and Goyal (2008) as efficient heuristics for the JRP. Brown (1967) presents an iterative procedure, which terminates when the number of cycles between successive replenishments is unchanged for each item. Goyal (1974a) reformulates the minimum cost model originally proposed and iteratively improves the number of cycles between successive replenishments for each item. This algorithm terminates when these are unchanged in successive iterations. Silver (1976) proposes a simple heuristic for determining the operating policy. This algorithm is heavily dependent on the first item, which could potentially lead to excess costs. Goyal and Belton (1979) came up with an improvement to this method, which produces equally good or better solutions, but sometimes produces solutions having higher costs (Kaspi and Rosenblatt, 1983). Kaspi and Rosenblatt (1985) suggested combining the methods proposed by Silver (1976) and Goyal (1974a). In the end, Goyal and Satir (1989) point out that one always must compare the cost of the JRP system to the cost of a simple EOQ solution, and make sure that the gain by having these methods is larger than the excess costs.

Kaspi and Rosenblatt (1991) proposed another algorithm for solving the classic JRP, called the RAND. This is based on computing m equally spaced values of the fundamental cycle within its upper and lower bound $[T_{min}, T_{max}]$, and then applying Silver (1976) improved algorithm for each T. Silver's method was originally iterative until the convergence of $k_i's$. Kaspi and Rosenblatt (1983) discovered that the major improvement happened during the first iteration. The classic JRP is usually solved by the RAND method (Hong and Kim, 2009).

Khouja and Goyal (2008) conclude that the research on JRP shows that effective and fast algorithms exist for solving the classic JRP. The focus on this problem has therefore decreased, and an increased focus on extensions of the problem was seen, for instance, the problem with dynamic demand, quantity discounts, transport restrictions, storage restrictions, and deteriorating products (Khouja and Goyal, 2008). According to Khouja and Goyal (2008), the most critical factor in choosing the method to use seems to be the ratio of the major setup cost to the average of the minor setup costs.

3.2.2 The JRP with Non-Stationary Demand

The JRP with deterministic time-varying demand (DJRP) is a widely studied problem and is relevant for this thesis since the problem studied involves time-varying demand. Khouja and Goyal (2008) described methods and algorithms for solving this problem, which consider a finite time horizon of H time periods. Khouja and Goyal (2008) refer to Boctor et al. (2004) which provide an overview of models and algorithms for the DJRP. Boctor et al. (2004) assume that there exist no quantity discounts, that backlogging is not allowed and that there are no limits on inventory levels or order quantities.

Boctor et al. (2004) explain four properties that are valid for optimal solutions to the DJRP:

1. Any optimal solution to the DJRP is such that if item type i is replenished at the beginning of period t, holding costs for this item type is not paid for time period $t-1$ (Wagner and Whitin, 1958).
2. The optimal order consists of a sum of the demand for an integer number of consecutive periods for each product (Wagner and Whitin, 1958).
3. The optimal inventory level for an item at the beginning of a time period is zero or a sum of demand for an integer number of periods (Wagner and Whitin, 1958).
4. The sum of holding costs for an item should not exceed the sum of joint setup costs and minor setup cost for that item, for any time period t (Silver, 1979)

Boctor et al. (2004) then provide more compact formulations of the problem using these properties, one based on property 2, and one based on property 2 and property 4. These make the problems less computationally extensive to solve since fewer solutions need to be considered. Boctor et al. (2004) also present a new heuristic to solve the DJRP. Their new improved formulation gave significantly better results than the classical formulation.

Narayanan and Robinson (2006) look further into the new heuristic introduced by Boctor et al. (2004) and other heuristics, and in addition to this present improved mathematical formulations. The mathematical formulations are denoted BLR after Boctor et al. (2004). The original formulations used by them are tested, as well as tighter reformulations of these. Narayanan and Robinson (2006) solve the problem for up to 40 items and 48 time periods. Narayanan and Robinson (2006) conclude that the model formulation is very important when solving MIP problems and that in this case, it is important to develop a formulation that gives a tight LP relaxation while yielding a manageable-sized problem. The implementations tested by Narayanan and Robinson (2006) all provided the same LP objective function values. Narayanan and Robinson (2006) point out that BLR heuristic approaches can be very effective, but they are not effective in solving problems with relatively high fixed-cost ratios.

3.2.3 The JRP with Stochastic Demand

The majority of literature considering the SJRP focuses on stationary demand (Khouja and Goyal, 2008). Markov decision processes (MDP) are commonly used in the formulation of the problem (Suetsugu et al., 2020a). (Suetsugu et al., 2020a) states that using the MDP formulation for up to four products is computationally feasible, but that it becomes combinatorially explosive for more than four products due to the exponential growth of the state space. As a result, numerous heuristic algorithms have been suggested for dealing with a large number of products (Suetsugu et al., 2020a).

The two most common classes of heuristic policies used to solve the SJRP are the periodic replenishment policy, introduced by Balintfy (1964), and the can-order policy, first introduced by Atkins and Iyogun (1988). Both of these assume that shortages are backordered and that the inventory positions are tracked (Johansen and Melchiors, 2003). Atkins and Iyogun (1988) also specify a second policy, denoted the modified periodic policy (MP) which is commonly referred to in literature. In this policy, all products that belong to a base set are ordered up to their order up-to level at each review point, while other products are brought up to their order up-to levels at every k review point (Khouja and Goyal, 2008). A well-known periodic replenishment policy is the one suggested by Viswanathan (1997), in which a periodic review $P(s, S)$ policy is reviewed. There are t time units between the reviews for all items. At each review point, all items with an inventory position at or below s are ordered up to their order up to level S (Johansen and Melchiors, 2003).

The $P(s, \boldsymbol{S})$ is also used in newer literature. Wang and Chen (2022) consider a periodic $P(s, \boldsymbol{S})$ policy, in which s is defined as the aggregate reorder level for all products and $\boldsymbol{S}$ is a vector of order-up-to levels for all products. Wang and Chen (2022) also considers service level constraints, involving a minimum cycle service level for each product. To solve this problem, Wang and Chen (2022) develop a mathematical programming model for optimizing the model parameters, and an exact algorithm to solve the model.

The can-order policy is more commonly known as a (s, c, S) policy, in which s is the must-order level, s is the can-order level and S is the order up-to level (Khouja and Goyal, 2008). The must-order level denotes the minimum inventory level at which an item must be reordered. Ordering an item at this level triggers the reordering of other items in the group that have inventory positions at or below their respective can-order levels (Axsäter, 2015). When ordered, the quantity is such that the new inventory levels are the order up-to-levels.

According to the widely recognized Silver et al. (1998), the periodic replenishment policy outperforms the can-order policy. The can-order policy is generally used for continuous review models. However, Johansen and Melchiors (2003) analyze the can-order policy for a periodic review model for the SJRP with stationary demand. The reasoning for this is that even though it often is possible to monitor the inventory levels continuously, there are usually limited replen-

ishment opportunities (Johansen and Melchiors, 2003). Johansen and Melchiors (2003) develop a method based on Markov decision theory to solve the problem. As opposed to Silver et al. (1998), Johansen and Melchiors (2003) demonstrate that the can-order policy performs better than the periodic review replenishment policy. For instances with irregular demand patterns, a reduction of up to 15% in total costs is observed (Johansen and Melchiors, 2003).

Lee and Chew (2005) propose a solution method for the SJRP that can handle non-stationary demand, where demands are auto-correlated and independent of each other. To better estimate future demand, past demand information is used. The policy proposed is periodic, and at every review point, the conditional mean and variance of demands are calculated (Lee and Chew, 2005). Lee and Chew (2005) then use the method proposed by Atkins and Iyogun (1988) to determine the marginal cost of delaying the order of each product. If the marginal costs are positive, then the replenishment of the product is delayed, otherwise, it is ordered. The policy proposed is tested for instances with 12 products and various structures and achieved up to 77% saving compared to the MP policy (Lee and Chew, 2005).

Yang and Kim (2020) considers an SJRP with non-stationary demand in a periodic review system. Yang and Kim (2020) assumes that a distribution for the demand cannot be found. Yang and Kim (2020) determines order up to levels that are variable with time. An integer programming model is developed to determine if an item should be ordered in the given time period or not (Yang and Kim, 2020) with the purpose of minimizing the costs for the upcoming period. A difference between this problem and the problem discussed in this thesis is that Yang and Kim (2020) use forecasts for the upcoming period only to determine if an order should be made, and it is hence not possible to order for multiple periods ahead, as opposed to the problem considered in this thesis where it is possible to order quantities that are supposed to cover demand for more than one time period.

3.3 Reinforcement Learning Methods Used to Solve the JRP

The first known RL-based approach to solving the JRP is Suetsugu et al. (2020b), which used a branching deep Q-network (DQN) with reward allocation. DQN is an off-policy algorithm that uses previous policy samples to enhance its value estimation (Meisheri et al., 2022). The purpose of Suetsugu et al. (2020b)'s approach was to ease some of the strict assumptions, for instance, zero lead time and stationary demand. The problem considered in this article is a periodic-review SJRP non-stationary demand, where container costs are shared by the replenished products (Suetsugu et al., 2020b). This method aims to minimize the sum of holding costs, shortage costs, and transportation costs (Suetsugu et al., 2020b). Individual setup costs are not considered. The action space is limited to four actions for each product, buying zero, one, two, or three items (Suetsugu et al., 2020b). Suetsugu et al. (2020b) state that their proposed agent achieves better results than the benchmark used for up to 50 products and non-stationary demand. Since the action space is very limited, the demand quantity and thus the dimension of the demand space

are also low. Suetsugu et al. (2020b) further state that appropriate reward allocation is very important to get a good solution.

Later, Suetsugu et al. (2020a) propose a multi-agent RL approach to solve the periodic review SJRP under stationary demand. A reason for the development of this approach was that the can-order policy and MP policy could not outperform each other in all cases, even though they each have their strengths and weaknesses (Suetsugu et al., 2020a). This model is on the other hand designed to handle demand variations, correlation in demand, varying number of products, and varying cost structures. To ensure cooperation between the different items, and hereby cost-sharing, a central joint-action selection unit is used such that decisions for all agents are done simultaneously (Suetsugu et al., 2020a). Suetsugu et al. (2020a) first formulated the model using a single-agent Q-learning approach, formulated as follows:

$$Q(s_t, a_t) = (1 - \alpha_t)Q_i(s_{i,t}, a_{i,t}) + \alpha_t(t_{i,t+1} + \gamma \max_{a_i} Q(s_{i,t+1}, a_i)) \tag{3.1}$$

where a_t are actions and s_t are states, often inventory levels.

Using this approach, the state space grew and the network could not converge. Because of this, the problem is modelled using a multi-agent formulation instead, still using Q-learning (Suetsugu et al., 2020a). This is formulated as follows:

$$Q_i(s_{i,t}, a_{i,t}) = (1 - \alpha_t)Q_i(s_{i,t}, a_{i,t}) + \alpha_t(t_{i,t+1} + \gamma \max_{a_i} Q(s_{i,t+1}, a_i)) \tag{3.2}$$

The biggest difference between Suetsugu et al. (2020a) and Suetsugu et al. (2020b) is that Suetsugu et al. (2020a) use other agents' actions in the state-action value function. To determine the joint-action selection, Suetsugu et al. (2020a) develop a joint-action selection heuristic. In addition to this, Suetsugu et al. (2020b) consider non-stationary demand, while Suetsugu et al. (2020a) consider stationary demand.

The results obtained from using the new approach are equal to or better than existing heuristic policies based on can-order or MP policies for most scenarios (Suetsugu et al., 2020a). However, the approach is only tested for up to 10 different items, so it would be favourable to test the method on larger instances to see if they are solvable within a reasonable time (Suetsugu et al., 2020a). In addition, the action space is limited to five actions for each product, bying zero, one, two, three, four or five items Suetsugu et al. (2020a).

Vanvuchelen et al. (2020) also use deep RL to solve the JRP. An addition in this article is that the capacity of a full truck is taken into account (Vanvuchelen et al., 2020). Vanvuchelen et al. (2020) use a proximal policy optimization (PPO) algorithm, which was developed by Schulman et al. (2017), and is a modern method within deep RL. The PPO algorithm implemented is an on-policy method (Meisheri et al., 2022). Actor-critic methods combine policy optimization

and dynamic programming by employing gradient ascent on the policy established by the actor, alongside an estimation of the expected future reward created by the critic (Vanvuchelen et al., 2020). Vanvuchelen et al. (2020) justify the choice of the PPO algorithm by the fact the method is less sensitive to parameter tuning than deep Q-learning, that it is able to capture the convergence properties of trust region methods, and that it is more sample efficient. It is therefore clear that the PPO algorithm has several benefits compared to deep Q-learning, although it demands higher modeling complexity (Vanvuchelen et al., 2020).

Vanvuchelen et al. (2020) define the state of the model at each time period to be the inventory level of each item at the beginning of the time period. The actions are defined as the order quantities of each item for each time period Vanvuchelen et al. (2020). The objective of the problem is to minimize the total costs over the time period. The value function is therefore the expected value of the cumulative costs for being in a certain state and following a certain policy (Vanvuchelen et al., 2020). For small instances, the value functions of the optimal policy can be obtained by solving the Bellman equation (Vanvuchelen et al., 2020). However, when the problem instance is larger, deep RL can be used to avoid the curse of dimensionality (Vanvuchelen et al., 2020).

Vanvuchelen et al. (2020) test their method on instances with only two items, and where demand is drawn from a uniform distribution. Their results demonstrate that their proposed method outperforms other existing heuristics in a small-scale experiment, and that is similar to existing heuristics for a large-scale experiment when the cost structures of the items are similar but outperform the existing heuristics when the cost structures are dissimilar (Vanvuchelen et al., 2020).

Hachaïchi et al. (2020) present a policy gradient-based RL method for the supply chain management. The goal of this work was to develop an agent able to place optimal orders with correct replenishment quantities, at the correct time. The problem Hachaïchi et al. (2020) solve is a multi-echelon problem with several distribution centers and several stores, and it is therefore different from the problem of this report. Still, it has some similarities, and the methods and results are therefore interesting. Hachaïchi et al. (2020) propose a Proximal Polity Optimization (PPO) algorithm and a Deep Deterministic Policy Gradient (DDPG) algorithm. The agents were able to find solutions that were close to optimal, and proof of concept was hereby provided. The PPO turned out to achieve the best results in this setting.

Meisheri et al. (2022) presents a new deep RL method to solve the JRP with varying lead times, unit weights and volumes, shelf lives, and shelf capacities. The method proposed by Meisheri et al. (2022) is parallelizable and they are hence able to solve the JRP for 100-220 products with better results than the baseline heuristics, deep Q-network, and PPO, and with results close to the theoretically optimal solution. Meisheri et al. (2022) does, however, consider a problem that is slightly different from the problem of this study. The problem Meisheri et al. (2022) studies is the problem considering a local warehouse, trucks, and stores. The suppliers are not considered,

and there are followingly no setup costs considered (Meisheri et al., 2022). The intricacy of our problem is on a different scale since the absence of setup costs reduces the necessity for coordinating the ordering times of various products. Thus, dealing with up to 220 products would introduce greater complexity in our case. However, the problem we are discussing in this thesis does bear certain resemblances.

3.4 Forecasting Methods Used When Solving the SJRP

As stated in Section 2.1, information about the demand data is often assumed to be known rather than obtained using forecasting methods, and the reasoning for the use of certain forecasting methods is usually not reasoned for. This section examines which forecasting methods are used in the articles presented that solve the SJRP.

Wang and Chen (2022) assumes stationary demand. The uncertain demand input to the model is given as a value from the normal distribution with a given mean and standard deviation. The demand is hence assumed to be normally distributed. How this is determined is not stated.

In the method proposed by Johansen and Melchiors (2003), the demand is determined using probability mass functions for each product. The probability mass functions vary with various examples tested. It states what they are in the different examples but without reasoning for why.

Lee and Chew (2005) considers demand that follows an autoregressive process. To handle this, an autoregressive model is used to forecast the conditional mean and the conditional variance of the demand at every period. This is done using a time series forecasting method proposed by Box et al. (2015). The demand is said to follow a normal distribution (Lee and Chew, 2005). This is also just stated and not reasoned for.

The forecasts used in Yang and Kim (2020) were derived using past demand data and a distribution of forecast errors. To generate the forecast for the upcoming period in time period τ, the demand data for time period τ and the forecast error are used. The forecast errors are generated using the normal distribution (Yang and Kim, 2020). The standard deviation of the distribution is calculated such that each period's forecast was within 5% error of the actual data with a probability of 99% (Yang and Kim, 2020). This is also increased to 10% and 15% to compare how the policy was influenced by the size of the forecast error (Yang and Kim, 2020).

3.5 Summary and Motivation for the Thesis

As seen in the sections above, the JRP is a widely studied problem. A lot of the papers discussed the classic JRP with constant demand. In recent years, however, a growing focus on extensions

of the problem has been observed. There is a significant amount of literature on the SJRP with stationary demand. The literature on the SJRP with non-stationary demand is, however, limited. Lee and Chew (2005) developed a policy that can handle non-stationary demand, but this is only tested for 12 products. Yang and Kim (2020) also developed a method to solve this problem, but only looking one period ahead, and hereby not taking into account that it might be beneficial to order to cover demand for more than one time period. Suetsugu et al. (2020b) proposed an RL method that is able to solve for 50 products, but the assumptions differ from the problem in this thesis, where minor setup costs and other major setup costs in addition to transportation are considered. In addition, Suetsugu et al. (2020b) consider instances with low demand and few possible actions, leaving much room for improvement. This thesis also focuses on improving the forecasts, which can majorly affect the results of the methods. This is not a central focus in the previously mentioned articles.

This master's thesis aims to develop solution methods that can solve the SJRP with seasonal demand for larger and more realistic problem instances than what has been done previously. The aim is to create two solution methods: a deterministic Mixed Integer Programming (MIP) model that handles uncertainty by looking ahead and deciding appropriate safety stocks, and a more complex RL method that can solve the problem for more realistic scenarios, with regard to setup costs and actions space, which has not currently been found in the literature. In order to achieve better results with the MIP method, several forecasting methods are tested to research which works best on the data used when testing. Due to seasonality in demand, forecasting methods that can handle this are tested. The problem is a relevant problem for many businesses, among them several of Visma's customers. The problem uses a periodic review system and tries to make the best decision at each review point. There are no warehouse space restrictions and no backorders but lost sales if demand cannot be met. In terms of grouping, the strategy used is IGS.

Chapter 4

Problem Description

The periodic review Stochastic Joint Replenishment Problem (SJRP) with seasonal demand is a sequential decision problem in which one at each time period decides replenishment quantities for a group of products that can be ordered simultaneously from a supplier. This is done to minimize the cumulative total costs, consisting of the sum of setup, holding, and shortage costs. An alternative to the shortage cost is to use the service level. This is, however, more difficult to implement.

The setup costs of a JRP consist of a fixed joint part for the order and a minor individual part for each item. The joint setup costs, including the cost of preparing and receiving the order, can then be shared by multiple products, possibly creating significant savings. The individual setup costs are fixed for each item and constant through time and are added if the item is ordered. Holding costs per unit are also assumed to be fixed for each product. The holding costs for each time period are calculated by summing the holding costs for all units left in the warehouse after the demand for the period is covered.

The system considered in this thesis consists of one supplier that supplies a group of products, one warehouse, and customers with a compound demand for each item each time period, e.g. each week. The decisions can only be made at the beginning of each time period. Thus, the system considered is a periodic review system. Each week, a tentative plan for a given planning horizon is made to estimate the total costs of the time periods. New information about the forecasts and inventory levels is provided periodically at the beginning of each time period. This can be utilized when making decisions. Orders can be placed each time period and are delivered at the beginning of the time period it is ordered for, meaning there is an assumption of no lead time.

The demand for the products is uncertain with seasonal variations in demand, meaning that the demand varies throughout a repetitive cycle, and in this case, a yearly cycle. Consequently, demand is considered non-stationary, which complicates the problem. Demand that is not

satisfied is considered lost sales, and there are therefore no backorders. To minimize the sum of shortage costs and holding costs, safety stock is introduced. The safety stock is determined based on the customer-specified service level and the uncertainty in the forecasts.

There are no upper or lower bounds on the sum or quantity of an order to be placed. Space restrictions on the warehouse are not considered in this problem because only a small group of the total selection stored in the warehouse is considered at a time.

Chapter 5

Solution Methods

As Chapter 4 states, the Stochastic Joint Replenishment Problem (SJRP) with Seasonal Demand is a sequential decision problem. At the beginning of each time period, a decision is made about which products should be replenished and in what quantities. This problem can be considered a Markov Decision Process (MDP), which is described in Chapter 2.

Since using deep Reinforcement Learning (RL) seems promising to solve this problem, and the literature on this is sparse, we have chosen to develop such a method, in an attempt to create better methods than those already existing. In the previous work presented in Gravdal (2022), a Mixed Integer Programming (MIP) model was tested to solve the deterministic JRP. We use an extension of this model, including safety stocks, as a benchmark for comparing with the RL approach. In addition to this, it is interesting to research how well the MIP model works on a simulation based on real data.

