Master Thesis

Industrial Engineering & Management
University of Twente

Integrating operational and tactical decision-making in spare part inventory management

Daan Peters

January 2023

Supervisors
University of Twente
dr. E. Topan
dr. M.C. van der Heijden

Company
ir. K. Alizadeh

UNIVERSITY OF TWENTE.

Preface

This thesis was written as part of my graduation assignment for the master Industrial Engineering & Management at the University of Twente. and marks the end of my student life. I will always look back with fondness to this chapter of my life and I am thankful for everyone that has been a part of it.

First of all, I want to thank Kaveh Alizadeh for the time and the enthusiasm you have put into supervising me throughout writing this thesis. I started with little prior knowledge about the subject of this thesis, but throughout the months I have learned a lot and our weekly meetings have definitely played a huge role in gaining this knowledge and keeping me motivated.

Secondly, I would like to thank Engin Topan and Matthieu van der Meijden for their time and input. Your valuable feedback, either given during meetings and in writing, helped a lot in completing this research and writing this thesis as it is.

Lastly, I would like to thank my friends and family for their support, both personal and content-related, throughout all my years as a student.

Daan Peters
Utrecht, 19-01-2023

Management summary

Fokker Services (FS) is an independent aerospace company that, within its Component Maintenance & Availability (CMA) program, offers contracted clients access to inventory of a wide range of repairable spare parts. This research focuses on the exchange services contracts, where upon failure a client receives a replacement part. The failed component is sent to a repair shop, after which it is added to the spare parts pool.

The base stock levels of the spare parts pool are determined on a tactical level. On a day-to-day level, the operational planner at FS is able to perform interventions to (proactively) prevent and (reactively) solve backorders. In the current way of working, the base stock policy is made without considering operational interventions. This research aims to mitigate this knowledge gap, resulting in the following main research question:

How can the spare part inventory planning on a tactical level (i.e. base-stock levels) and on an operational level (i.e. interventions) be integrated?

Literature In literature, a wide range of approaches (e.g. heuristics, linear programming, decision rules) are found to integrating tactical and operational decision-making in inventory management. Most researches only consider one type of intervention (often lateral transhipments). Only one research was found that considers repairable spare parts. Vanvuchelen, Gijsbrechts, and Boute (2020) find that Deep Reinforcement Learning is a promising approach in solving sequential decision-making problems where little is known about the optimal policy structure. Next to these observations, FS prefers a data-driven and generalizable approach. Therefore, we choose to use Deep Reinforcement Learning as approach to integrating the operational and tactical decision-making.

Method We create two models to represent the two levels of decision-making. We use the Deep Q-Learning algorithm to train the neural networks of both our models. For both models, we use discrete event simulation to find the rewards (negative costs) and state transitions. We first train the operational model to make (near-)optimal decisions for the timing and types of interventions to perform, considering backorder and intervention costs. We consider time units of 14 days to find the general effect of performing interventions. First, we consider expediting repair jobs as sole intervention. Later, we demonstrate the possibility of adding interventions as a model extension. The operational neural network is trained for a wide range of component characteristics (e.g. demand rate, acquisition cost, repair lead time) and base stock levels. This allows for our model to be very scalable: a larger set of SKUs should not (significantly) impact the learning process.
With the learned interventions from the operational model, we train our tactical model to determine the base stock levels on an integrated level. Specifically, we determine the base stock levels that minimize the total costs (backorder, intervention, holding and service level penalty costs). These costs are found by simulating multiple consecutive periods with each period starting with the learned intervention.

Results Our operational planning model is trained to expedite repairs jobs to prevent and solve backorders. We compare our experiment results to that of a greedy heuristic. After performing different experiments, we find that in general more interventions are performed for SKUs with:

- high acquisition costs: preventing backorders is more important for expensive items,
- high demand rates: probability of a backorder is higher for higher demand rates,
- high repair lead times: the impact of expediting is larger for higher lead times,
- low probabilities for successful expediting: more interventions are required to successfully expedite a repair job.

We conclude that our operational model learns logical actions, meaning that the (generic) modelling approach works as intended. In terms of integration, we find that our tactical (rather: integrated) model is able to reduce total costs by balancing backorder and expediting costs with holding and SLA penalty costs. For finding the integrated base stock levels, we use both Deep Q Learning and Simulated Annealing, both giving comparable results. We find that performing interventions allows for lower base stock levels. Mainly, when compared to the greedy heuristic, we find that our model learns to hold less stock for SKUs with:

- low acquisition costs: backorders costs do not outweigh holding costs,
- low demand rates: for lower demand rates, it is more cost efficient to expedite once in a while than to hold more stock,
- high repair lead times: the repair lead time can be reduced more for items with longer repair lead times. SKUs with shorter repair lead times generally require less stock,
- high expediting success probabilities: the model learns to hold less stock for SKUs for which interventions have more effect.

For the acquisition costs, demand rate and expediting success probability, we find that base stock levels are reduced most for SKUs for which we (need to) perform more interventions. We see the opposite to be true when considering the repair lead time. A combination of larger expediting impact and the higher base stock levels found by the greedy heuristic result in more room for improvement for items with high repair lead times.

As a model extension, we add the possibility to hire a component from a third party. We have found that using a combination of interventions allows for a further reduction of total costs.

Discussion & conclusion This thesis describes an exploratory research into using Deep Reinforcement Learning for the integration of operational and tactical decision-making for a repairable spare part inventory setting. The MDP formulation used in DRL allows for the relatively easy addition of interventions. Our generic, scalable approach of training the operational model to perform interventions for multiple SKUs is found to be effective in finding good, logical actions. The integrated model finds base stock levels that enable a total cost reduction of 1 to 2 %, when compared to a greedy heuristic, by balancing backorder and intervention costs with holding and SLA penalty costs. The reduction in holding costs outweigh the backorder and intervention costs. Next to the total cost reduction, we found a 3 to 5.5% reduction in total inventory investment costs.

Some stability issues were found while training both planning models. We expect this instability to be a result of the use of deterministic policies in the Deep Q-Learning algorithm, which is known to lack robustness. Another limitation of this research is that we only consider the effect of performing interventions on a global level (using time units of 14 instead of allowing daily interventions).

Despite of the mentioned limitations, we find promising results that show that Deep Reinforcement Learning is a suitable approach for the integration of the two described planning levels. We believe that this thesis provides a solid base for future research into the subject.

On a practical level, our integration approach helps in taking operational impact into account when determining base stock levels. This can result in improved service levels, reduced number of interventions and reduced inventory investment costs.

Three contributions to literature are identified: 1) A new approach (using DRL) into the integration of both planning levels. 2) A generic training approach that allows one neural network to learn optimal actions for different types of components. 3) We have shown that our formulation allows for the relatively easy addition of interventions and the possible benefits of combining interventions.

For future research, we recommend to: 1) use a more stable algorithm (using stochastic policies); 2) reformulate the repair pipeline to limit the size of the state vector and allow for stochastic lead times; 3) include more than two interventions (as not seen before in literature); 4) explore the possibility of building one neural network to make both tactical and operational decisions.

List of abbreviations

A3C	Asynchronous Advantage Actor Critic
AC	Actor Critic
ADP	Approximate Dynamic Programming
AI	Artificial Intelligence
ANN	Artificial Neural Network
BSL	Base stock level
CMA	Component Maintenance & Availability
DCL	Deep Controlled Learning
DP	Dynamic Programming
DQL	Deep Q Learning
DQN	Deep Q Network
DRL	Deep Reinforcement Learning
EBO	Expected number of backorders
ERP	Enterprise Resource Planning
FS	Fokker Services
ILP	Integer Linear Programming
KPI	Key Performance Indicator
MDP	Markov Decision Problem
MILP	Mixed Integer Linear Programming
ML	Machine Learning
OPP	Operational planning professional
PPO	Proximal Policy Optimization
RL	Reinforcement Learning
SA	Simulated Annealing
SCT	Service Control Tower
SKU	Stock keeping unit
SLA	Service level agreements
SU	Serviceable Unit
UU	Unserviceable Unit
VFA	Value Function Approximation

Contents

1 Introduction

This chapter describes the research goal of this thesis, some background information about the motivation behind this research and the planned approach of how this research will be performed.

1.1 Problem context

Service providers make Service Level Agreements (SLAs) with their customers, in which they balance high service levels with low costs (Topan, Eruguz, Ma, van der Heijden, & Dekker, 2020). To achieve these goals, strategic, tactical, and operational decision-making have to be aligned. On an operational level, real-time performance data is checked daily to intervene when necessary (e.g. to prevent or solve backorders). Closely monitoring performances for after-sales service supply chains is accommodated in a so-called Service Control Tower (SCT).

In the current situation at Fokker Services, the operational planning is hierarchically dependent on the tactical planning. The tactical planning is based on aggregated demand and supply information per spare part such as demand rate, supply lead times, and acquisition costs. Within the scope of this thesis, only base stock levels are considered on a tactical level. Due to the stochastic nature of demand and supply, tactical decisions may not suffice to prevent possible operational interruptions. Therefore, on an operational level, some flexibility is created by performing reactive and proactive interventions. Examples of interventions taken by the Operational Planning Professional (OPP) are emergency shipments, expedited repairs, and renting components from external parties.