Solving the deterministic MIP model for this problem is a simplification. There are however benefits of solving this kind of model. First, it is easier to implement and understand the modelling. Second, it can be used for larger instances with more products, due to less complexity. In addition, it is necessary to have more exact input data when solving more exact methods. When this is not available, it might be a good alternative to use this model.

The RL approach can, on the other hand, could potentially handle more complexity, and catch structures that the deterministic MIP cannot catch. Furthermore, unlike deterministic methods, RL can improve its performance over time, refining its policies as it gains more information from interactions, which may result in more robust and efficient solutions in the long run. However, as the problem complexity increases, the state and action spaces grow exponentially, making it difficult to thoroughly explore all possible combinations.

This chapter presents the two solution methods developed. Section 5.1 presents the benchmark approach, a mathematical MIP-based approach, while Section 5.2 presents the more complex

method based on deep RL.

5.1 MIP-Based Approach

This section presents a MIP-based approach for solving the SJRP with non-stationary demand, involving seasonal variations. Section 5.1.1 defines relevant notation. Section 5.1.2 introduces the mathematical model with the objective function and constraints.

The MIP approach is implemented in a rolling horizon framework. The MIP model is run at the beginning of each time period to make decisions based on updated information on forecasts and inventory levels. The model presents the problem solved in time period 1, and the actions are optimized over the time horizon $1...\mathcal{T}$. The model outputs a tentative plan for the given time horizon, which provides an approximation of the costs for the horizon. However, the goal is to implement the best decision possible in the given time period, based on the input parameters. The only decision implemented is the first-period decision. The plan is updated at the beginning of each time period.

The inventory levels at the beginning of the planning horizon are given as parameters based on what the company currently have in stock of the products.

5.1.1 Notation

This section presents the indices, sets, parameters and variables used in the mathematical model.

Indices

- p product
- t time period
- τ number of periods the order should cover demand for

Sets

- $\mathcal{P}$ set of products

Parameters

- S^M major setup cost for one order

- $\mathcal{T}$ number of time periods in the planning horizon
- S_p minor setup cost for product p for one order
- F_{pt} forecasted demand for product p at time period t
- H_p unit inventory holding cost for product p in inventory for one time period
- $SS_{pt\tau}$ safety stock for product p when ordering for τ periods at time period t
- $M_{pt\tau}$ an upper bound for what can be ordered of product p at time period t covering demand for τ periods
- $\alpha_{p\tau}$ required service level of product p when ordering for τ periods
- σ_{pt} standard deviation of the demand forecast error of product p at the start of period t

Variables

- x_{pt} replenishment quantity of product p at the beginning of time period t
- $y_{pt\tau}$ binary variable = 1 if and only if product p is replenished at the beginning of period t covering demand for τ periods, = 0 otherwise
- v_t binary variables taking the value 1 if an order is placed for period t, = 0 otherwise
- i_{pt} inventory level of product p at the end of time period t

5.1.2 Objective Function and Constraints

$$\min TC = \sum_{t\in\mathcal{T}} S^M \cdot v_t + \sum_{p\in\mathcal{P}}\sum_{t\in\mathcal{T}} S_p \cdot \sum_{\tau=1}^{\mathcal{T}-t} y_{pt\tau} + \sum_{p\in\mathcal{P}}\sum_{t\in\mathcal{T}} H_p \cdot i_{pt} \tag{5.1}$$

$$i_{p,t-1} + x_{pt} = F_{pt} + i_{pt} \qquad p \in \mathcal{P}, t = 1, ..., \mathcal{T} \tag{5.2}$$

$$x_{pt} \leq \sum_{\tau=1}^{\mathcal{T}-t} M_{pt\tau} \cdot y_{pt\tau} \qquad p \in \mathcal{P}, t = 1, ..., \mathcal{T} \tag{5.3}$$

$$\sum_{p\in\mathcal{P}}\sum_{\tau=1}^{\mathcal{T}-t} y_{pt\tau} \leq v_t \cdot |\mathcal{P}| \qquad t = 1, ..., \mathcal{T} \tag{5.4}$$

$$\sum_{\tau=1}^{\mathcal{T}-t} y_{pt\tau} \leq 1 \qquad p \in \mathcal{P}, t = 1, ..., \mathcal{T} \qquad (5.5)$$

$$i_{pt} \geq y_{pt1} \cdot SS_{pt1} + \sum_{\tau=2}^{\mathcal{T}-t+1} y_{pt\tau} \cdot (SS_{pt\tau} + \sum_{t'=1}^{\tau-1} F_{p,t+t'}) \qquad p \in \mathcal{P}, t = 1, ..., \mathcal{T} \qquad (5.6)$$

$$i_{pt} \geq SS_{pt1} \qquad p \in \mathcal{P}, t = 1, ..., \mathcal{T} \qquad (5.7)$$

$$y_{pt\tau} \in \{0, 1\} \qquad p \in \mathcal{P}, t = 1, ..., \mathcal{T}, \tau \in 1, ..., \mathcal{T} \qquad (5.8)$$

$$v_t \in \{0, 1\} \qquad t = 1, ..., \mathcal{T} \qquad (5.9)$$

$$x_{pt} \geq 0 \qquad p \in \mathcal{P}, t = 1, ..., \mathcal{T} \qquad (5.10)$$

Equation (5.1) gives the objective function, which is the total relevant costs considered. The first term represents the joint setup costs, the second term represents the minor setup costs and the third term consists of the holding costs.

Constraints (5.2) make sure the inventories are balanced, by making sure that what is bought and the difference in inventory before and after equals forecasted demand for each product for each time period. Constraints (5.3) ensure that minor setup costs incur in case an order including the product is made during the time period. The $M_{pt\tau}$'s are given the value of the forecasted demand for the next τ periods for product p in time period t plus the value of the safety stock for ordering product p in time period t to cover demand for τ periods. The $M_{pt\tau}$'s are calculated in this way to ensure that the $y_{pt\tau}$ variables take the value 1 for the correct value of τ.

Constraints (5.4) ensure that the joint setup costs are incurred if an order is placed for each time period. Constraints (5.5) ensure that one orders for at most one certain number of periods ahead. Constraints (5.6) specify that the inventory levels can be no less than the safety stock amount, plus the cumulative demand forecast for all future periods covered by the order. This constraint considers different order coverage periods and their respective safety stock levels and forecasts, thereby maintaining adequate inventory to meet demand while accounting for uncertainty.

Constraints (5.7) specify that the inventory levels must be no less than the safety stock amount required when ordering to cover demand for one time period. The safety stocks when ordering

for one period ahead, or not ordering, are determined using the customer service level, which is set by the company, and by the uncertainty in the forecasts. The service level measure used is S_1, described in Section 2.2.7.

When ordering for more than one period, the process changes slightly. The uncertainty increases with time, suggesting a need for larger safety stock when ordering to cover demand for several time periods. Figure 5.1 demonstrates how the safety stock increases with increasing τ. It is, however, crucial to prevent the safety stocks from becoming excessively large, which could lead to unnecessarily high holding costs. When placing orders to satisfy demand only for the upcoming period, one considers the possibility of running out of product to consume, leading to incurred shortage costs. These costs can be substantial as they might consist of both lost sales and weakened relationships with the customers. However, if an order to cover more than one period of demand is placed, there's a risk of ordering sooner than anticipated if the order quantity proves insufficient. This scenario could lead to a smaller penalty cost compared to the shortage cost.

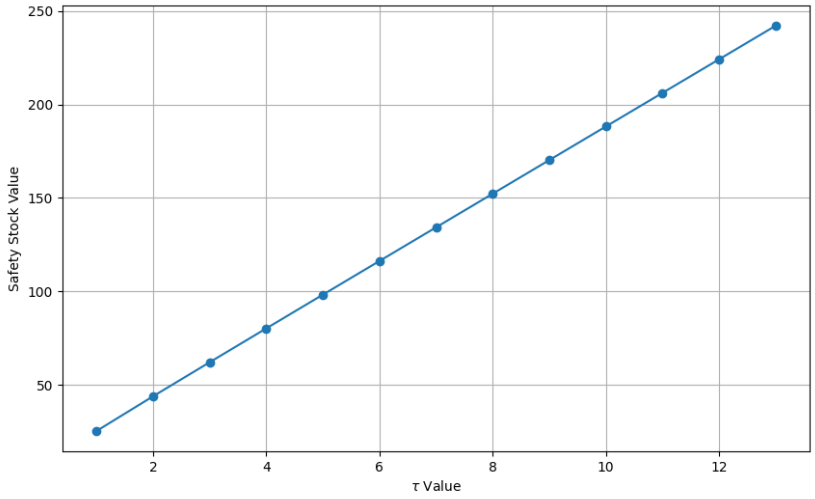


Figure 5.1: The safety stocks for varying values of τ.

Because of this, other values for the service levels for $\tau \geq 2$, hereafter referred to as the internal service levels, are tested. The uncertainty grows with time, resulting in increased safety stocks when using the customer service levels as the internal service levels when ordering to cover demand for more than one period. This can potentially result in an overestimate of the safety stock leading to excessive inventory levels. To see if this was the case, we tested three methods for decreasing the service levels, and thus the safety stocks, for $\tau \geq 2$:

- Exponential decrease of service level:

$$\alpha_{p\tau} = \alpha_{p,\tau-1} \cdot \beta \tag{5.11}$$

- Constant lower service level:

$$\alpha_{p\tau} = \alpha_{p,1} \cdot c \tag{5.12}$$

- Linear decrease of service level:

$$\alpha_{p\tau} = \alpha_{p,1} - (\tau - 1) \cdot \omega \tag{5.13}$$

Different values β, c, and ω are investigated in the computational study in Chapter 7. The parameters yielding the lowest total cost in the tests are used in further computations of the models.

The formula for computing safety stocks, with the customer service levels for $\tau \leq 1$ and the internal service levels for $\tau \geq 2$, is:

$$SS_{pt\tau} = \varphi^{-1}(\alpha_{p\tau}) \cdot \sqrt{\sum_{t'=1}^{\tau} \sigma^2_{p,t+t'}} \tag{5.14}$$

where $\varphi(\cdot)$ represents the cumulative density function for the standard normal distribution and $\varphi^{-1}(\alpha_{p\tau})$ is the safety factor. The service levels, denoted $\alpha_{p\tau}$, vary with τ as described above. $\sigma^2_{p,t+t'}$ represents the standard deviation of the demand forecast error for product p when looking t' periods ahead at time period t. These are calculated as outlined in Section 2.1.3.

Constraints (5.8) and (5.9) specify binary constraints on the necessary variables, while Constraints (5.10) specify the non-negativity constraints on the replenishment quantities.

5.2 Reinforcement Learning Approach

This section presents our implementation of the deep RL algorithm used for solving the SJRP with seasonal demand. The specific RL algorithm used is the Deep Deterministic Policy Gradient (DDPG), which is described in Section 2.3.

The subsequent sections elaborate on important aspects of the implemented RL approach, such as the environment, the states, the actions, the reward function, and the algorithm itself.

5.2.1 States

The state representation in our RL setup includes two components: historical demand data and the current inventory levels for each product. In particular, we include the demand data for the previous 13 periods and the inventory level of the current period, leading to a state representation of 14 periods in total.

In terms of dimensionality, for a problem with n products, the state is a matrix $S \in \mathbb{R}^{14\times n}$, where each row corresponds to a period (13 periods of past demand plus the current inventory level), and each column corresponds to a specific product. This format enables the agent to access both temporal and cross-sectional information regarding demand and inventory levels, which are important in order to predict future demand quantities and coordinated replenishment decisions.

Let s_t denote the state at time t and be defined as:

$$s_t = \begin{bmatrix} d_{t-13,1} & \dots & d_{t-13,n} \\ \vdots & \ddots & \vdots \\ d_{t-1,1} & \dots & d_{t-1,n} \\ I_{t,1} & \dots & I_{t,n} \end{bmatrix} \tag{5.15}$$

where $d_{t-i,p}$ represents the demand of product p in the i-th period before time t, and $I_{t,p}$ represents the inventory level of product p at time t. Here, the matrix s_t is of shape $14 \times n$, where n is the number of products.

5.2.2 Actions

The actions in our RL setup are simply the order quantities for each product. As with the state representation, the action representation is not limited and can take any value equal to or greater than 0. Let x_{pt} be the replenishment quantity of product p at time period t. The action representation, denoted as a_t, is defined as:

$$a_t = \begin{bmatrix} x_{t,1} \\ x_{t,2} \\ \vdots \\ x_{t,n} \end{bmatrix} \tag{5.16}$$

where $x_{t,p}$ represents the replenishment quantity for product p at time period t. Each $x_{t,p}$ is a non-negative real number, reflecting that we can order any quantity greater than or equal to 0 for each product.

In this way, our action space is $\mathbb{R}^n_{\geq 0}$, representing the non-negative real numbers in n dimensions. This action representation allows the model to decide the optimal quantity to order for each product at each time step, based on the current state s_t.

5.2.3 Rewards

The rewards R_t at each time step t are calculated using the total costs associated with a given action a_{t-1}, which is the replenishment order quantities for each product. The total cost, which we aim to minimize, consists of shortage costs, holding costs, major setup costs, and minor setup costs.

Let C_t denote the total cost at time t, and be calculated as follows:

$$C_t = \text{shortage\_costs}_t + \text{holding\_costs}_t + \text{major\_setup\_cost}_t + \text{minor\_setup\_costs}_t \quad (5.17)$$

Since the aim is to minimize these costs, we define the immediate reward R_t at time t as the negative total cost:

$$R_t = -C_t \quad (5.18)$$

This means that lower costs will yield higher rewards making the RL agent incentivized to take actions that lead to lower total costs. By maximizing the rewards over time, the RL agent learns to make decisions that minimize the total cost.

However, Equation (5.18) serves just as a starting point for the reward function. For some product instances, reward shaping needs to be done in order to get the RL agent to learn the proper policy faster and achieve the wanted service level. It is important to note that the problem of our thesis is not a cost minimization problem but a cost minimization problem under a service constraint. Thus the shortage cost should also be trimmed in cases where it results in poor service levels. One example of reward shaping is to give the RL agent some extra punishment if the customer service level is not met.

5.2.4 Environment

The environment in RL controls the current state and determines the next state given a specific action. Our environment integrates the interface convention of OpenAI's Gym library, a popular tool for developing and comparing reinforcement learning algorithms (Brockman et al., 2016). The main responsibilities of our environment are encapsulated within two functions: `get_observation()` and `step(action)`. The `get_observation()` function returns the current state of the system, while the `step(action)` function takes an action as an input, executes it, and transitions the system into a new state. In addition to the new state, the `step(action)` function also provides the reward associated with the transition and a flag indicating whether the terminal state (end of the episode) has been reached. An outline of the `step(action)` function is provided in Algorithm 2.

Algorithm 2 Taking a step in the environment

```
procedure STEP(action)
    total_reward ← 0
    observation ← get_observation()          ▷ retrieving the state from the environment
    if count of positive actions > 0 then
        total_reward ← major_cost                                  ▷ major costs are given
    end if
    for p in self.products do
        p_action ← action[p]
        if p_action > 0 then
            Update inventory level based on action
            p_minor_cost ← minor_cost                              ▷ minor costs are given
        else
            p_minor_cost ← 0
        end if
        Simulate demand for product for the current period
        Calculate p_shortage_cost and p_holding_cost
        Update inventory level based on demand
        p_reward ← p_minor_cost + p_holding_cost + p_shortage_cost
        Add individual_reward to total_reward
    end for
    steps ← steps + 1
    if steps = 52 then
        done ← True
    else
        done ← False
    end if
    return observation, total_reward, done
end procedure
```

5.2.5 Pre-training of the RL agent using Supervised Learning

Before employing the RL algorithm, our approach begins with a supervised learning phase to initially train the actor-network used in the RL algorithm. To create the training data for this phase, we conduct a simulation using MIP model presented in Section 5.1 with deterministic demand. As the demand in this simulation is deterministic, the MIP model does not need to include any safety stocks. This ensures that the actions (order quantities) derived from the model are as accurate as possible for a given state.

The simulation generates a series of inventory levels, demands, and actions for each time step in the series. These form our training dataset: the inventory levels and demands are used as the input to the neural network, and the corresponding actions serve as the target output for the network. We then train our network to minimize the difference between its outputs and the actions of the deterministic model.

By training the neural network on this dataset, the goal is to learn decisions made by the MIP model under deterministic demand. This pre-training phase serves to initialize the RL agent with a reasonable policy. In a stochastic environment with complex actions and demand space, learning a robust policy solely through random exploration can be challenging. However, by initializing the RL agent with a pre-trained neural network, the exploration of the state-action space becomes more efficient. This initial knowledge from expert decisions enables the agent to focus its exploration on refining and adapting the policy to handle the uncertainties of stochastic demand.

After this pre-training phase, the DDPG algorithm is employed to fine-tune the RL agent's policy in a stochastic demand environment. This combined approach with reinforcement learning and supervised learning enables the model to learn from both expert decisions and its own exploration of the environment.

5.2.6 Deep Deterministic Policy Gradient Solution Method

The RL approach in this thesis is based on the DDPG algorithm. A generalized description of the DDPG algorithm is given in Section 2.3. Furthermore, we refer to Lillicrap et al. (2015) for a more detailed explanation of the algorithm. Our implementation integrates measures such as gradient clipping for increased stability and efficiency, facilitating the DDPG algorithm's handling of the high-dimensional, continuous action space in our problem.

Actor Network

The Actor Networks in our RL approach consist of two Long Short-Term Memory (LSTM) layers, each with 200 hidden units. These layers are responsible for processing temporal sequences and

capturing their dependencies and are suitable for predicting time series data as in this thesis. Further details about the specific LSTM configuration are provided in Table 5.1. Note that the learning rate for the actor networks during the RL phase is not outlined, since this varies from product instance to product instance. The proper learning rates will be analyzed in Section 7.2.

Table 5.1: Summary of actor network configuration.

Network Component	**Configuration**
State pathway	2 × [LSTM(200, ReLU) + Dropout(0.5)]
Output layer units	Equals to number of products
Output activation function	ReLU
Optimizer	Adam
Learning Rate (during the supervised learning phase)	0.001
Loss function (during the supervised learning phase)	Mean Squared Error
Batch size (during the supervised learning phase)	128

Critic Network

The Critic networks consist of separate pathways for the state and the action inputs. They are both passed through individual processing before being concatenated. Each pathway consists of a Dense layer with 32 units and a subsequent Dropout layer with a dropout rate of 0.5. After the concatenation of the state and action pathways, the resulting output passes through two more Dense layers, each with 32 units and followed by a Dropout layer with a dropout rate of 0.5. The output of the network is a single value corresponding to the given state-action pair. Further details about the specific LSTM configuration are provided in Table 5.1.

Table 5.2: Summary of critic network configuration.

Network Component	**Configuration**
State/Action Pathways	Dense (32 units, ReLU) + Dropout (0.5)
Concatenation	State and Action pathways
Post-concatenation layers	2 × [Dense (32 units, ReLU) + Dropout (0.5)]
Output layer units	1
Output activation function	ReLU
Optimizer	Adam
Learning Rate	0.003

Chapter 6

Design of Computational Study

This chapter describes the design of the computational study. The mathematical model based on Mixed Integer Programming (MIP) and the deep Reinforcement Learning (RL) approach introduced in Chapter 5 are implemented in Python. These models are tested using a selection of test instances. Section 6.1 describes how the test instances are generated, while Section 6.2 describes how the simulations are performed.

The specifications of the hardware and software used in the testing of the MIP approach are listed in Table 6.1, while the specifications of the hardware and software used in the testing of the RL approach are listed in Table 6.2. The methods were tested on two different computers because of the time limitations. Because of the large difference in running time, the small variations of running time on the computers tested were insignificant.

Table 6.1: Specifications of software and hardware used to test the MIP approach.

Processor	Intel(R) Core(TM) i7 10700 CPU, 2.90 GHz
Memory	16 GB RAM
CPU Cores	8
Operating System	Windows 10
Gurobi Optimizer Version	9.1.2
Python Version	3.8.8

6.1 Generation of Test Instances

In this thesis, the term *test instance* refers to a specific instance of the Stochastic Joint Replenishment Problem (SJRP). Each problem is characterized by a set of input parameters, such as the products included, values of major and minor setup costs, forecasted demands for each

Table 6.2: Specifications of software and hardware used to test the RL approach.

Processor	Apple M2 Max
Memory	32 GB
CPU Cores	12
Operating System	macOS
Python Version	3.9.13

product for each time period, holding costs for each product and unit costs for each product. The key parameters of a test instance are summarized in Table 6.3. The following subsections describe how the parameters of the test instances are determined.

Table 6.3: Key parameters of a test instance.

Notation	**Description**
$\mathcal{P}$	The set of products considered
S^M	Major setup cost
S_p	Minor setup cost for each product p
h_p	Holding costs for each product p
C_p	Unit cost of purchasing a unit of product p
$\alpha_{p\tau}$	Service level product p when buying to cover demand for τ periods

6.1.1 Generation of Demand

The dataset provided by Visma contains 5129 products with data spanning from a few weeks to six years. Table 6.4 shows the format of the dataset along with an example row. The products were categorized into lumpy, intermittent, smooth and erratic demand based on weekly granularity using the cut-off values demonstrated in Figure 2.1.

It is worth noting that the data provided by Visma is sales data and not demand data. In this thesis, we assume that the sales data represent the demand, which is a simplification. Sales data may not accurately reflect true demand due to factors such as stockouts and promotional activities. Therefore, while the assumption is suitable for the purposes of the thesis, it may not capture the complete intricacies of the demand dynamics in real-world scenarios.

Table 6.4: First row of the sales dataset provided by Visma.

product hash	sales quantity	date of sale	requested delivery date	unit price	unit cost
bed089afded097f4b6aa38f45a931365	3.0	2017-02-21	2017-02-21	58.96	124.2

Given the sparse nature of the data, just using actual sales data will not yield a sufficient number

of observations to accurately evaluate the performance of the different models. Therefore, we generate demand for each product based on Visma's dataset, aiming to train and test the models on data that realistically reflects demand patterns. The NumPy library in Python is utilized when generating the random components in the demand patterns. By using a specified seed in NumPy, the demand data can be reproduced consistently. This ensures that the RL and MIP approaches can be compared using the exact same demand data, enabling fair and meaningful evaluations and comparisons of their performance.

Two distinct approaches are utilized to generate data, each tailored to address certain demand categories. One approach addresses the sporadic demand categories, lumpy and intermittent, and the other approach addresses the regular demand categories, smooth and erratic.

Generation of Erratic and Smooth Demand

To generate erratic and smooth demand, the demand for the real products is first decomposed using an additive model of the form:

$$demand\,value = level + seasonality + residual \tag{6.1}$$

In this equation, the *level* component represents the expected value of the demand. Visma has indicated that their standard practice is to focus on recent years' data when generating forecasts, as this best captures the current state of demand. Therefore, the decomposition is based on just a few years of data. Within these years, most products do not exhibit a clear upward or downward trend in demand. Hence, the approximation of no trend is a reasonable approach.

There are two main reasons behind the choice of an additive model. The first reason is the fact that the variance in demand for most products is approximately constant over time, while the second is that the demand might be zero in some periods. The multiplicative model implies that the effect of the level, seasonality, and residuals multiply together to produce the observed data. The occurrence of zero demand may distort these factors, as anything multiplied by zero results in zero. Additionally, multiplicative models assume that the variance increases over time, which is not the case with the majority of our products. In contrast, additive models assume a constant variance, which aligns better with our observations. Additive models are also capable of handling zero values without disrupting the model's components.

Given the nature of our product demand, which is typically quite low, ranging from 0 to 50 for erratic and smooth products, we have selected demand distributions that prevent the simulation from generating negative demand. For the products with smooth demand, we have opted for the Poisson distribution. The Poisson distribution is a common choice in modelling low-demand data, as discussed in Section 2.1.1. In the case of erratic products, where the variance often

surpasses the mean, we have employed the negative binomial distribution, which is discussed in more detail in Section 2.1.1. To generate demand data, we add the level and seasonality components and introduce noise drawn from the respective demand distribution.

Generation of Intermittent and Lumpy Demand

To generate data for intermittent and lumpy products, the demand is again decomposed as shown in Equation (6.1). The generation process involves separately determining the occurrences of demand and the corresponding demand quantity, which is also the principle in Croston's method for forecasting intermittent demand (Croston, 1972).

Johansen and Melchiors (2003) use the Bernoulli demand process to generate demand occurrences instead of a Poisson demand process to fit the periodic model. The system considered in this thesis is also periodic, and the demand occurrences are, therefore, simulated by drawing from a Bernoulli distribution, to decide whether demand occurs in each period. The success probability parameter p used in the distribution is determined by the average number of positive demand occurrences for the specific product. Assuming that demand occurs as a Bernoulli process and thus is geometrically distributed is also a common approach in the forecasting of sporadic demand (Syntetos et al., 2005a). Nonetheless, we acknowledge that this approach may oversimplify the actual complexity of demand, as it assumes demand occurrence in each period is independent, potentially overlooking any autocorrelation within the demand data.

The demand quantity is estimated using a Gaussian kernel density estimation (KDE), which is a non-parametric way of estimating the probability density function of a random variable. Despite the Gaussian nature of the kernel, it is crucial to understand that this does not imply that the demand data adheres to a Gaussian distribution. The kernel's purpose is to smooth the data and provide a refined estimate of the underlying probability density function. The kernel function is fitted to the residuals obtained from the decomposed data, to gain a better understanding of the demand quantity distribution. The KDE was implemented using the 'Scikit-learn' library in Python, a popular machine learning library with a comprehensive suite of statistical tools.

Once the KDE is fitted, noise is generated by sampling from this fitted distribution. The sampled noise and demand occurrences are then combined. If demand occurs in a particular period, the corresponding noise is added to the demand. If no demand occurs, no noise is added for that period.

Finally, the level and seasonality components are reintegrated into the data.

6.1.2 Products

We chose to explore the application of our RL approach for two- and four-product scenarios, without imposing restrictions on demand or actions. This is because previous literature discussing the SJRP using an RL approach has focused either on smaller instances with two products and certain demand limitations or on larger problem instances with only a few discrete actions. Thus, by using our RL approach to solve this more complex problem, we can both fill in the gaps left by previous research and check how well the RL method can be adjusted to handle a broader action and state space.

Because we are interested in comparing the two approaches, the MIP approach is also tested for these instances, and not on larger instances.

Each instance contains only products from the same demand category. Comparing the approaches on instances with similar demand patterns might reveal how well each approach is able to handle products with various demand categories.

6.1.3 Cost Structure

Varying cost structures are tested to research the effectiveness of the developed approaches for different scenarios. First, different values for major and minor setup costs are tested. It is hypothesized that greater major setup costs are likely to give an incentive for more coordination, while larger minor setup costs might give solutions more similar to solving problems for each product individually.

Major setup costs are selected to avoid placing an order at every time period, thus encouraging coordination. The baseline value for the major setup cost is set to 2500. To see how the major setup affects the problem's complexity and solution, a decrease of the baseline major setup costs of 50% and an increase of 100% are tested.

The minor setup costs are determined using a ratio compared to the major setup costs. The ratio is the sum of minor setup costs for all products divided by the major setup cost as shown here:

$$r = \frac{\sum_{p \in \mathcal{P}} S_p}{S^M} \tag{6.2}$$

The minor setup cost is equal for all products because the minor setup cost often is independent of the unit procurement cost and procurement quantity. The main goal is to get different order cycles for each product, and this is achieved with equal minor setup costs due to different demand structures. The ratios between the major setup cost and the minor setup cost tested in the computational study are 0.6, 1.2, and 2.4. These are inspired by Narayanan and Robinson

(2006). The baseline value used in most instances is 1.2, as done in Gravdal (2022).

The holding costs are set to $0.1 \cdot C_p$ per time period in stock, as it was in the preliminary work by Gravdal (2022).

The unit procurement costs are determined using the average of the unit procurement costs for each product in the sales dataset Visma provided.

The shortage costs are calculated using the following equation for service level found in Axsäter (2015):

$$Service\ level = \frac{Shortage\ cost}{Holding\ cost + Shortage\ cost} \tag{6.3}$$

which gives the formula for shortage costs:

$$Shortage\ cost = \frac{Holding\ cost}{\frac{1}{Service\ level} - 1} \tag{6.4}$$

This equation is found when deriving the optimal service level in the case with normally distributed demand (Axsäter, 2015). The service level measure used is 'fill rate', denoted S_2, described in Section 2.2.7. This is derived using the relationship between the expected total costs per time unit and the service level (Axsäter, 2015). Using it in this setting is, therefore, a simplification, however, it still provides a reasonable estimate. The shortage costs are used in the reward function of the RL agent, and to compute the actual costs of using each method in the simulations, and compare these.

6.1.4 Notation

The main differences between instances are: (1) the products included, (2) the major setup cost, (3) the setup cost ratio, and (4) the holding costs. Based on this, the instances are given names in the following way, to identify important components instantly:

`p<numProductTypes>_er<numErratic>_sm<numSmooth>_in<numIntermittent>_lu<numLumpy>_h<holdingCosts>_S<majorSetupCost>_r<setupCostRatio>`

The first entry refers to the number of products, the second, third, fourth and fifth entries refer to the number of erratic, smooth, intermittent and lumpy products, the sixth refers to the holding costs, measured in a percentage of the base unit costs, the seventh refers to the major setup cost and the last refers to the ratio between the sum of minor setup costs and the major setup cost.

6.2 Simulation Framework

This section presents the implementation of the simulation frameworks set up for the MIP and RL approaches. The simulation framework aims to test the effectiveness of the two approaches for solving the SJRP. The framework simulates multiple episodes, each consisting of 52 time periods, here weeks, to evaluate the performance of the two approaches under varying demand scenarios.

The simulation framework has some similarities for both the MIP and RL approach such as the setup parameters and generation of demand, although they are implemented slightly differently. The RL agent also requires a training phase upon comparing it with the MIP. The following sections outline the common steps for both methods and the specifics of each simulation framework.

6.2.1 Simulation Setup

The simulation setup consists of initializing the simulation parameters. In addition to the key parameters for the test instances discussed in Table 6.3, the following settings specific to the simulation frameworks should also be initialized:

- `forecasting_method`: Specifies the method used for demand forecasting. This is only relevant for the MIP model, as the RL agent learns to forecast as part of its training.
- `n_episodes`: Determines the number of simulation episodes to run.
- `episode_length`: Sets the length of each simulation episode in time periods.
- `warm_up_length`: Specifies the length of the warm-up period before collecting data.
- `reset_length`: Determines the length of the reset period between each episode.
- `seed`: A random seed used for reproducibility.

A typical setup is outlined in the following table:

Table 6.5: Configuration parameters for the simulation process.

Section	Parameters
Model Settings	
product_categories	{"erratic": 0, "smooth": 2, "intermittent": 0, "lumpy": 0}
n_time_periods	13
joint_setup_cost	2500
minor_setup_ratio	1.2
service_level	0.95
should_include_safety_stock	True
Simulation	
n_episodes	100
episode_length	900
warm_up_length	1
reset_length	13
seed	0
MIP Settings	
forecasting_method	"Holt Winters"

6.2.2 Data Generation

The data generation process for different product categories is based on the provided real data, as discussed in Section 6.1.1. This ensures realistic demand patterns during simulation. To enable fair and meaningful comparisons between different solution methods, the same seed is utilized when generating data for each method. This use of NumPy's seed function ensures reproducibility and allows for assessing the performance of the models under identical conditions. The data is generated prior to the simulation, with each generated product spanning the same length as the number of dates iterated in the simulation loop. A start date is determined based on the generated data, and for each step in each episode of the simulation, the date is incremented by a week to obtain new data from the generated products.

6.2.3 Simulation of the MIP Approach

The following sections describe the specifics of the MIP approach simulation. A brief overview of the steps is outlined in Algorithm 3.

Forecast Generation

The calculation of forecasts of the data, as well as the standard deviations, is also done prior to the simulation, using the already generated data. This is time-saving when tuning the MIP with different parameters for decreasing the safety stock levels. For each time step t, we compute the forecast $n\_time\_periods$ ahead using the generated date up until time t. The calculation of the standard deviations is done in correspondence to the method outlined in Section 2.1.3. We

use the Holt-Winters forecasting method presented in Section 2.1.2 to generate the forecasts.

Simulation Loop for the MIP Approach

The simulation loop consists of simulating many episodes of the SJRP using the MIP model to decide the actions at each time step in the episode. This subsection provides an overview of the simulation loop and its key components.

The *run_ one_ episode()* function outlined in Algorithm 3 serves as a key component in simulating a single episode of the JRP. It begins by initializing necessary variables and lists to store costs, inventory levels, actions, forecast errors, service levels, and other relevant metrics. Within a loop iterating over each time step, the function optimizes joint replenishment decisions using the MIP model. The MIP model considers current inventory levels, forecasts, standard deviations, β value, the predetermined customer service level, and the holding, shortage and setup costs to determine the optimal actions. predetermined customer service levels are always set to 0.95, and are, therefore, not used as input in the function. The inventory levels are updated based on the recommended actions from the MIP and actual demand, and the performance metrics for comparing it to the RL approach, as well as the actions, are calculated and stored. All the performance metrics such as the obtained total costs, shortage cost, holding cost, setup costs, and achieved service levels are aggregated throughout the episode. Finally, at the end of the episode, the performance metrics and actions are stored for further analysis and reporting.

Three distinct periods are used in the simulation loop: the warm-up period, the main simulation period, and the resetting period. Each period plays a specific role in simulating the SJRP. For each of these periods, the inventory levels at the end of one episode, serve as the start inventory levels of the next episode.

Warm-up Period: To establish realistic start inventory levels, a warm-up period is conducted prior to the main simulation. The warm-up period spans one year (52 time periods) and the inventory levels at the end of this period are used as input at the beginning of the main simulation phase. However, no results or data are collected during this period. It is primarily focused on gradually adjusting the inventory levels from zero to their respective initial states for each product.

Main Simulation Period: Following the warm-up period, the main simulation begins. It consists of a series of episodes, each lasting one year. The results from each episode are collected and written to file for further analysis. For the comparison with the RL approach, 1000 episodes were simulated for each test instance. The choice of 100 was guided in part by the MIP model's extensive time requirements, and in part by the observation that the mean total costs seem to stabilize at 100 episodes, which will be illustrated in Chapter 7.

Resetting Period: To ensure the independence of each simulation, a resetting period is im-

plemented between each of the main simulation episodes. This resetting period spans 13 time periods but is not storing any results or data. It allows the simulation to start independent of the previous episode's outcomes, ensuring the independence of each simulation.

It is important to note that the *run_ one_ episode* function is utilized during all three periods, including the warm-up period, main simulation period, and resetting period.

Algorithm 3 Simulation Framework MIP

Require: Configuration Parameters

Simulation Setup
Initialize simulation parameters and configuration settings.

Demand Data Generation
Generate seasonal demand data for different product categories.

Forecasting
Calculate forecasts for each time period based on the generated demand data

Simulation Loop
Perform warm-up period and update inventory levels, start date, and models
for episode in range(*n_episodes*) **do**
Run *run_ one_ episode* function to simulate the Joint Replenishment Problem
Store actions, total costs, holding costs, shortage costs, setup costs, achieved service level and other useful metrics
Perform reset period
end for

Episode Simulation
function RUN_ONE_EPISODE(start_date, n_time_periods, products, episode_length, forecasts, inventory_levels, beta)
Initialize variables and lists for costs, inventory levels, orders, forecast errors, service levels, etc.
for each time step in the episode **do**
Initialize the MIP with the current inventory levels, forecasts, standard deviation costs.
Run the MIP model to optimize the joint replenishment decisions.
Update inventory levels, based on action from the MIP and the actual demand for the current period.
Calculate costs and service levels and store actions and other relevant metrics.
end for
return Total costs, inventory levels, actions, average run time, achieved service levels etc.
end function

Store information
Write relevant performance metrics and actions to file for later analysis.

6.2.4 Training and Testing the RL Agent

This section describes the training and testing processes of the RL agent for solving the SJRP. The RL agent aims to learn an optimal policy for making joint replenishment decisions based on the observed state of the system. In contrast to the MIP model, the RL agent is trained on multiple yearly episodes of generated data upon being tested. The training and testing of the RL agent were done using TensorFlow's Keras Library in Python.

Training the RL Agent

Before starting the RL training process, a pre-training phase is conducted using Supervised Learning. This is discussed further in Section 5.2.5. The pre-trained agent is then utilized in the RL environment to further improve its actions when facing a stochastic environment.

The train function in the RL approach is responsible for training the RL agent through a series of episodes. Algorithm 1 and Algorithm 2 illustrate a more detailed explanation of the training algorithm. The function begins by initializing variables and lists for storing rewards and average rewards. It also loads pre-trained actors for further training. The environment's costs are set based on the product instance parameters.

The training process follows an episodic loop of 52 time steps, just as in the MIP case. Within each episode, the RL agent interacts with the environment by observing the current state, selecting actions based on the policy from the actor network, and receiving rewards. The agent's exploration-exploitation trade-off is controlled by the Ornstein-Uhlenbeck exploration strategy. The state, action, reward, and next state transitions are recorded in a buffer for experience replay which is used to train the actor and critic networks.

The Ornstein-Uhlenbeck exploration strategy adds a level of noise to the selected actions, which encourages exploration and helps the RL agent to overcome local optima. By incorporating noise into the action selection process, the RL agent is encouraged to explore different actions which are near its current policy, providing a broader exploration of the action space.

The learning process continues until the specified number of episodes is reached, or by early stopping. Early stopping is a technique used in machine learning to prevent overfitting and improve generalization by terminating the learning process based on a predefined criterion, such as the convergence of performance metrics or lack of improvement over a certain number of iterations. The learning process stops if the agent does not improve within 50 epochs. After training, the RL agent's models are saved for future comparison analysis with the MIP model. The average episodic rewards are plotted to visualize the learning progress.

Testing the RL agent

To evaluate the performance of the RL agent, a test function is implemented in a manner similar to the MIP model, incorporating a warm-up period and a reset period. The test function initiates by generating products for the environment and initializing dictionaries and lists to store performance metrics.

Before the simulation phase, a trained actor model is loaded from a file. The RL agent interacts with the environment in a similar manner to the training process, but with a key difference. Unlike during training, the agent does not update its models. Instead, it utilizes the trained actor model to select actions based on the observed state. Furthermore, no noise is added to the agent's actions during testing, indicating that the agent follows its target policy without exploration.

During each episode in the simulation, various performance metrics are calculated and recorded, such as average order quantities, average non-order counts, achieved service levels, and joint ordering counts. These metrics provide insights into the RL agent's performance SJRP, facilitating a meaningful comparison with the MIP model. A total of 1000 episodes were simulated for each test instance.

The following algorithm presents a summary of the test processes employed for evaluating the RL agent's performance:

Algorithm 4 Testing the RL Agent

```
function TEST(episodes)
    Load the pre-trained actor model
    Generate products for the environment
    Initialize dictionaries and lists for storing metrics
    Initialize environment with costs, start inventory level and product demands
    Perform warm-up period
    for each episode in episodes do
        Get the initial state from the environment
        while not done do
            Select an action based on the current state and the trained actor model
            Apply action to the environment and receive the next state, reward and done flag
            Update the episodic metrics and lists
        end while
        Calculate and record performance metrics
        Perform reset period
    end for
    Write the performance metrics and actions to file for later use.
end function
```

Chapter 7

Computational Study - Tuning

This chapter presents a study of the tuning of the two developed methods. This is done before the testing, to get better performances using both methods. Section 7.1 describes the tuning process of the MIP approach, while Section 7.2 describes the tuning of the RL approach.

7.1 Tuning the MIP

The following sections describe the steps committed to tuning the MIP approach, including the choice of a forecasting method, the determining of service levels and the choice of time horizon in the model.

7.1.1 Determining Forecasting Method

In order to find a robust solution for the MIP model under stochastic demand, an appropriate forecasting method is essential. In this context, the forecasting methods selected and tested on the generated data represent a broad spectrum of popular and widely-used methodologies in time series analysis. The selected methods are:

- Holt-Winters method
- Seasonal AutoRegressive Integrated Moving Average (SARIMA)
- Recurrent Neural Network
- The extension of Croston's method by Teunter et al. (2011), hereafter referred to as the TSB method
- Naïve method with using the value one year ago as the forecast

These are discussed in more detail in Section 2.1.2, except for the Naïve method which serves as a benchmark. The Naïve method uses the demand value from the same week the previous year as its forecast.

The forecasting methods were tested on generated data for 20 products of each demand category. Figure 7.1 illustrates the actual demand and the results of each forecasting method for instances with varying demand categories. Their performance was evaluated based on Mean Absolute Error (MAE) and Root Mean Square Error (RMSE). These performance measures provide insight into the accuracy of each forecasting method. The MAE measures the average over the test sample of the absolute differences between demand forecast and actual demand where all individual differences have equal weight. RMSE measures the square root of the average of squared differences between the demand forecast and actual demand.

Looking at the results in Table 7.1, the Holt-Winters method consistently provides the most accurate forecasts across all categories, followed by the Naïve method and SARIMA. It has the lowest MAE and RMSE in all cases except for the MAE where the Naïve method performs slightly better. The TSB method appears to be considerably less accurate in all categories. This is not very surprising, as the TSB method is designed for one-step-ahead forecasts, particularly for intermittent demand. In this study, we are dealing with multi-step-ahead forecasting of seasonal demand.

This suggests that the Holt-Winters method is the most reliable for forecasting in this context, irrespective of the demand category. These findings underline the importance of selecting an appropriate forecasting method for the data at hand, as the choice of method can significantly impact the accuracy of the forecasts and, consequently, the optimization of the MIP model. This is evident in Table 7.2, where we see that using the Holt-Winters method decreases the average costs by about 6% for an instance consisting of 16 products, with 4 products from each demand category.

In the implementation of these forecasting methods, hyperparameter optimization was a critical step to ensure the accuracy and reliability of the forecasts. For this, we employed grid search methodologies in Python, which is a searching technique that systematically goes through a subset of hyperparameter settings in order to find the hyperparameter with the best performance.

For the Holt-Winters method, we utilized the ExponentialSmoothing function in the Statsmodels library in Python. The function estimates the smoothing parameters using maximum likelihood estimation (MLE), an approach that aims to find the parameter values that maximize the likelihood of making the observations given the parameters. For the Holt-Winters method, these parameters include the level smoothing parameter α, the trend smoothing parameter β, and the seasonal smoothing parameter γ. These smoothing parameters were uniquely determined for each product.

As for the SARIMA model, the order parameters (p, d, q) along with the seasonal order param-

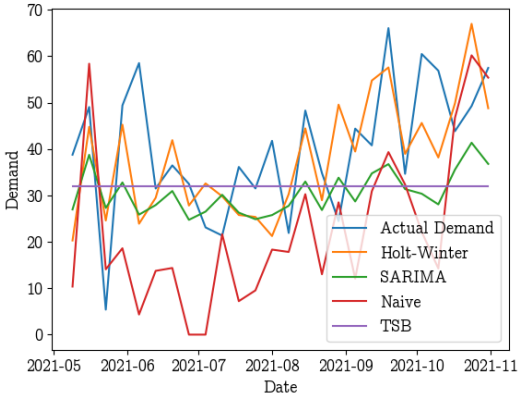
Table 7.1: Forecasting method comparison

Erratic Category		
Method	MAE (95% CI)	RMSE (95% CI)
Holt-Winters	**3.74 [2.91, 4.57]**	**4.65 [3.69, 5.61]**
TSB	4.95 [3.88, 6.02]	6.00 [4.74, 7.26]
SARIMA	3.96 [3.11, 4.80]	4.93 [3.90, 5.95]
Naïve	4.61 [3.65, 5.57]	5.82 [4.67, 6.97]
Smooth Category		
Method	MAE (95% CI)	RMSE (95% CI)
Holt-Winters	**1.14 [0.91, 1.37]**	**1.42 [1.13, 1.71]**
TSB	1.42 [1.18, 1.67]	1.70 [1.42, 1.98]
SARIMA	1.18 [0.96, 1.40]	1.46 [1.17, 1.75]
Naïve	1.38 [1.14, 1.62]	1.73 [1.44, 2.03]
Intermittent Category		
Method	MAE (95% CI)	RMSE (95% CI)
Holt-Winters	**0.32 [0.17, 0.47]**	**0.56 [0.32, 0.79]**
TSB	1.19 [0.94, 1.44]	1.29 [1.01, 1.58]
SARIMA	0.34 [0.19, 0.49]	0.59 [0.35, 0.83]
Naïve	0.33 [0.18, 0.48]	0.70 [0.45, 0.95]
Lumpy Category		
Method	MAE (95% CI)	RMSE (95% CI)
Holt-Winters	0.55 [0.38, 0.72]	**1.05 [0.76, 1.34]**
TSB	2.31 [1.94, 2.67]	2.54 [2.15, 2.93]
SARIMA	0.58 [0.41, 0.75]	1.13 [0.84, 1.42]
Naïve	**0.53 [0.36, 0.70]**	1.26 [0.96, 1.56]

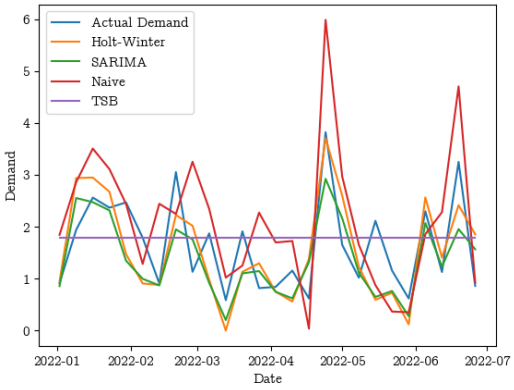
eters (P, D, Q, S) were also optimized using a grid search technique. Each combination of these parameters specifies a different SARIMA model, and the grid search allows the model to select the one that provides the most accurate forecasts on the historical data. The same methodology was used to find the smoothing parameter for the demand rate, α, and the smoothing parameter for the demand interval, β.

Table 7.2: Comparison of Naïve and Holt-Winters forecasting methods for four products of each demand category.

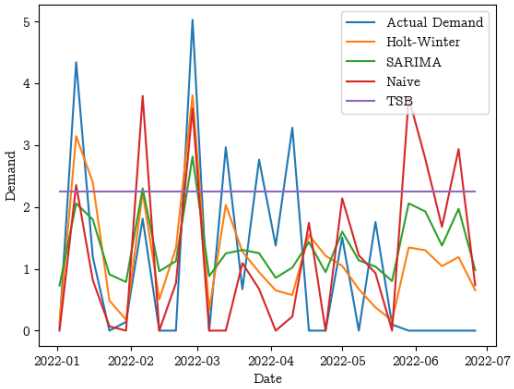
Naïve Approach (NOK)	**Holt-Winters Approach (NOK)**	**Average Cost Difference**
318417.2 ± 2598.5	299408.4 ± 1936.5	19008.8 (6.0%)



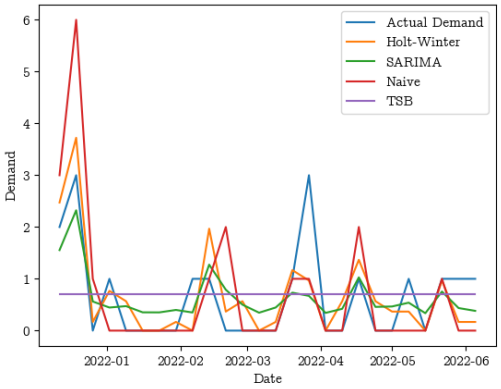
(a) Erratic



(b) Smooth



(c) Intermittent



(d) Lumpy

Figure 7.1: Forecast comparison under the four demand categories. The forecasting horizon is 26 weeks.

7.1.2 Determining the Internal Service Levels

As stated in Section 5.1.2, the safety stocks used in the MIP-based model are determined based on the service level, and the uncertainty in the forecasts. As explained in Section 5.1.2, it might be beneficial to set the service levels when ordering for more than one period lower than the service levels, referred to as internal service levels, when ordering for zero or one time period, referred to as customer service levels. This is done to avoid excessive safety stocks, leading to high holding costs.

Three parametrizations are used to calculate the internal service levels used in the model for $\tau \geq 2$. These parametrizations were initially tested with a wide range of values for 20 simulations for each instance to get a sense of the most relevant values to use in further testing. Some of the results from the pre-testing are shown in Appendix A. The initial testing revealed that keeping the internal service levels relatively close to the customer service levels was the most promising approach.

The values that seemed most promising were then tested on up to 100 simulations of 52 time periods each. Figure 7.2 illustrates how the average total costs develop as a function of how many simulations have been conducted for an instance of the problem. Based on this, it is assumed that 100 simulations are adequate for the average costs to stabilize. To reduce the number of computations, an early stopping method was implemented. The stopping criteria used was that the 90% confidence interval of the differences between the total costs of the same instance using two different values did not contain zero, indicating that one value was better than the other. Comparisons were made with the currently best-identified value for each parametrization method. If the current best value appeared to be better, the simulations were stopped. A minimum of 20 simulations was completed, to avoid stopping too early out of luck with the first simulations.

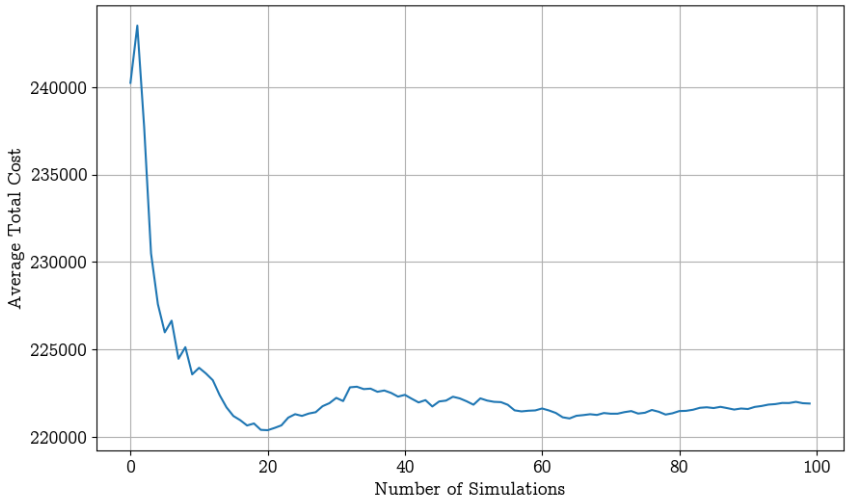


Figure 7.2: The average total costs as a function of the number of simulations conducted.

During the testing of different values in each of the parametrizations, instances with four products from one demand category are tested. The reasoning behind this is to research if the best beta value could be found for each demand category on a general level.

The first parametrization is an exponential decrease in the service level for $\tau \geq 2$:

$$\alpha_{p\tau} = \alpha_{p,\tau-1} \cdot \beta \tag{7.1}$$

The second parametrization is using a constant lower service level for $\tau \geq 2$:

$$\alpha_{p\tau} = \alpha_{p,1} \cdot c \tag{7.2}$$

The last parametrization of the service levels explored in this computational study is a linear decrease in service levels for $\tau \geq 2$ as follows:

$$\alpha_{p\tau} = \alpha_{p,1} - (\tau - 1) \cdot \omega \tag{7.3}$$

A minimum service level of 0.3 was set when using this parametrization. This was mainly done to avoid problems concerning a service level of zero or a negative service level.

Figure 7.3 demonstrates how the average holding, shortage and setup costs vary with increasing β values using the exponential parametrization. To ensure more accurate comparisons, 100 simulations were performed for each case. A tendency of higher holding costs for higher β values

and thus service levels can be observed. This aligns with expectations, as a higher service level typically means higher inventory levels, thereby increasing holding costs. The setup costs seem to decrease when the β value increases. This is likely because fewer orders are placed too early, resulting in lower setup costs. However, there is not a strictly decreasing, and one can observe a small jump in setup costs from $\beta = 0.97$ to $\beta = 0.975$. The reasoning for this can be that $\beta = 0.97$ fits a bit better with the ordering structure, so orders are put less frequently than with $\beta = 0.975$. The shortage costs vary with varying β values, but they do not seem to follow any linear relation. This is as expected, the shortage costs are determined by the customer service level, which remains the same.

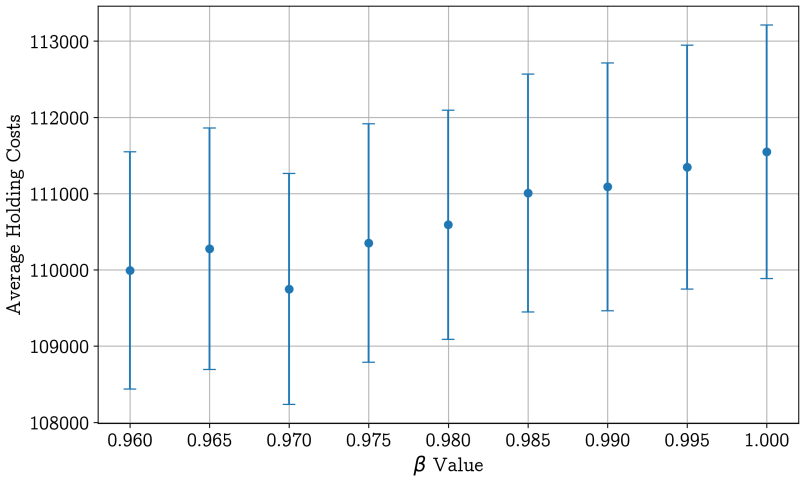
Figure 7.4 shows how the average total costs vary for different values for each of the parametrizations. There are clear variations in the average total costs for different β values when using the exponential parametrization. They do not however follow a linear trend, making it difficult to assume if an increase or decrease would be beneficial. The constant parametrization seems to be more stable but with a slight increase in costs when c is increased. The linear parametrization seems to follow an almost linear curve, with increasing total costs with increasing parameter values.

Table 7.3 presents the optimal values identified for the three parametrizations, each applied to two instances with four products from the same demand category, for each demand category. However, a thorough examination of the data in this table does not reveal a consistent pattern in determining optimal values for instances with four products originating from the same demand category.

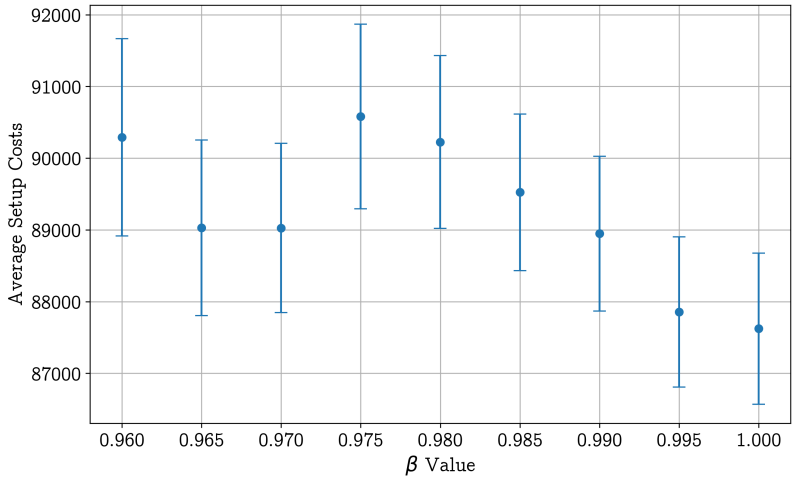
Table 7.3: Best parameter values for each parametrizaton method for instances with four products from one demand category. The seed represents one specific combination of products.

Category	**Seed**	**Best β**	**Best c**	**Best ω**
Erratic	0	0.96	0.995	0.1
Erratic	2	0.995	0.965	0
Smooth	0	0.98	0.96	0.005
Smooth	2	0.975	0.96	0.045
Intermittent	0	1	1	0.01
Intermittent	2	1	0.955	0.005
Lumpy	0	1	0.99	0
Lumpy	2	0.98	0.985	0.0

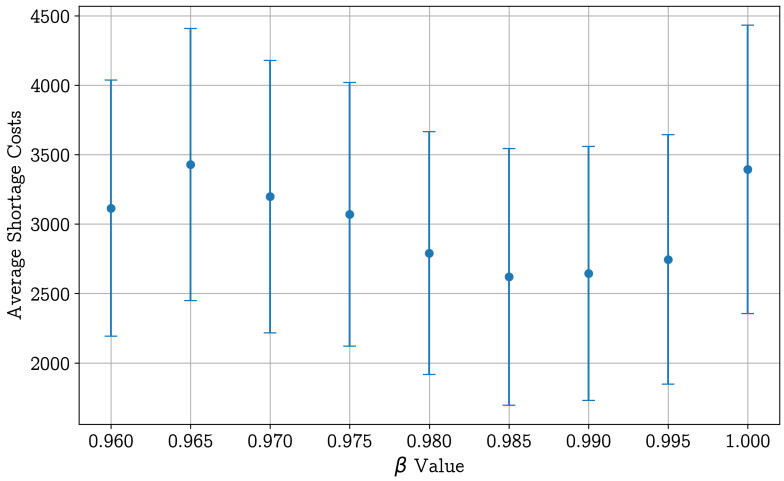
Figure 7.5 presents box plots of the total costs associated with the optimal parameter values for each parametrization for an instance with four erratic products. We conducted 1000 simulations for each parameter configuration to ensure more precise representations. Observing the data,



(a) Holding costs

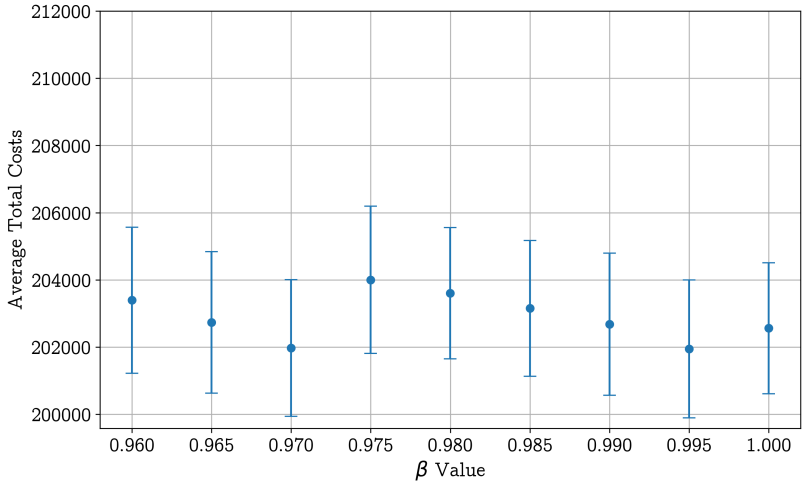


(b) Setup costs

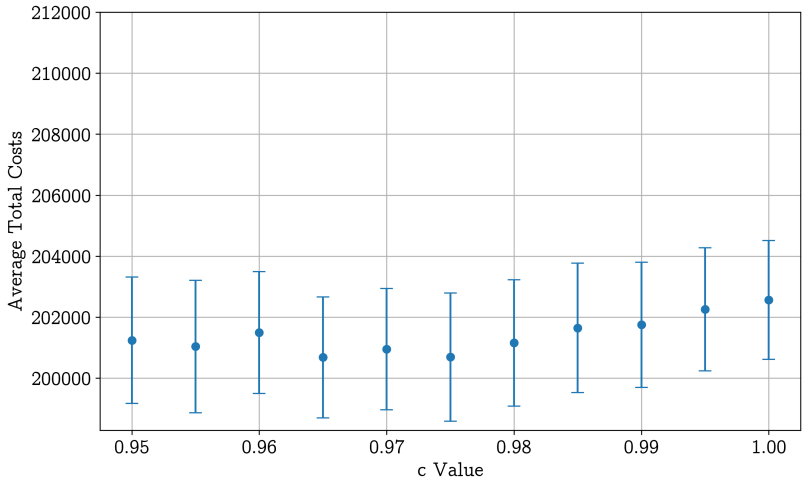


(c) Shortage

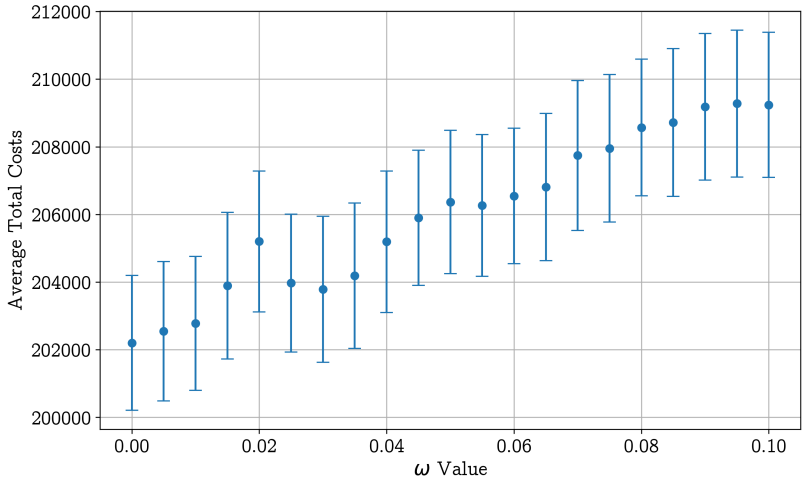
Figure 7.3: Holding, setup and shortage costs for varying values of β using the exponential parametrization.



(a) Exponential parametrization



(b) Constant parametrization



(c) Linear parametrization

Figure 7.4: Total costs for varying values of β, c and ω in the parametrizations for an instance with four erratic products.

the linear parametrization with a value of 0.0 appears to provide the lowest average total costs. However, it is important to note that the cost values across different parameters are relatively comparable. This suggests that although the linear parametrization with 0.0 seems to be slightly more cost-effective, the differences are not statistically significant enough to definitively state that one parameter configuration is superior to the others.

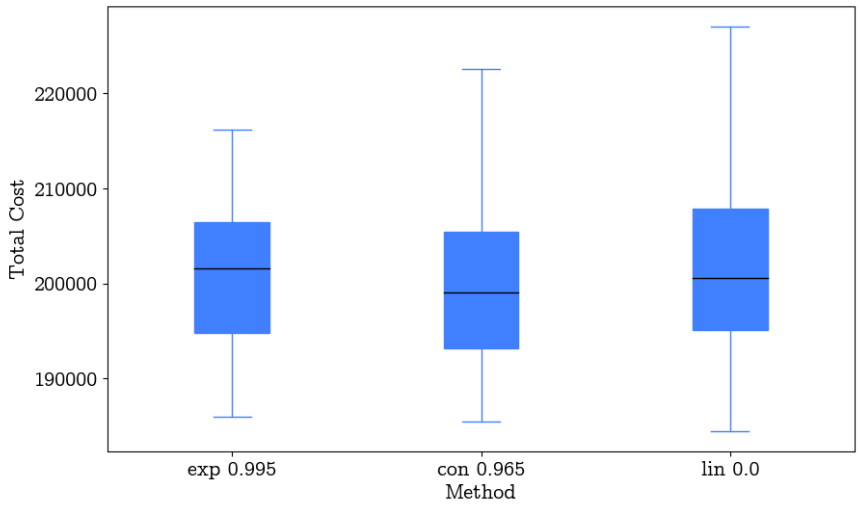


Figure 7.5: Box plot for the best-performing value using each method for an instance with four erratic products.

Based on the analysis of the different parametrizations, no parametrization seems to be universally better than the others. Also, there does not seem to be one value for each parametrization that is universally better than the others. This indicates that these must be tuned individually for each instance or perhaps individually for each product. The analysis also revealed that in the cases where a reduction in internal service level seemed better, only a small reduction was ideal. This also resulted in relatively small cost decreases. One, therefore, needs to consider the costs of performing this step to the potential cost savings of performing it. It should also be mentioned that even though there are tendencies of lower costs for some lower internal service levels, it could not be determined that it was significantly better compared to keeping the internal service level equal to the customer service levels.

7.1.3 Determining the Length of the Time Horizon

The length of the time horizon one considers in the MIP model has an effect on the problem's complexity and hence the solution time. A longer time horizon equals an increased solution time. At the same time, a longer time horizon may provide better solutions, since the decisions can be made with better precision. It is therefore a trade-off. To research the effect of changing the time horizon when solving the MIP-based model, different lengths of the time horizon are tested. The values tested for $\mathcal{T}$ are 13, 26 and 52.

Table 7.4 shows the average costs per simulation and the average run time of running the Gurobi model for instances with a varying number of time periods. As anticipated, an increase in the number of time periods from 13 to 26 decreases the costs. This reduction can be because it can make a more informed decision based on the expected demand for a longer period ahead. However, when increasing from 26 to 52, a marginal increase in total costs can be observed. We believe this can be because of the increasing uncertainty in the forecasts when looking this far ahead. The computational time increases when the number of time periods in the horizon increases. The computational time seems to grow exponentially with an increasing number of time periods. Due to limited time for computations, the simulations are run with instances with 13 time periods in further testing. However, in a real-world scenario, the model is anticipated to be run weekly, and the number of time periods can thus be higher than 13.

Table 7.4: Average costs and run time for a varying number of time periods with test instance p8_er2_sm2_in2_lu2_h0.1_S2500_r1.2 using the MIP approach.

Test instance	Average total costs (NOK)	Average run time
13	250663.8[245733.5,255234.1]	3.65 $\cdot 10^{-3}$ s
26	243248.2[240806.2,245690.2]	3.93 $\cdot 10^{-2}$ s
52	243684.9[241232.1,246137.8]	4.80 $\cdot 10^{-1}$s

7.2 Tuning of the RL

During training our agent in the RL environment, we observed that learning rates exceeding 1×10^{-4} yield sub-optimal performance, while a learning rate of 1×10^{-5} proves to be quite effective. This is shown in Figure 7.8a. The best choice appears to be a combined learning rate approach, which accelerates the agent's learning process. In this approach, the initial 20 episodes are assigned a learning rate of 0, and then the rate is increased to 1×10^{-4}. This capitalizes on the fact that the actor, being pre-trained during the supervised learning phase, already has learned a considerably improved policy compared to an untrained agent.

The motivation behind assigning a zero learning rate for the first 20 episodes is to allow the critic to learn an appropriate value of the state before actor updates are initiated. This avoids the already decent actor policy being updated by values from a critic that is just freshly initialized and outputs random evaluations of states. After 20 episodes, the critic is assumed to provide more reliable evaluations, thus enabling the adjustment of the learning rate to 1×10^{-4}. We observed that this method outperforms the approach of simply setting a 1×10^{-4} learning rate from the start.

As evident in Figure 7.8b, the selection of an appropriate learning rate for the critic is vital for the performance. Based on our findings, a learning rate of 1×10^{-3} slightly outperforms a

learning rate of 1×10^{-2}. As can be observed, finding an optimal learning rate is crucial for the performance as both increasing it to 1×10^{-1} and decreasing it to 1×10^{-4} destroys the learning abilities of the agent.

It is likely that a learning rate of 1×10^{-1} causes drastic updates within the critic network in response to new observations, leading to overfitting.

On the other hand, a lower learning rate of 1×10^{-4} results in underfitting. In this case, the weight updates within the critic network are too mild, preventing the critic from finding an optimal solution. Consequently, the evaluation of states is inaccurate, such that the agent will make suboptimal decisions.

Hence, the balance between the learning rate of the critic and the actor is delicate. The critic network's evaluations serve as a basis for the actor's policy updates, a relationship that was discussed in Section 2.3. Any imbalance in this relationship could potentially undermine the overall performance of the reinforcement learning agent.

Figure 7.6 shows that pre-training of the actor network is a necessity to get the agent to learn a robust policy. The untrained model is not able to explore good actions as it starts off with just a random policy. By pre-training the agent using supervised learning, it starts off with a decent policy making it easier to explore more of the correct actions in the stochastic environment. Having been pre-trained exclusively on deterministic cases, it does not account for uncertainties in the demand.

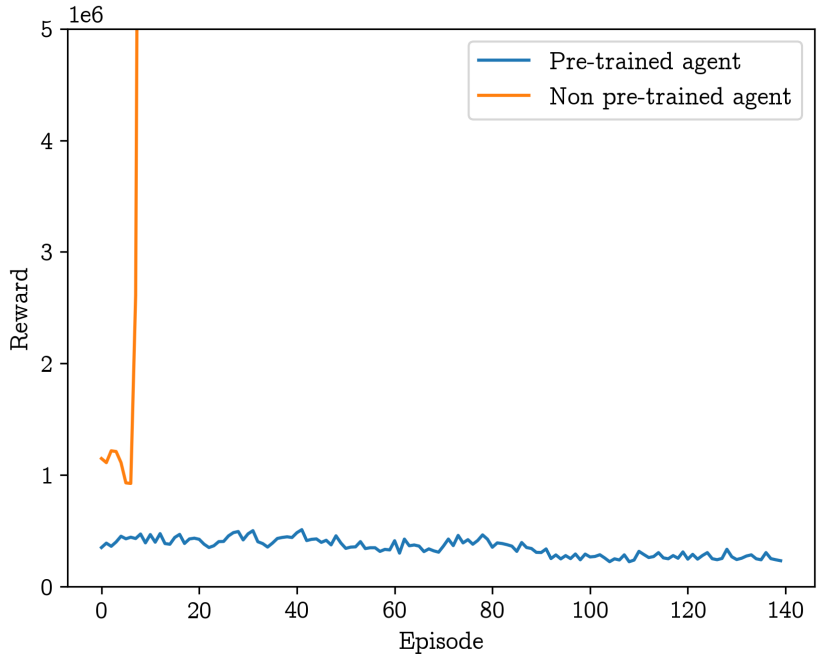


Figure 7.6: Comparing the pre-trained agent with non-pre-trained agent

However, throughout the training phase, the agent learns to maintain safety stock in inventory levels to mitigate the risk of stockouts. This learning is reflected in Figure 7.7, which compares the distributions of shortage, holding, and setup costs between the pre-trained and the fully-trained agent.

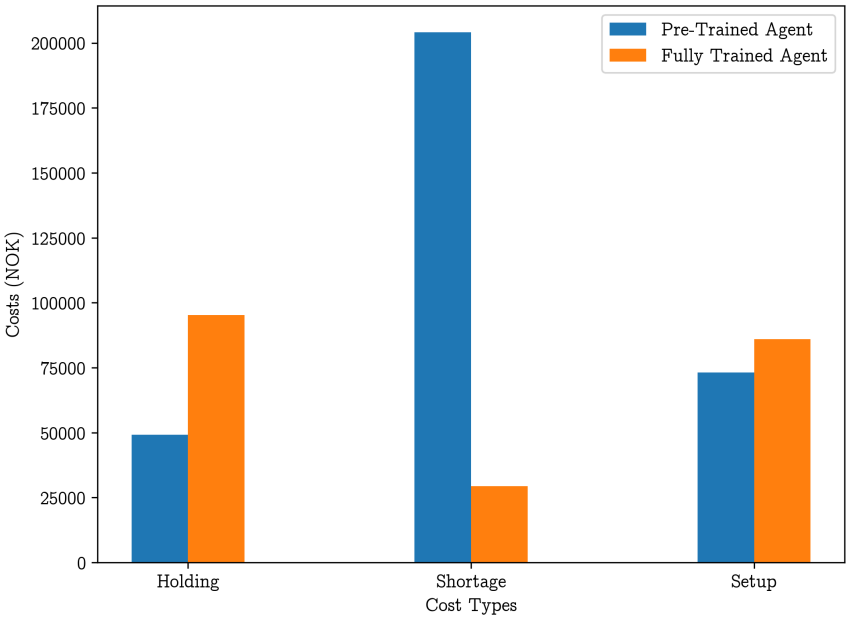
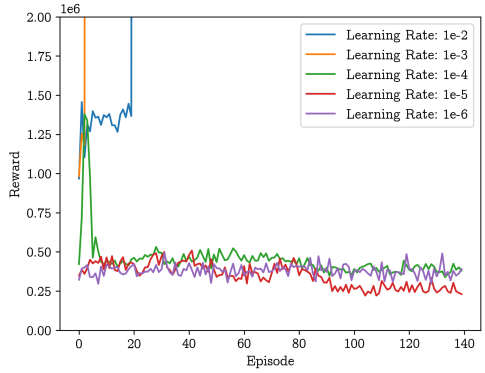


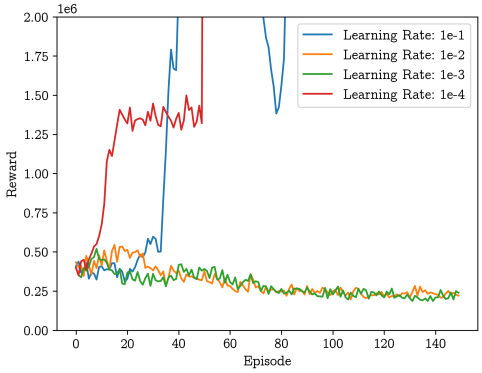
Figure 7.7: Comparing the pre-trained agent with a fully trained agent in terms of holding, shortage and setup costs.

Figure 7.8c shows the different rates of the soft update parameter τ. Finding the right value of tau is also crucial for the learning process, and choosing a soft update rate of either 1×10^{-3} or 5×10^{-4} yields the best performance. This parameter influences the rate at which the target networks are updated to match the main networks and consequently affects the stability and speed of learning. A too low value of τ slows down the learning process while a too high value introduces instability because the target network is updating too rapidly.

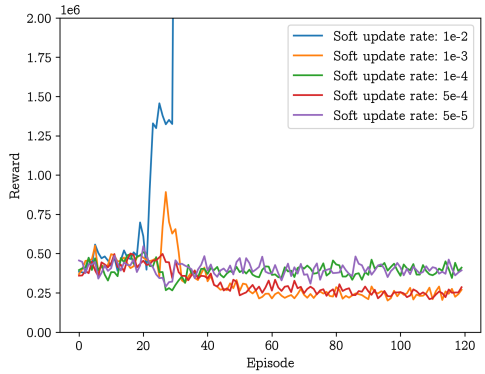
The discount rates of 0, 0.5, 0.8, 0.95, and 0.99 were tested, which is shown in Figure 7.8d. The optimal discount rate found was 0.99, suggesting that prioritizing future rewards when making ordering decisions for the SJRP increases performance. This outcome confirms that the SJRP is fundamentally a problem of long-term optimization where immediate costs or rewards are not as significant as those accumulated over a longer horizon. A high discount factor leads the agent to consider a larger number of future time steps when making a decision, effectively 'looking further into the future.' This makes sense in the context of the SJRP, as order decisions have long-lasting implications in terms of shortage costs, holding costs, and the coordination between products to reduce setup costs.



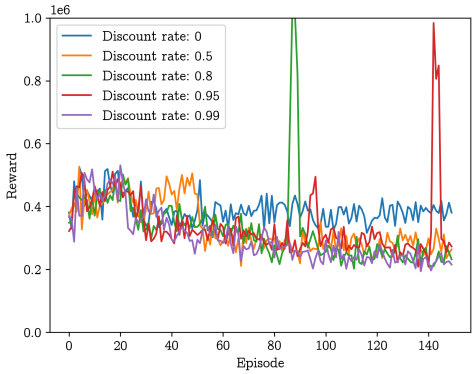
(a) Comparing learning rate for the actor networks



(b) Comparing learning rates for the critic networks



(c) Comparing soft update rates



(d) Comparing discount rates

Figure 7.8: Comparisons of different parameters in the RL model.

Table 7.5 summarizes the parameters used in our RL approach.

Table 7.5: Parameters used in the RL approach.

Parameter	**Value**
Total episodes	1000
Actor learning rate	0.0001 (0.00001 first 20 episodes)
Critic learning rate	0.001
γ (discount factor)	0.99
τ (soft update of target networks)	0.0005
Ornstein-Uhlenbeck noise standard deviation	1.0 (then 0.3 after reaching 0.3)

Chapter 8

Computational Study - Results

This chapter presents a comparative analysis of the RL approach with the Deep Deterministic Policy Gradient (DDPG) algorithm and the Mixed-Integer Programming (MIP) approach, each applied to identical datasets generated with a specific seed. Both models are fined tuned upon comparison, as described in Chapter 7. Section 8.1 presents a comparison of the methods considering two-product scenarios, and Section 8.2 presents a four-product scenario. Section 8.3 presents a comparison of the methods when the cost structure of the problem is changed.

8.1 Two Product Scenarios

Both solution methods are tested under scenarios with two products. This is done to see how the approaches respond to various problem sizes. The test instances contain products from one demand category each. This is chosen because we are interested in how the methods perform on products with different demand patterns. This might provide insights into which demand structures the two methods are better or worse at handling, which in turn could give valuable insights into the strengths and weaknesses of the developed approaches.

Table 8.1 presents a cost comparison of the RL approach and MIP approach tested on instances with two products, using a confidence interval of 95%. With regards to costs, the RL approach is significantly better than the MIP approach for all instances except for the instance with products from the smooth demand category. However, as shown in Table 8.2, the MIP approach generally achieves higher service levels.

A possible explanation for the superior performance of the MIP approach in the smooth demand category could be that the forecasting method used in this approach yields more accurate forecasts for this particular category. The Holt-Winters method utilized is generally more suited to time series data that has some degree of regularity or predictability to it, which is typical for the smooth category. The Holt-Winters method might thus capture the prediction part better

than the RL agent for this category.

On the other hand, if the dataset is too erratic, a forecasting method might struggle to identify a clear pattern, and the forecasts it generates may be less accurate. It can also be observed from Table 8.1 that the RL approach has a greater advantage over the MIP approach for erratic and lumpy demand with regards to costs, than for the intermittent. The RL approach might be more suited than the MIP approach to handling these erratic and lumpy demand patterns due to its ability to learn complex policies and coordinate actions effectively in a stochastic environment.

The reason for using a major setup cost of 1250 for the test instance with smooth demand products, instead of 2500, is that we found the order frequency to be more representative of the smooth demand category in this case. Setting major setup costs to 2500 for the products with smooth demand resulted in an order frequency of only a few orders per year. According to Visma, most products of this demand category are ordered at a higher frequency than that. For the rest of the product categories, a major setup cost of 2500 provided a more realistic order frequency than a major setup cost of 1250.

Table 8.1: Comparison of the RL and MIP approaches for test instances p2_erX_smX_inX_luX_S2500_r1.2, where the X is 2 for the given demand category and 0 for the others. p2_er0_sm2_in0_lu0_S1250_r1.2 is used for smooth. Values are presented as mean[CI_low, CI_high].

Demand Category	RL Approach (NOK)	MIP Approach (NOK)
Erratic	210532.5[209470.5, 211594.5]	221357.6[219649.7, 223065.5]
Smooth	58888.2[58621.2, 59155.2]	58110.1[57256.5, 58963.7]
Intermittent	28289.2[28003.0, 28575.4]	31162.4[30255.2, 32069.5]
Lumpy	31490.76[31106.4, 31875.1]	33135.1[32220.8, 34049.4]

Table 8.2 presents a comparison of the RL and MIP approaches with respect to the average order quantities during the order periods ('Avg Order Quantity'), the average number of orders per year ('Avg Order Frequency'), and the achieved service level.

Though the RL approach attains lower costs compared to the MIP approach for products with intermittent demand, it does not meet the service level criteria of 95%. If the shortage costs fully represent the costs of reducing the service level, then this might indicate that the ordering policy of the RL approach is better than the MIP approach. However, as discussed in Section 5.1.2, other costs for not meeting the predefined 95% service level such as customer dissatisfaction must also be taken into account. Consequently, although the RL approach may prove more efficient in terms of the combination of holding, shortage, and setup costs, maintaining high service levels is equally critical and this aspect may favour the MIP approach in certain scenarios.

For the products with smooth demand, we see that the MIP approach and the RL approach find similar ordering policies in terms of order frequency and order quantity. Both methods also

achieve a service level higher than the customer service level of 95%, though the MIP approach achieves a higher service level than the RL approach. As demonstrated in Figure 8.1, both approaches coordinate their orders to ensure that the two products are ordered jointly, resulting in major setup cost savings. From Figure 8.2, we see that the demand for the two products falls within the same range and frequency. It is therefore reasonable that the two products are ordered together each time they are ordered.

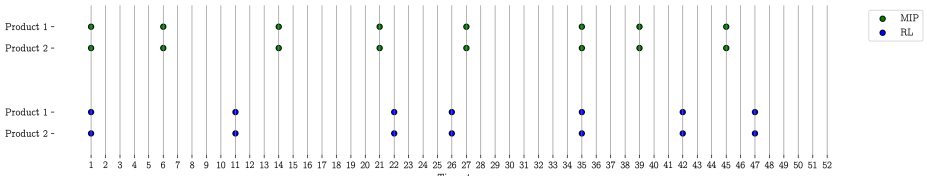


Figure 8.1: Comparing the ordering schedules for the RL and MIP approaches for the instance with smooth demand. Start inventory levels are set to zero.

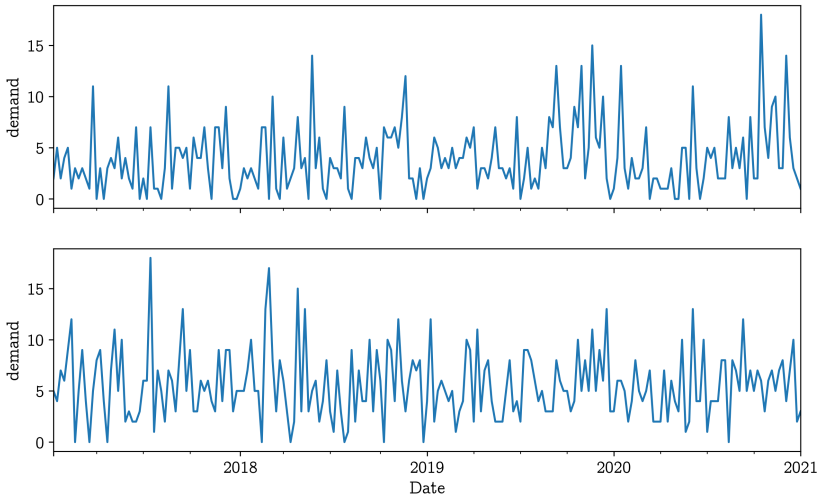


Figure 8.2: The generated demand for the products used in the test instance with smooth demand.

For the two products with erratic demand, the order frequency and order quantity vary a bit more between the two solution methods and between the products. As shown in Table 8.2, the RL approach tends to order product 1 less frequently than the MIP approach, but in larger quantities. Given that the ordering policy for product 2 is similar between the two solution methods, it appears that the ordering policy for product 1 is primarily responsible for the RL approach's superior performance. Figure 8.3 also demonstrates that product 1 has high variability in demand, with the demand ranging from 0 to nearly 100. As stated above, the RL approach might be better suited to handling more variability, making it superior in this case.

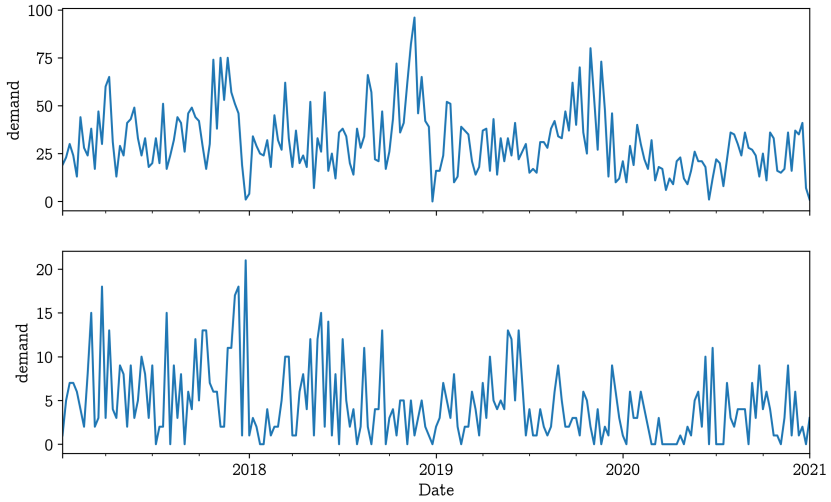


Figure 8.3: The generated demand for the products used in the test instance with erratic demand.

The MIP model requires a certain amount of safety stock for each period to handle demand uncertainties for the upcoming period, and the volume of the safety stock is based on the standard deviations of the forecast errors as discussed in Section 2.1.3. Thus, the MIP model may suffer from greater forecast uncertainty when order quantities fluctuate widely. In scenarios where these errors are considerably large, the model might overestimate the safety stock quantity.

The overestimation of safety stock can lead to fewer reorder points since the excessive safety stock might cover more periods than what was accounted for. This might lead to high holding costs. Contrarily, the RL approach does not impose any specific requirements for safety stock levels. Instead, it learns and adapts its own safety stock policy. This adaptive characteristic might enable the RL approach to discover that maintaining lower safety stock levels can result in lower total costs.

Although the RL approach has an advantage over the MIP approach with regard to cost for the test instance with intermittent demand, Table 8.2 shows that the RL is not satisfactory on the achieved service level. This is a result of ordering smaller quantities and less frequently than the MIP approach does. We can therefore not state that one approach is superior to the other for this instance.

We observe from Table 8.2 that the average order frequencies are alike for both products, under both approaches, for the instance with lumpy demand. However, the order quantities are higher for the MIP approach, resulting in a high volume ordered in total. As with the erratic case, this might be to the overestimation of safety stocks, and it results in unnecessary high holding costs, as indicated in Figure 8.5.

Furthermore, the shortage costs for the lumpy instances in Figure 8.5 combined with the achieved

service level in Table 8.2 indicate a larger difference in the number of lost sales for the RL approach. In other words, the RL does not only have more stock-out scenarios but also losses more sales per stock-out instance on average. This is what could be expected as the demand is lumpy, i.e. has a large difference in demand sizes.

Table 8.2: Comparison of the RL and MIP approaches in terms of average order quantity, average order frequency, and achieved service level for two products with different demand categories

		Product 1			Product 2		
Demand Type	Approach	Avg. Order Quantity	Avg. Order Frequency	Achieved Service Level	Avg. Order Quantity	Avg. Order Frequency	Achieved Service Level
Erratic	RL	83.7	18.6	98.6%	30.9	7.8	99.0%
	MIP	74.9	21.3	99.9%	31.9	7.5	99.9%
Smooth	RL	23.9	8.2	96.3%	32.0	8.0	98.1%
	MIP	24.3	8.3	100.0%	34.3	8.5	99.9%
Intermittent	RL	18.6	2.1	91.1%	13.3	2.0	96.0%
	MIP	19.1	4.0	99.0%	18.8	2.9	97.3%
Lumpy	RL	22.2	3.4	94.8%	24.2	3.4	95.1%
	MIP	24.8	3.6	98.3%	29.6	3.6	98.6%

Figure 8.4 illustrates how the two approaches differ in terms of achieved service level, with the MIP always achieving a higher service level than the RL agent. For the MIP, the service level is generally much higher than the customer service level of 95%.

In cases where the customer service level is specified to be at 95%, it is more desirable to maintain it near this target, rather than significantly exceeding it. This is because exceeding it will lead to additional holding costs. It is thus arguable that the achieved service levels for all instances using the MIP approach are too high. One possible explanation for this could be joint ordering as a result of the major setup costs. The joint ordering results in more frequent orders for some products than if they would be purchased alone, because they join in on an order before their individual 'must order' quantity is reached.

One potential approach to address this would be to lower the service level criteria in the MIP approach so that the achieved service levels become closer to 95%, and see if this reduces the costs. Due to time constraints, we have not investigated this aspect of tuning the MIP model in our thesis.

The RL approach however seems to hit the customer service levels better. It is worth noting that the RL approach does not have the service level criteria explicitly incorporated. Instead, it uses the rewards to adjust its ordering policy, and in particular the shortage costs to adjust its achieved service level.

Although the RL approach has lower total costs than the MIP for the intermittent demand instance, it achieves a lower service level than the customer service level of 95%. This could be because the shortage cost is set too low, so the agent is not incentivized to reach the specific

service level. We attempted to shape the reward function by increasing shortage costs to reach the customer service level, but the agent did not seem to learn this efficiently.

For the erratic demand instance, we were able to adjust the shortage costs in order to hit the customer service level and still beat the MIP approach on costs. Without reward shaping, we achieved a service level of about 90% for product 2 and had higher costs than the MIP model. By increasing the shortage costs for product 2 by 100%, the agent not only learned to avoid a policy that achieved a higher service level but also managed to coordinate its actions effectively, and hereby outperform the MIP approach.

This outcome is likely due to the increase in shortage costs, which strongly incentivized the agent to order more products and reduce shortages. To avoid excessive setup costs, it needed to plan orders for several periods ahead. In this manner, we were able to guide the agent to explore a more beneficial action space. This is a prime example of how reward shaping can enhance the performance of an RL agent.

As observed in Figure 8.4, the RL approach achieves a service level very close to the customer service level for the instance with lumpy demand. Figure 8.4 also illustrate that the achieved service levels are close to the customer service level when using the RL approach on the instance with smooth demand.

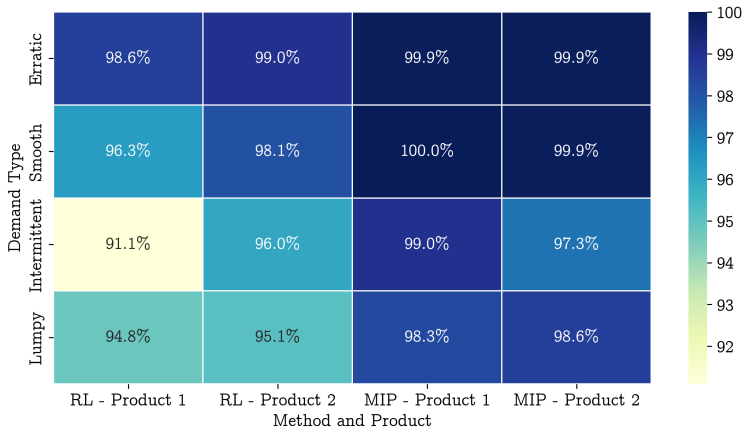


Figure 8.4: Achieved service levels for the MIP and RL approach under different products and product categories

Figure 8.5 demonstrates how the total costs of each instance for each method are distributed across holding, setup, and shortage costs. The error bars depicted at the top of the bar plots represent the confidence interval of the data. Although the total costs are generally lower for the RL approach, except for the test instance with smooth demand, the shortage costs are higher for the RL approach as a result of the less frequent orders and the lower achieved service level observed in Table 8.2. For the MIP approach, the shortage costs are almost non-existent as a result of the high service levels. The setup costs are as expected generally higher for the MIP,

as a result of more frequent orders. The holding costs are also higher for the MIP, likely due to excessive safety stocks.

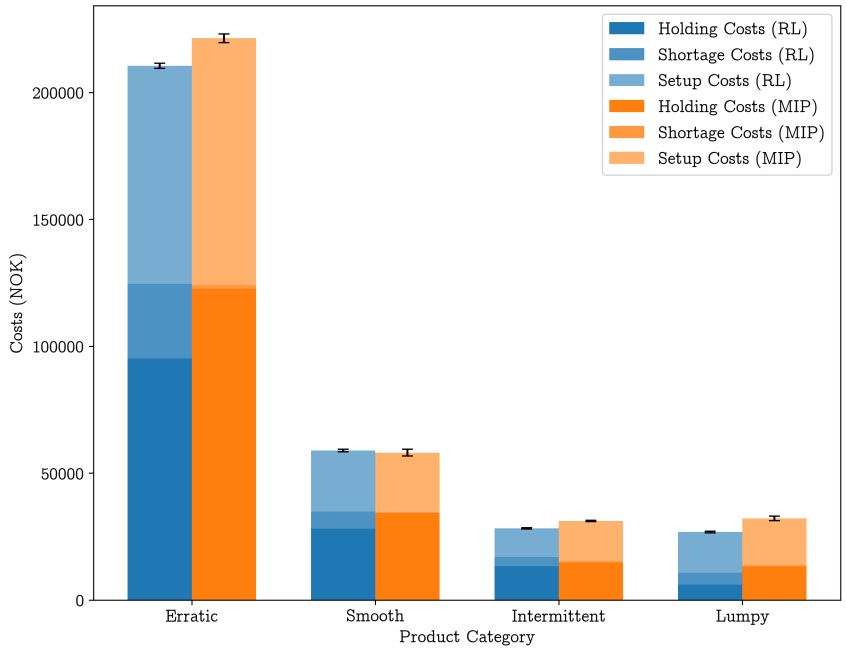


Figure 8.5: Comparing cost compositions of the RL and MIP approaches for different demand categories. The error bars at the top of each bar indicated the confidence interval.

It can be observed from Figure 8.5 that the confidence intervals for the RL approach are much smaller than for the MIP approach. The reasoning for this is that 1000 simulations are done with the RL agent, while 100 are done for the MIP, as outlined in Section 6.2.1.

8.2 Four Product Scenario

An instance with four products is tested to research how the approaches scale for an increasing number of products. The comparison of the average costs of the test instance with four products is shown in Table 8.4. As we can see, when increasing from two to four products, the MIP approach outperforms the RL model, as the RL model does not seem to learn a near-optimal policy. When increasing from two to four products, both the number of actions possible for the agent and the state space increase exponentially. For instance, if each product usually orders between 0 and 100 (which is a realistic situation for several products), then the number of possible actions increases from 100^2 to 100^4.

A key challenge in RL arises from the trade-off between exploration and exploitation. As the

action space expands, the likelihood of locating near-optimal actions through pure exploration diminishes. Consequently, the process of training an RL agent to learn a near-optimal policy could potentially become a question of available computational resources and time. This underscores the need for efficient exploration strategies and potentially imposing structure on the action and state spaces in more high-dimensional problems.

Although it seems like the RL approach does not learn an optimal policy, it is worth noting that it is still able to learn a policy that is not too far from the MIP in terms of costs. This may indicate that it learns many of the aspects of the problem but fall short on some.

Table 8.3 illustrate the average order quantities, average order frequencies and the achieved service levels for the test instance with four products of erratic demand. The table illustrates that the order frequency is similar for both approaches for products 1,2 and 3, but varies more for product 4. The achieved service level also varies more for this product. This might indicate that it would be better for the RL agent to order this product more frequently.

Table 8.3: Average order quantities, average order frequencies and achieved service levels for each product under RL and MIP approaches for an instance containing four products.

Approach	Product	Avg. Order Quantity	Order Frequency	Achieved Service Level
RL	1	57.7	26.9	95.4%
RL	2	34.3	3.8	98.7%
RL	3	41.3	3.7	99.0%
RL	4	63.3	3.7	92.6%
MIP	1	64.4	25.0	99.7%
MIP	2	16.7	4.0	99.9%
MIP	3	16.4	5.0	100.0%
MIP	4	27.7	9.0	99.9%

Table 8.4 illustrates the composition of holding, shortage and setup costs under the RL and MIP approaches for the instance containing four products. As we observed for the two product scenarios, the MIP gets higher holding costs and lower shortage costs compared to the RL approach. This is likely a result of overly high service levels in the MIP as with the two product scenarios.

Table 8.4: Comparison of cost components under the RL and MIP approaches for an instance containing four products.

Approach	Total Costs (NOK)	Holding Costs (NOK)	Shortage Costs (NOK)	Setup Costs (NOK)
MIP	203396.1 (100.0%)	109992.4 (54.1%)	3113.7 (1.5%)	90290.0 (44.4%)
RL	235873.8 (100.0%)	67399.8 (28.6%)	78160.57 (33.1%)	90313.5 (38.3%)

Figure 8.6 demonstrates the ordering schedules for the RL and the MIP for one episode for the instance with four erratic products. By observing this, it looks like the RL sees a benefit from ordering all products together when ordering multiple products, while the MIP orders two, or three products together when this is more suitable. The RL agent has, however, not learned that there might be a benefit from ordering two or three products together. That might lead to

more holding costs because some products are ordered before to get the 'benefit' of ordering all products together. This could explain why it is outperformed by the MIP approach.

This example illustrates that the RL agent has learned many of the aspects of the problem, like how much to order, and how frequently, and that there is a benefit to ordering products together. However, there are still aspects it is unable to learn due to the large state space.

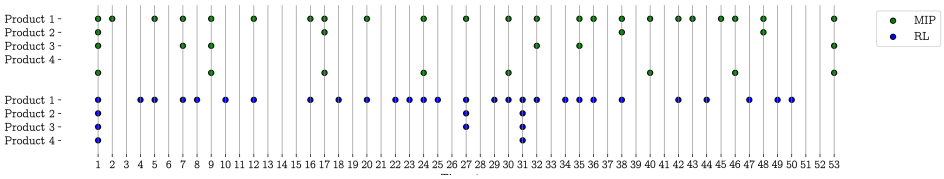


Figure 8.6: Comparing the ordering schedules for the RL and MIP approaches for one episode of the p4_er4_sm0_in0_lu0_S2500_r1.2 instance. Start inventory levels are set to zero.

Because the RL approach was outperformed by the MIP approach in the four product scenarios, we did not test the approaches on larger instances. The previous work done in Gravdal (2022) researched a similar MIP model as the model developed in this thesis. The problem was then solved for instances with up to 192 products within a reasonable time. It is therefore assumed that the MIP approach can be utilized for larger instances of the problem.

8.3 Varying Setup Costs

In this section, we present a comparative analysis examining the impact of changing cost structures, specifically with respect to major and minor setup costs on the performances of the two developed optimization methods. Given the computational time and resources required to train the RL agent and the MIP model, we limit our investigation to an instance comprising two erratic products, namely the `p2_er2_sm0_in0_lu0_S2500_r1.2` instance from Table 8.1. This specific instance was chosen because the order frequency for both products is neither very high nor low, and the products differ both in order frequency and order quantity. This setup allows us to explore a wider range of potential effects when adjusting setup costs on both the ordering policies and cost structures (meaning the distribution of holding, shortage, and setup costs). Our goal is to test the robustness of the two methods and see if either method outperforms the other under specific cost scenarios.

The robustness of a method is essential as it indicates its ability to maintain high-performance levels under diverse operating conditions. This can have significant implications in real-world applications where costs can vary due to factors such as market conditions, supplier policies, or operational constraints. For instance, it might be the case that one method is superior when the order frequency is high, while the other is superior when the order frequency is low. Such

insights could guide companies in choosing the appropriate optimization method based on their specific cost conditions.

From Table 8.1 it is evident that RL approach is significantly better than the MIP approach for the instance `p2_er2_sm0_in0_lu0_S2500_r1.2`. This does not necessarily mean that it is better for other cost structures of the same product, as changing the cost structures will most likely affect the ordering policy. It might also be the case that there is a larger gap between the two approaches for some cost structures than others.

In the following sections, we refer to different variations of the instance `p2_er2_sm0_in0_lu0_S2500_r1.2` using more descriptive terms for readability. This original instance is referred to as the 'Base Case'. We consider four variations of the setup costs. Variations, where the minor setup costs ratio is doubled or halved, are referred to as 'Double Minor' and 'Half Minor', respectively. Variations, where the major setup costs are doubled or halved, are referred to as 'Double Major' and 'Half Major' respectively. These terms should allow for a more intuitive understanding of the variations being considered. The terms and their corresponding instances are summarized in Table 8.5.

Table 8.5: Naming conventions for different variations of the base case instance.

Terminology	**Instance Identifier**
Base Instance	`p2_er2_sm0_in0_lu0_h0.1_S2500_r1.2`
Double Minor	`p2_er2_sm0_in0_lu0_h0.1_S2500_r2.4`
Half Minor	`p2_er2_sm0_in0_lu0_h0.1_S2500_r0.6`
Double Major	`p2_er2_sm0_in0_lu0_h0.1_S5000_r1.2`
Half Major	`p2_er2_sm0_in0_lu0_h0.1_S1250_r1.2`

A comparison of total costs for the different cost instances is presented in Table 8.6. Similarly as described in Section 8.1, the confidence intervals of the RL approach are more narrow than for the MIP as a result of a higher count of simulations.

Table 8.6: Comparison of total costs under RL and MIP approaches for products with erratic demand under different cost structures. Values are presented as mean[CI_low, CI_high].

Setup Costs	**RL Total Costs (NOK)**	**MIP Total Costs (NOK)**
Base Instance	210532.5[209470.5, 211594.5]	221357.6[219649.7, 223065.5]
Half Major	161437.6[160936.9, 161938.3]	163089.2[161215.5, 164962.9]
Double Major	283887.4[282935.2, 284839.6]	303007.3[300939.8, 305074.5]
Half Minor	192308.3[191315.4, 193301.2]	200648.6[198985.6, 202311.6]
Double Minor	255467.2[254679.1, 256255.3]	261779.7[259892.8, 263666.6]

The RL approach has significantly lower costs than the MIP approach with a confidence interval

of 95%, in all cases, except for the 'Half Major' case, where it can not be determined that one approach is significantly better than the other. In this case, we were not able to fine-tune the RL agent as we wanted and had to use the same RL agent utilized in the 'Base Instance'. It is, however, worth noting that it still has a lower mean than the MIP approach. For all the other methods, the RL agent outperformed the MIP model and was able to learn the new cost structures efficiently.

The inability to fine-tune the RL agent in the 'Half Major' case, can be due to a variety of factors. A potential explanation emerges during an examination of the MIP solution for the 'Half Major' instance in Figure 8.8 and Figure 8.9. It appears that the order frequency for product 1 is substantially higher for this specific cost instance, while the corresponding order quantity is substantially lower. This suggests that the optimal strategy may involve placing orders more frequently but in smaller quantities compared to the ordering strategy learned by the RL agent.

RL algorithms often struggle with balancing immediate rewards with delayed ones. In this instance, although the major setup costs are reduced compared to the 'Base Instance', ordering still incurs considerable immediate minor costs. As a result, the agent receives a larger immediate reward when choosing not to order, which could potentially misdirect the agent's learning process. If the agent fails to explore the action space that involves ordering to cover demand for a larger number of periods, its policy may get stuck in a local optimum.

This challenge could be related to the agent's reward function and exploration strategy. The agent may not have explored the state space sufficiently to realize that more frequent orders could lead to lower long-term costs. One potential solution could be to shape the reward function to promote more frequent ordering for product 1 as discussed in Section 8.1. However, such reward shaping should be handled with care. Overemphasizing frequency might lead the agent to learn a suboptimal policy by ordering too much. The balance between exploration and exploitation is delicate, but finding the right one might improve the RL agent's performance in scenarios like this.

Table 8.7 shows the service levels for the different test instances. We see that both the MIP and the RL approach serve a higher service level than the customer service level criteria of 95%. For a discussion of why this is the case, see Section 8.1. However, just as in Section 8.1, the RL approach provides a slightly lower service level, which introduces higher shortage costs, but lower holding costs. Figure 8.7 confirms this, as the shortage costs for the RL approach are much higher than for the MIP approach, while the holding costs are lower in all cost setups except the 'Half Major' setup.

Table 8.7: Comparison of achieved service levels under RL and MIP approaches for products with erratic demand under different setup costs

Setup Costs	Approach	Product 1	Product 2
Base Instance	RL	98.6%	99.0%
Base Instance	MIP	99.9%	99.9%
Half Major	RL	98.9%	99.4%
Half Major	MIP	99.5%	99.8%
Double Major	RL	98.9%	98.6%
Double Major	MIP	99.9%	99.9%
Half Minor	RL	99.0%	97.8%
Half Minor	MIP	99.8%	99.9%
Double Minor	RL	99.4%	99.9%
Double Minor	MIP	99.9%	100.0%

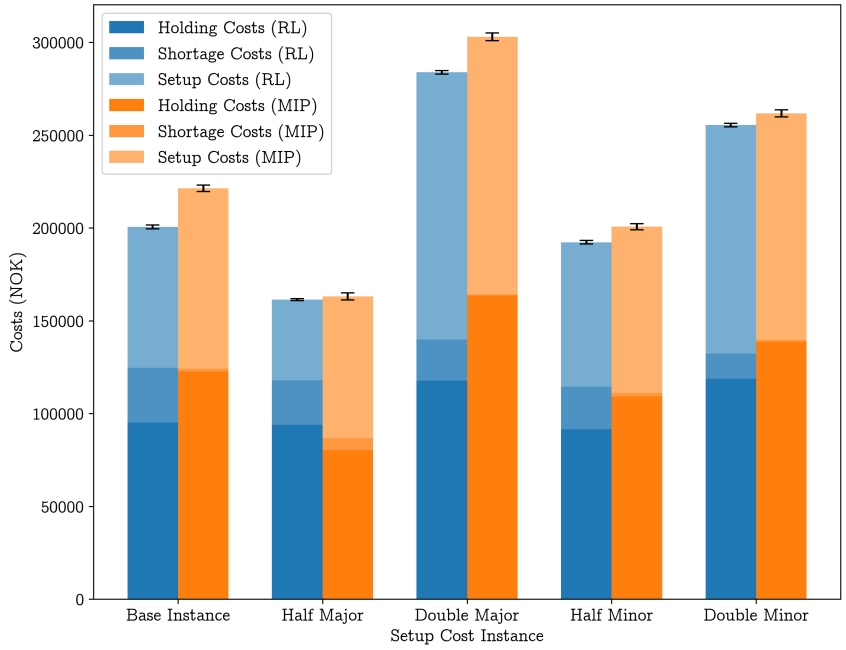


Figure 8.7: Comparing the RL and MIP approaches on cost compositions for different cost structures

Figure 8.8 and Figure 8.9 illustrate the order frequencies and the average order quantities for the instances with different cost structures. Product 1 consistently manifests a higher average order frequency and quantity than product 2. The joint bars Figure 8.8 refers to the joint ordering frequency. These figures indicate that the average order frequencies decrease and the average order quantities increase when the major or minor setup costs are doubled. This is reasonable

as increasing ordering costs encourages larger, less frequent orders to reduce overall expenditure. Conversely, halving the setup costs results in higher frequencies and lower quantities.

Comparing the RL approach with the MIP approach reveals that doubling the major setup costs has a more pronounced impact on the RL approach. For the MIP approach, the order frequency and quantity of product 2 remain relatively stable, likely due to its tendency to join orders with product 1, as indicated by Figure 8.8. This is also evident in Figure 8.11 which shows reorder points for an example episode for the 'Double Major' instance, where the start inventory levels are set to zero. However, the order frequency of product 1 decreases and the order quantity increases, as expected to save total costs.

The RL approach, however, adjusts the ordering policy for product 2 under the 'Double Major' scenario. This could be due to the variability in the RL agent's exploration process. The shifting reward function could also contribute to a different optimal policy. Figure 8.7 reveals that the RL approach is superior to the MIP approach despite their distinct order policies.

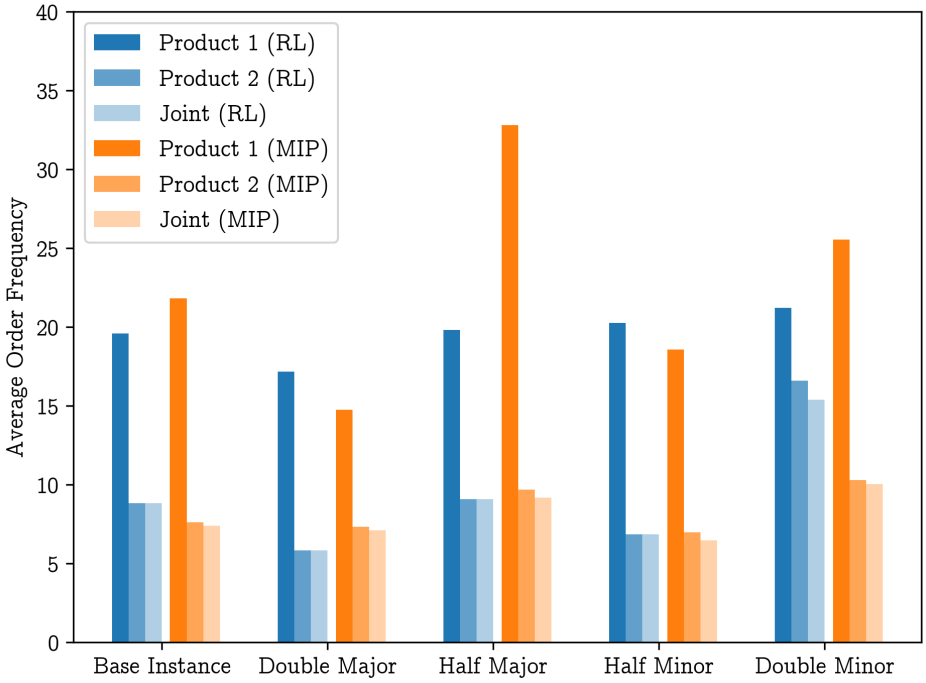


Figure 8.8: Comparing the average order frequency for the RL and MIP approaches for different cost structures

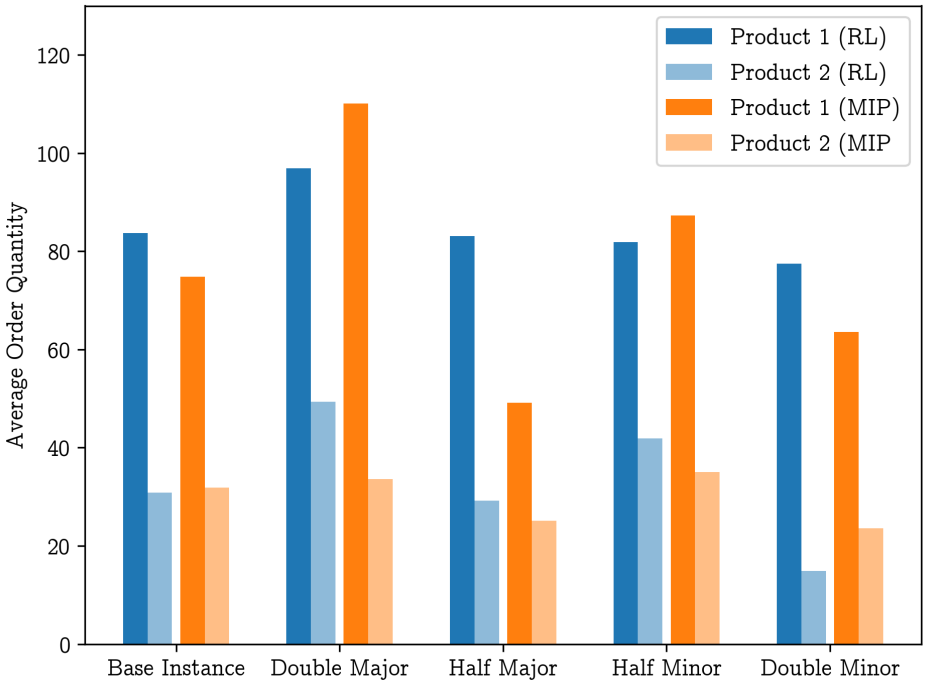


Figure 8.9: Comparing the average order quantity for the RL and MIP approaches for different cost structures

It is interesting to note that the MIP approach has more independent orders for product 2 than the RL approach does. Yet, in the example episode shown in Figure 8.11, product 2 places an independent order in period 10 for the RL approach. This may not be optimal given the high frequency of product 1 orders, indicating potential areas for policy improvement. However, delaying the order to join the order of product 1 in a later period may also be suboptimal, risking high shortage costs if stock is out.

When the major setup costs are reduced, the order frequency increases while the order quantity decreases because it becomes less expensive to order. Thus, keeping high inventory levels would yield unnecessary high holding costs.

The influence of varying the ratio between minor and major setup costs on ordering patterns is another critical aspect. Changes in minor setup costs have a smaller impact on product 1 for the RL approach, but significantly alter product 2's ordering frequency and quantity. For the MIP approach, the effects of increasing and decreasing the ratio have similar effects on both products. In particular, when minor setup costs are reduced, the product orders increase due to reduced costs associated with joining orders and vice versa.

When the minor costs are reduced, the RL approach exhibits a significant adjustment in its ordering policy for product 2. The reduction of minor costs allows product 2 to join orders more often. This adaptation could be advantageous if the reduced minor setup costs outweigh the corresponding holding costs associated with less frequent, larger orders. This contrasts with the

'Half Major' case, where reductions do not directly influence product 2's contribution to total costs, as it typically joins an already set-up order. However, a reduction in minor costs explicitly affects product 2's contribution to the total costs. This could be why the ordering frequency for product 2 increases more in the 'Half Minor' case than in the 'Half Major' case. This effect is evident in both the MIP approach and the RL approach although more extreme for the RL approach.

Higher safety stock levels in the MIP approach, as shown in Table 8.7 and Figure 8.7, could explain why the order frequency does not increase as much as in the RL approach when minor costs are halved. The RL agent, with lower safety stocks, can order more frequently without accumulating excessive inventory. Hence, it provides a more responsive and dynamic ordering policy. The sample episodes in Figure 8.10 and Figure 8.12 exemplify how the reorder policies for product 2 diverge between the MIP and RL approaches for the 'Base Instance' and the 'Half Minor' instance, illustrating this.

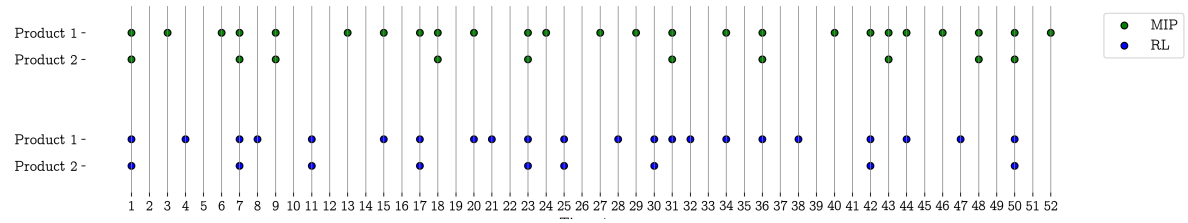


Figure 8.10: Comparing the ordering schedules for the RL and MIP approaches for the 'Base Instance'. The dots in the plot represent replenishment periods. Start inventory levels are set to zero.

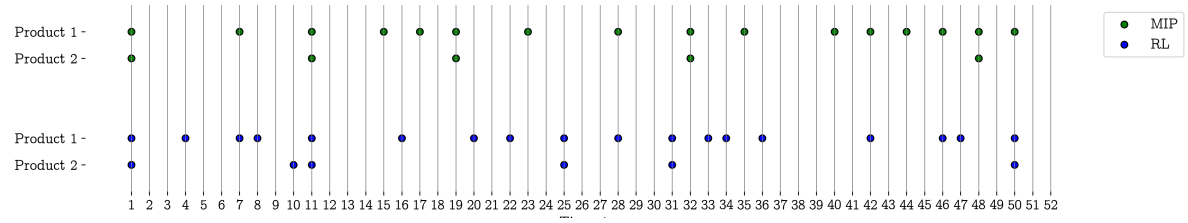


Figure 8.11: Comparing the ordering schedules for the RL and MIP approaches for the 'Double Major' cost instance. The dots in the plot represent replenishment periods. Start inventory levels are set to zero.

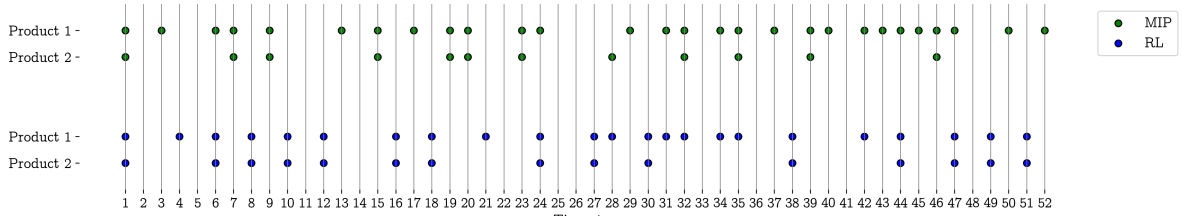


Figure 8.12: Comparing the ordering schedules for the RL and MIP approaches for the 'Half Minor' cost instance. The dots in the plot represent replenishment periods. Start inventory levels are set to zero.

Chapter 9

Future Work

Both the Mixed Integer Programming (MIP) approach and the deep Reinforcement Learning (RL) approach show promising results for solving the Stochastic Joint Replenishment Problem (SJRP) with seasonal demand. The MIP approach is superior for larger instances, while the RL approach is better for small instances. Nevertheless, there is still room for improvement with regard to the performance. In addition, there are several extensions of the problem formulation that could be interesting to investigate in further research. Future research opportunities with regard to both the problem formulations and method improvements are outlined in this chapter.

The problem considered in this thesis has some simplifying assumptions which could be interesting to modify in order to make the problem more realistic. Firstly, lead time is assumed to be zero, which is unrealistic. In a realistic setting, this is positive, and might also be stochastic. Hence, future work could focus on incorporating a positive or even stochastic lead time providing a more accurate representation of practical scenarios.

The problem formulation also assumes that there are no restrictions on the warehouse capacity. However, in reality, companies often do not have unlimited storage capacity. Hence, an upper bound on inventory levels could be interesting to add in future work.

Moreover, the problem formulation used in this thesis does not have any capacity constraints. In a realistic setting, the ordered quantities often need to fit into a certain number of trucks. Such a constraint could be beneficial to add in future extensions. In the previous work presented in Gravdal (2022), quantity discounts were considered in the problem. This aspect is not addressed in this thesis and could add considerable value to future research.

Integrating the correlations in demand for different products into the SJRP may also add value to the problem as it could improve replenishment strategies. For instance, if two products have positively correlated demand, then an increase in demand for one product could signal a corresponding increase in demand for the other. This could influence the joint replenishment

decision and timing, potentially leading to cost savings and better service levels.

Since the achieved service levels in Section 8.1 are much higher for the MIP than what is set by the customer, performing a more extensive parametrization test of the interval service levels could potentially improve its performance. There is a possibility that one could find a parametrization that better suits the curve of the optimal internal service levels. If so, it could lead to identifying a parameter value that allows for a reduction in safety stock most likely reducing the costs. Another interesting experiment would be to investigate lowering the customer service level in the model itself, in order to achieve a service level closer to the customer-specified service level.

The RL approach shows promising results in instances with two products, however, the service level is in some instances lower than the customer service level. This might be avoided by further tuning of the agent, for instance by shaping the reward function. Shaping the reward could be done by increasing the shortage costs part of the reward function for each product, incentivizing to less shortage costs. As discussed in Section 8.1, such a tuning led to some improvement of one of the test instances, so it could be developed further in future work.

Furthermore, it could be interesting to fine-tune the state of the agent. The method currently uses the demand for the past 13 periods and the inventory level of the current period as the state. One possible improvement would be to include forecasts in the state space, to provide the agent with a more thorough description of the current state. Since the demand is seasonal, it could also be interesting to add a component that encapsulates the seasonal period and see if this improves the learning process.

In this thesis, the actor network used in the RL approach was pre-trained using supervised learning as discussed in Section 5.2.5. This strategy helped the RL agent explore more of the promising action space. Another variant could be to initialize the replay buffer used in the DDPG algorithm with promising states and actions from the pre-training phase. Since minibatches from this replay buffer are used to refine critic evaluations, initializing it with promising values ensures exposure of the critic to 'positive cases'. Adopting this approach instead of per-training the actor, could result in an actor that balances exploration of the action space more dynamically, while still benefiting from the critic's assessments of previously identified promising states and actions.

A prominent field for future work is to investigate strategies to make the RL approach perform well also for larger product instances. The RL agent struggles to learn for instances with more than two products due to the computational complexity. Both the state space and action space increase exponentially with the number of products.

One possible way to address this issue might be by performing a wider analysis of exploration strategies. The performance of the RL agent is tightly coupled to how well it explores the action space. This becomes even more important as the number of products increases. Thus, finding an efficient way of exploring new states while also exploiting existing policy could make the RL agent perform well for larger product instances.

It would interesting to explore some of the modified versions of the DDPG in future work. One example of such a modification is the Twin Delayed Deep Deterministic policy gradient developed by Fujimoto et al. (2018). Exploring TD3 could be of interest as it introduces several advancements over DDPG, which could potentially result in improved performance. TD3 incorporates features like clipped double Q-learning to handle overestimations of Q-values, and delayed policy updates to reduce the variance of policy and value updates (Fujimoto et al., 2018). Such improvements could help the RL agent learn more efficiently, and possibly achieve better outcomes in the joint replenishment problem.

Another promising extension of DDPG to investigate further is the Multi-Agent Deep Deterministic Policy Gradient (MDDPG), which was introduced by Lowe et al. (2017). This method is designed for use in a multi-agent setting, where multiple agents cooperate in order to achieve a common goal. The key idea is that each agent's critic is fed with extra information about the policies of the other agents, while the actor only has access to their local information (Lowe et al., 2017). This could be applied to the SJRP by assigning a distinct actor to each product. This might potentially improve the understanding of each product's unique demand dynamics. However, efficient coordination of multiple agents is challenging and must be addressed to successfully utilize this method in the context of SJRP.

Chapter 10

Concluding Remarks

This thesis aimed to develop two solution methods for solving the Stochastic Joint Replenishment Problem (SJRP) with seasonal demand. The problem formulation was developed in collaboration with Visma, who contributed with practical insights and real-world applicability. The developed methods determine which products should be ordered and the order quantities at each time period. The objective is to minimize the sum of holding, setup and shortage costs.

The SJRP is a well-studied problem. However, the extension addressing non-stationary or seasonal demand patterns is considerably less explored. This is worth noting, given the fact that many products underlie these patter. Such demand patterns affect the ordering policy of the products and it is therefore relevant to develop solution methods to solve the problem in this setting.

The two solution methods developed are a mathematical Mixed Integer Programming (MIP) approach and a Reinforcement Learning (RL) approach. Both methods were tuned to enhance performance, and a comparative analysis was conducted to test the two approaches on different product instances with different demand patterns and cost distribution. The MIP approach serves as a benchmark for the RL approach.

The MIP approach is an extension of the previous work in Gravdal (2022) that handles the uncertainty in demand by including a safety stock for each product. This method is dependent on accurate forecasts as inputs, both for determining the optimal safety stock and actions. A comparative analysis of several forecasting methods was therefore conducted, and the Holt-Winter method was chosen as it showed superior performance.

The RL approach uses the state-of-the-art Deep Deterministic Policy Gradient (DDPG) to learn a robust ordering policy. The states are based on 13 periods of historic demands as well as the current inventory level, and the reward function is the negative of the total costs. The actor network utilized in the DDPG algorithm is pre-trained using supervised learning. This is done by

training the actor network on samples collected from the MIP model with deterministic demand and no safety stocks. The results show that this pre-training phase is crucial for the agent to learn in the stochastic RL environment, as it enables the agent to explore more promising states and actions from the start.

Our finding shows that the developed RL approach outperforms the developed MIP model in terms of total costs for instances of two products, except for one product instance with smooth demand in which it performs equally well. This is a significant improvement compared to what has been observed in the existing literature for RL solutions for solving the SJRP. For the four product case, the MIP approach is superior to the RL approach, showcasing the MIP's ability to tackle larger problem instances.

Unique to our approach for solving the SJRP using RL is that we have not imposed any limitations on the action and demand spaces, a scenario yet to be explored in the existing literature. Vanvuchelen et al. (2020) also solved the SJRP for two products using Proximal Policy Optimization (PPO), but with a uniform demand distribution. The non-stationarity of demand in this thesis, introduces further complexity into the state space and actions, making it more difficult to achieve an optimal policy. Furthermore, seasonal demand fluctuations can introduce non-linear, time-delayed rewards, making the RL learning process even more complex.

While RL has shown promising results for managing SJRP with two products, the increase in complexity and the nuances of reward function in the four-product case make the problem significantly more challenging for the RL agent. In these cases, the MIP model outperforms the RL agent. This calls for more sophisticated methods, perhaps integrating domain-specific knowledge into the learning process of the RL agent, to make it work even for more complex tasks. Future research could explore strategies such as reward shaping, exploration strategies, incorporating domain knowledge, or the use of more complex RL architectures to improve performance. Additionally, understanding the impact of demand seasonality and designing strategies to handle time-delayed rewards could be beneficial.

The MIP approach can in practice be used for larger instances of the problem. In the previous work of Gravdal (2022), a similar model was developed. This was able to solve instances with up to 192 products within a reasonable time. However, the addition of forecasting methods in the MIP model to tackle the uncertainty in demand significantly increases the computation time. Despite this, since the model only needs to be run one time, it is able to provide an ordering decision within a reasonable time in a real-world setting. It is reasonable to assume however that the MIP developed in this thesis can also be used in practice for larger instances.

It is worth noting that while the RL agent needs to undergo training before generating ordering decisions, it is very quick in providing decisions once trained. As long as the underlying demand structures of the products remain unchanged, the RL agent can be repeatedly utilized to generate new ordering decisions. This is in contrast to the MIP model, which does not require pre-training for ordering decisions but exhibits longer run times.

In conclusion, both the MIP approach and the RL approach demonstrate an ability to solve the SJRP with seasonal demand. The RL approach outperforms the MIP model in instances with two products, showcasing its effectiveness in handling demand stochasticity without restrictions on demand patterns or action space. This flexibility is a unique characteristic of the RL approach presented in this thesis. For now, the MIP approach proves to be more applicable in a practical setting, since it is able to solve larger instances of the problem within a reasonable time. Both the RL approach and MIP approach have room for further improvement and development, serving as motivation for future work.

Bibliography

Aggarwal, C. C. et al. (2018). Neural networks and deep learning. *Springer*, 10(978):3.

Arkin, E., Joneja, D., and Roundy, R. (1989). Computational complexity of uncapacitated multi-echelon production planning problems. *Operations research letters*, 8(2):61–66.

Atkins, D. R. and Iyogun, P. O. (1988). Periodic versus "can-order" policies for coordinated multi-item inventory systems. *Management Science*, 34(6):791–796.

Axsäter, S. (2013). When is it feasible to model low discrete demand by a normal distribution? *OR spectrum*, 35(1):153–162.

Axsäter, S. (2015). *Inventory control*, volume 225. Springer.

Balintfy, J. L. (1964). On a basic class of multi-item inventory problems. *Management science*, 10(2):287–297.

Bastos, L. d. S. L., Mendes, M. L., Nunes, D. R. d. L., Melo, A. C. S., and Carneiro, M. P. (2017). A systematic literature review on the joint replenishment problem solutions: 2006-2015. *Production*, 27.

Berling, P. and Marklund, J. (2013). A model for heuristic coordination of real life distribution inventory systems with lumpy demand. *European Journal of Operational Research*, 230(3):515–526.

Boctor, F. F., Laporte, G., and Renaud, J. (2004). Models and algorithms for the dynamic-demand joint replenishment problem. *International Journal of Production Research*, 42(13):2667–2678.

Boute, R. N., Gijsbrechts, J., Van Jaarsveld, W., and Vanvuchelen, N. (2022). Deep reinforcement learning for inventory control: A roadmap. *European Journal of Operational Research*, 298(2):401–412.

Box, G. E., Jenkins, G. M., Reinsel, G. C., and Ljung, G. M. (2015). *Time series analysis: forecasting and control*. John Wiley & Sons.

Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J., and Zaremba, W. (2016). Openai gym. *arXiv preprint arXiv:1606.01540*.

Brooks, C. (2019). Introductory econometrics for finance.

Brown, R. G. (1967). Decision rules for inventory management.

Carnot, N., Koen, V., and Tissot, B. (2005). *Economic forecasting*. Springer.

Cattani, K. D., Jacobs, F. R., and Schoenfelder, J. (2011). Common inventory modeling assumptions that fall short: Arborescent networks, poisson demand, and single-echelon approximations. *Journal of operations management*, 29(5):488–499.

Croston, J. D. (1972). Forecasting and stock control for intermittent demands. *Journal of the Operational Research Society*, 23(3):289–303.

DeMatteis, J. J. and Mendoza, A. G. (1968). An economic lot-sizing technique, i: The part-period algorithm. *IBM systems Journal*, 7(1):30–46.

Ekanayake, N., Joshi, N., and Thekdi, S. A. (2016). Comparison of single-echelon vs. multi-echelon inventory systems using multi-objective stochastic modelling. *International Journal of Logistics Systems and Management*, 23(2):255–280.

Fujimoto, S., Hoof, H., and Meger, D. (2018). Addressing function approximation error in actor-critic methods. In *International conference on machine learning*, pages 1587–1596. PMLR.

Gonçalves, J. N., Carvalho, M. S., and Cortez, P. (2020). Operations research models and methods for safety stock determination: A review. *Operations Research Perspectives*, 7:100164.

Goyal, S. (1974a). Optimum ordering policy for a multi item single supplier system. *Journal of the Operational Research Society*, 25(2):293–298.

Goyal, S. and Belton, A. (1979). Note on" a simple method of determining order quantities in joint replenishments under deterministic demand". *Management Science (pre-1986)*, 25(6):604.

Goyal, S. K. (1974b). Determination of optimum packaging frequency of items jointly replenished. *Management Science*, 21(4):436–443.

Goyal, S. K. and Satir, A. T. (1989). Joint replenishment inventory control: deterministic and stochastic models. *European journal of operational research*, 38(1):2–13.

Gravdal, A. (2022). The joint replenishment problem with deterministic seasonal demand and quantity discounts. Unpublished, Norwegian University of Science and Technology.

Hachaïchi, Y., Chemingui, Y., and Affes, M. (2020). A policy gradient based reinforcement learning method for supply chain management. In *2020 4th International Conference on Advanced Systems and Emergent Technologies (IC_ ASET)*, pages 135–140. IEEE.

Harris, F. W. (1913). How many parts to make at once.

Holt, C. C. (1957). Forecasting trends and seasonals by exponentially weighted moving averages. *ONR Memorandum*, 52(52):5–10.

Hong, S.-P. and Kim, Y.-H. (2009). A genetic algorithm for joint replenishment based on the exact inventory cost. *Computers & Operations Research*, 36(1):167–175.

Huang, M.-G. (2009). Real options approach-based demand forecasting method for a range of products with highly volatile and correlated demand. *European journal of operational research*, 198(3):867–877.

Hyndman, R. J. and Athanasopoulos, G. (2018). *Forecasting: principles and practice*. OTexts.

Jiang, C. and Sheng, Z. (2009). Case-based reinforcement learning for dynamic inventory control in a multi-agent supply-chain system. *Expert Systems with Applications*, 36(3):6520–6526.

Johansen, S. G. and Melchiors, P. (2003). Can-order policy for the periodic-review joint replenishment problem. *Journal of the Operational Research Society*, 54(3):283–290.

Kaelbling, L. P., Littman, M. L., and Moore, A. W. (1996). Reinforcement learning: A survey. *Journal of artificial intelligence research*, 4:237–285.

Kaspi, M. and Rosenblatt, M. J. (1983). An improvement of silver's algorithm for the joint replenishment problem. *AIIE Transactions*, 15(3):264–267.

Kaspi, M. and Rosenblatt, M. J. (1985). The effectiveness of heuristic algorithms for multi-item inventory systems with joint replenishment costs. *International Journal of Production Research*, 23(1):109–116.

Kaspi, M. and Rosenblatt, M. J. (1991). On the economic ordering quantity for jointly replenished items. *The International Journal of Production Research*, 29(1):107–114.

Khouja, M. and Goyal, S. (2008). A review of the joint replenishment problem literature: 1989–2005. *European journal of operational Research*, 186(1):1–16.

Kot, S. and Grondys, K. (2011). Theory of inventory management based on demand forecasting. *Polish journal of management studies*, 3(1):147–155.

Lee, L. H. and Chew, E. P. (2005). A dynamic joint replenishment policy with auto-correlated demand. *European Journal of Operational Research*, 165(3):729–747.

Lillicrap, T. P., Hunt, J. J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D., and Wierstra, D. (2015). Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*.

Lowe, R., Wu, Y. I., Tamar, A., Harb, J., Pieter Abbeel, O., and Mordatch, I. (2017). Multi-agent actor-critic for mixed cooperative-competitive environments. *Advances in neural information processing systems*, 30.

Meisheri, H., Sultana, N. N., Baranwal, M., Baniwal, V., Nath, S., Verma, S., Ravindran, B., and Khadilkar, H. (2022). Scalable multi-product inventory control with lead time constraints using reinforcement learning. *Neural Computing and Applications*, 34(3):1735–1757.

Narayanan, A. and Robinson, E. (2006). More on 'models and algorithms for the dynamic-demand joint replenishment problem'. *International Journal of Production Research*, 44(2):383–397.

Narayanan, A. and Robinson, P. (2010). Evaluation of joint replenishment lot-sizing procedures in rolling horizon planning systems. *International Journal of Production Economics*, 127(1):85–94.

Peng, L., Wang, L., and Wang, S. (2022). A review of the joint replenishment problem from 2006 to 2022. *Management System Engineering*, 1(1):9.

Puterman, M. L. (1990). Markov decision processes. *Handbooks in operations research and management science*, 2:331–434.

Schulman, J., Wolski, F., Dhariwal, P., Radford, A., and Klimov, O. (2017). Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*.

Sewak, M. (2019). *Deep reinforcement learning*. Springer.

Silver, E. A. (1976). A simple method of determining order quantities in joint replenishments under deterministic demand. *Management science*, 22(12):1351–1361.

Silver, E. A. (1979). Coordinated replenishments of items under time-varying demand: dynamic programming formulation. *Naval Research Logistics Quarterly*, 26(1):141–151.

Silver, E. A. and Meal, H. C. (1973). A heuristic for selecting lot size quantities for the case of a deterministic time-varying demand rate and discrete opportunities for replenishment. *Prod. Inventory Manage.*, 2:64–74.

Silver, E. A., Pyke, D. F., Peterson, R., et al. (1998). *Inventory management and production planning and scheduling*, volume 3. Wiley New York.

Suetsugu, H., Narusue, Y., and Morikawa, H. (2020a). Periodic-review joint replenishment policy using multi-agent reinforcement learning.

Suetsugu, H., Narusue, Y., Morikawa, H., et al. (2020b). Branching deep q-network agent with reward allocation mechanism for joint replenishment policy. *(TOM)*, 13(2):36–49.

Sutton, R. S. and Barto, A. G. (2018). *Reinforcement learning: An introduction*. MIT press.

Syntetos, A. A. and Boylan, J. E. (2001). On the bias of intermittent demand estimates. *International journal of production economics*, 71(1-3):457–466.

Syntetos, A. A. and Boylan, J. E. (2005). The accuracy of intermittent demand estimates. *International Journal of forecasting*, 21(2):303–314.

Syntetos, A. A., Boylan, J. E., and Croston, J. (2005a). On the categorization of demand patterns. *Journal of the operational research society*, 56:495–503.

Syntetos, A. A., Boylan, J. E., and Croston, J. D. (2005b). On the categorization of demand patterns. *The Journal of the Operational Research Society*, 56(5):495–503.

Teunter, R. H., Syntetos, A. A., and Babai, M. Z. (2011). Intermittent demand: Linking forecasting to inventory obsolescence. *European Journal of Operational Research*, 214(3):606–615.

Thomopoulos, N. T. and Thomopoulos, N. T. (2015). *Demand forecasting for inventory control*. Springer.

Tiacci, L. and Saetta, S. (2009). An approach to evaluate the impact of interaction between demand forecasting method and stock control policy on the inventory system performances. *International Journal of Production Economics*, 118(1):63–71.

Vanvuchelen, N., Gijsbrechts, J., and Boute, R. (2020). Use of proximal policy optimization for the joint replenishment problem. *Computers in Industry*, 119:103239.

Visma (2018). Om Visma. `https://www.visma.no/om-visma/`. Accessed: 2022-10-10.

Viswanathan, S. (1997). Note. periodic review (s, s) policies for joint replenishment inventory systems. *Management Science*, 43(10):1447–1454.

Vollset, K. B. (2023). Forecasting correlation in demand. Unpublished, Norwegian University of Science and Technology.

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

Wang, L. and Chen, H. (2022). Optimization of a stochastic joint replenishment inventory system with service level constraints. *Computers & Operations Research*, 148:106001.

Watkins, C. J. C. H. (1989). Learning from delayed rewards.

Willemain, T. R., Smart, C. N., Shockor, J. H., and DeSautels, P. A. (1994). Forecasting intermittent demand in manufacturing: a comparative evaluation of croston's method. *International Journal of forecasting*, 10(4):529–538.

Winters, P. R. (1960). Forecasting sales by exponentially weighted moving averages. *Management science*, 6(3):324–342.

Yang, Y. H. and Kim, J. S. (2020). An adaptive joint replenishment policy for items with non-stationary demands. *Operational Research*, 20:1665–1684.

Zipkin, P. H. (2000). *Foundations of inventory management*. Singapore: McGraw-Hill.

Appendix A

Results from the Pre-Testing of Parametrizations

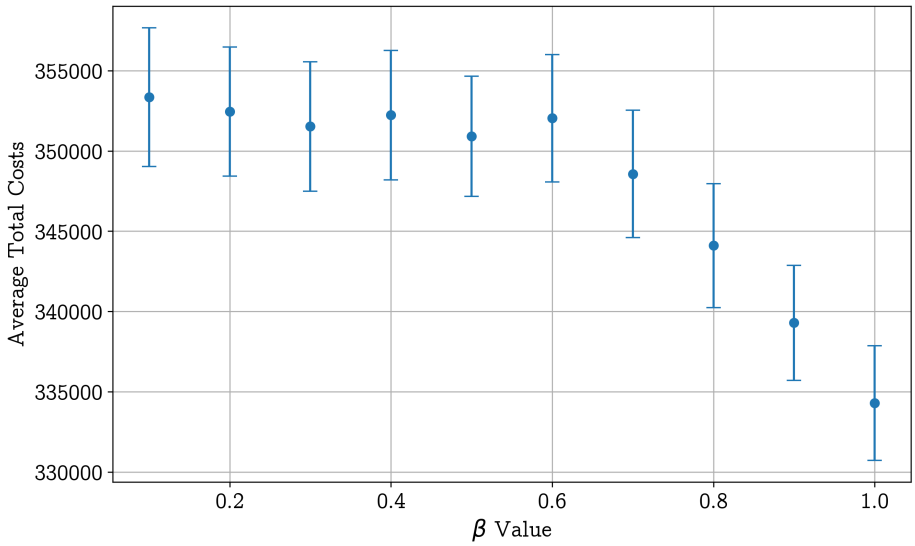


Figure A.1: Plot of the pre-testing of β values.

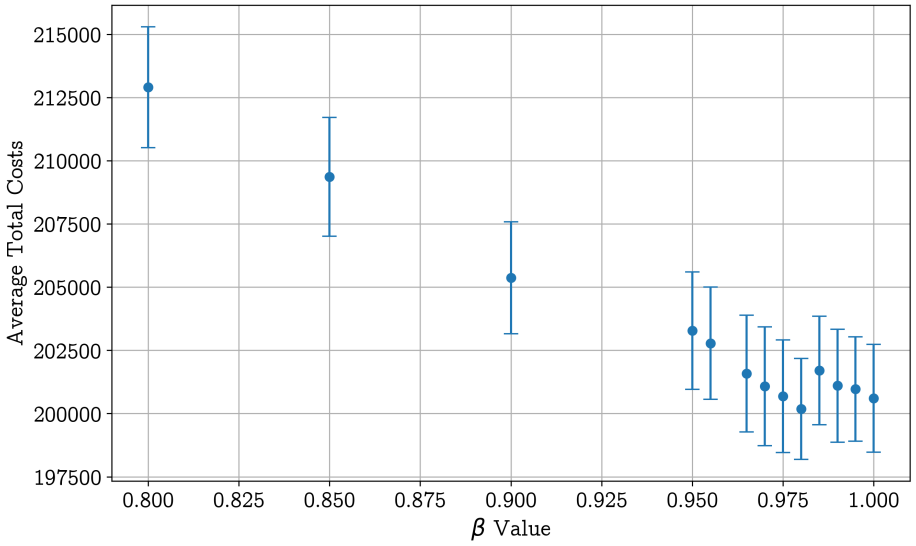


Figure A.2: Plot of closer pre-testing of β values.

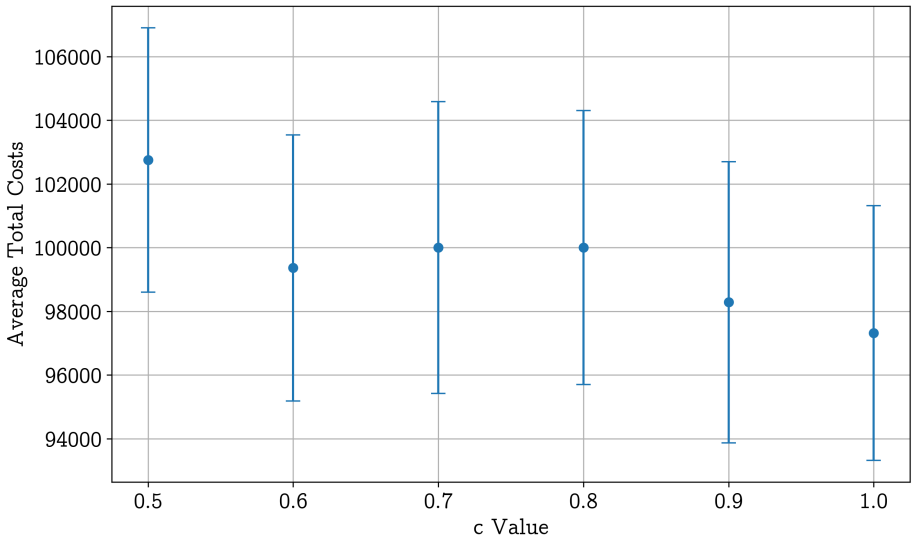


Figure A.3: Plot of pre-testing of c values.

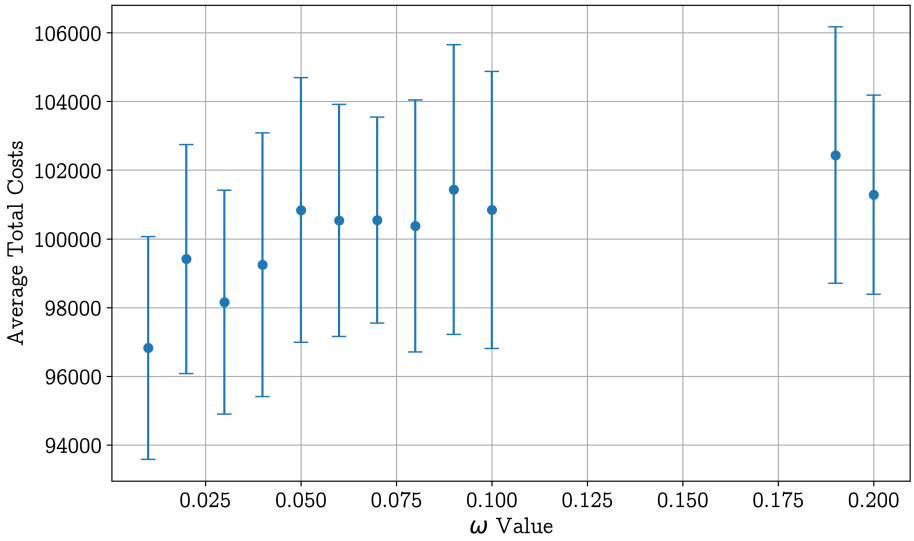
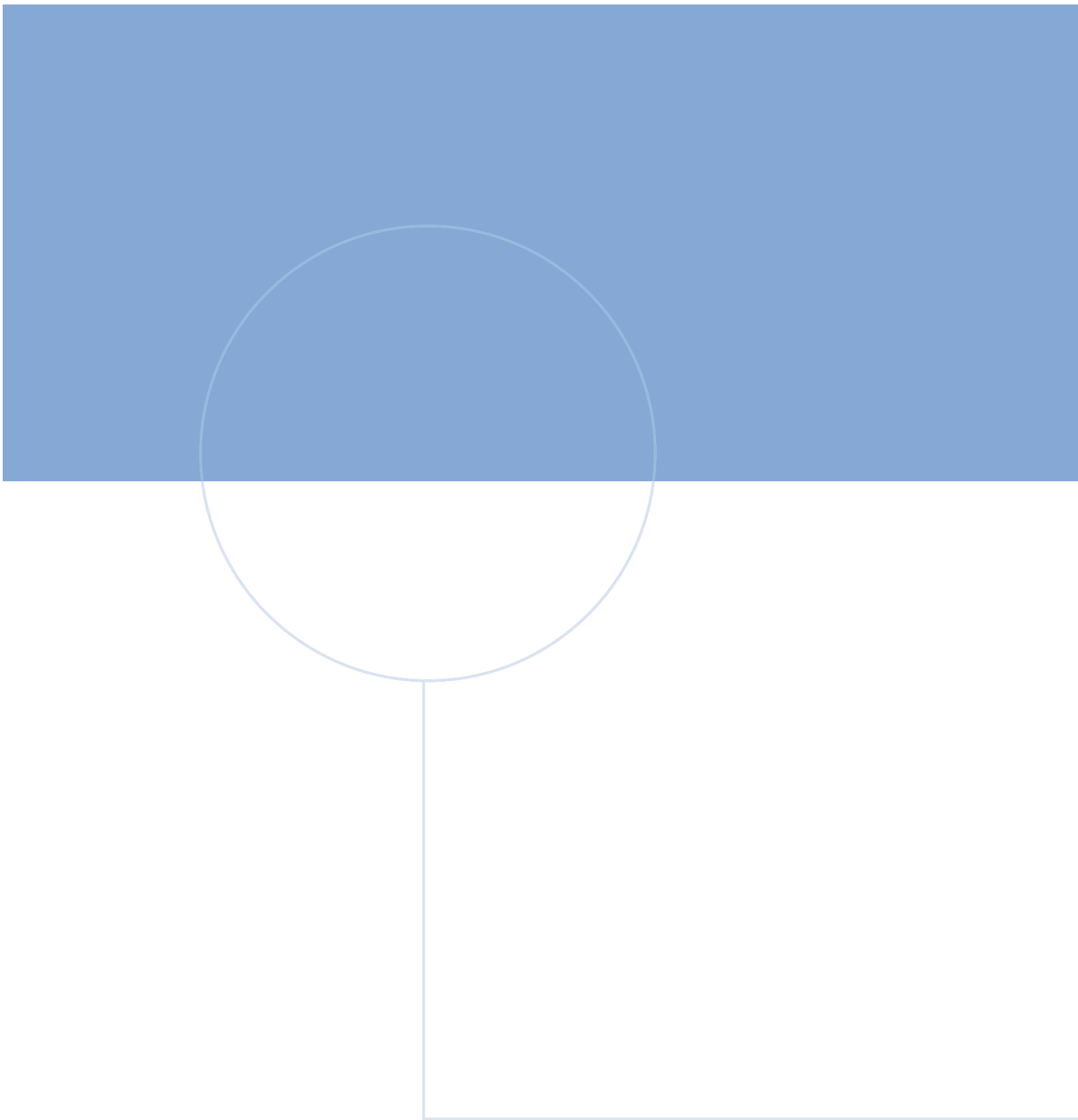


Figure A.4: Plot of pre-testing of ω values.



NTNU

Norwegian University of
Science and Technology