Reactive interventions are taken when alerts (e.g. direct backorders, high turnaround times) arise. As part of proactive interventions, the OPP can deviate from the tactical planning (i.e. hold more stock than the determined base stock level). An example of a situation where a proactive intervention might be taken is when a certain product is only demanded in pairs (multiples of two). Having only one item (or another uneven number of items) in stock would not make sense, even though on a tactical (system) level it would be optimal. Another example would be that an OPP prefers to have more parts on stock to minimize the chance of not being able to meet the demand of a certain customer that has had late orders in the past. For roughly 10% of the components Fokker Services holds, the OPP deviates from the base stock levels determined on a tactical level. More information about the different planning levels in the current situation at Fokker Services is given in Chapter 2.

1.2 Problem statement

The operational decisions (i.e. interventions) are taken manually by the OPP. These decisions are made to solve or prevent short-term inventory and/or delivery problems, and thus affect the performance of Fokker Services. Currently, the relation between the operational and the tactical planning is unclear, which might result in an sub-optimal synchronization between the two planning levels. For example, the operational flexibility enabled by interventions (Topan & van der Heijden, 2020) is not taken into account when deciding on the base stock levels. Therefore, for some components, Fokker Services might hold more inventory, and thus more costs, than necessary to obtain the SLAs.

Fokker Services wants to investigate possible data-driven ways to create and optimize an integrated operational and tactical planning.

1.3 Research questions

The main research question results from the described problem statement and is formulated as:

In what way can the spare part inventory planning on a tactical level (i.e. base-stock levels) and on an operational level (i.e. interventions) be integrated?

To obtain an answer to this question, some sub-questions have been formulated. These are listed and elaborated upon below.

To answer the main research question in a way that is useful for Fokker Services, a thorough understanding of the current spare part management at the company is required. Therefore, the first sub-research question is about the current situation at Fokker Services. This question will be answered in Chapter 2.

1. What is the relationship between the tactical and operational planning in the current situation at Fokker Services and what aspects can be improved?

According to Topan et al. (2020), limited literature is available on the topic of integrating operational and tactical planning. In addition, Petter (2021) describes the relationship between the two planning levels as a black box, indicating that the relation between the two planning levels is unknown. This research aims to contribute to literature by mitigating this knowledge gap.

In their literature review, Topan et al. (2020) found different types of relationships between the tactical and the operational planning. Most papers assume a hierarchical approach where the operational planning is dependent on the 'given' tactical planning. We are interested in the methods that focus on an integrated approach and therefore aim to create an overview of these methods and other relevant information in Chapter 3.1.

2. What kind of relations between the tactical and operational planning are described in literature?

From their literature review, Topan et al. (2020) found that integrated approaches outperform sequential approaches regarding costs and spare part availability. However, the computational complexity is high. Artificial Intelligence might be a method that can improve on this aspect.

In his research, Petter (2021) shows that Artificial Intelligence, more specifically Reinforcement Learning, can benefit operational decision-making in spare part management. He also indicates that larger problems are often only solvable using neural networks. Vanvuchelen et al. (2020) add to this that Deep Reinforcement Learning (DRL), which includes neural networks, is a promising topic when solving sequential decision-making problems, especially if little information is available about the optimal policy structure. Therefore, DRL is found to be a promising solution method for the goal of this research. More information about (Deep) Reinforcement Learning and how it relates to the methods found in literature will be given in Chapter 3.2 and 3.3.

3. What is (Deep) Reinforcement Learning and how does it position itself to the integrated-planning methods found in literature?

In Chapter 4, a (Deep) Reinforcement Learning method will be chosen and operationalized to integrate both planning levels at Fokker Services.

4. How can (Deep) Reinforcement Learning be used to integrate the two planning levels in the situation of Fokker Services?

Deep Reinforcement Learning is expected to be an effective tool for optimizing the integrated planning. Therefore, we are comparing the performances of the different solution methods in a selection of scenarios (for different data sets). The experimental design and experiments results for both models will be presented in Chapter 5.

5. How can we evaluate the proposed method and how does it perform compared to conventional methods?

Finally, in chapter 6, the conclusion, contribution to literature, limitation and recommendations of this research will be addressed. How all the research questions will be answered will briefly be discussed in the following subsection.

1.4 Methodology

A gap in literature, such as the topic of this research, is defined by Heerkens and van Winden (2012) as a knowledge problem. To solve a knowledge problem, the authors identified a research cycle which consists of eight phases, depicted in Figure 1.1.

Phases 1 to 3 have been discussed in the previous sections. In this section, the remaining phases will briefly be addressed.

For phase 4, the research design, a literature study is performed to explore possible ways of integrating the tactical and operational planning. Additionally, (Deep) Reinforcement Learning will be explored as a potential approach for integrating the planning levels. Following the findings from the literature study, a research design is created.

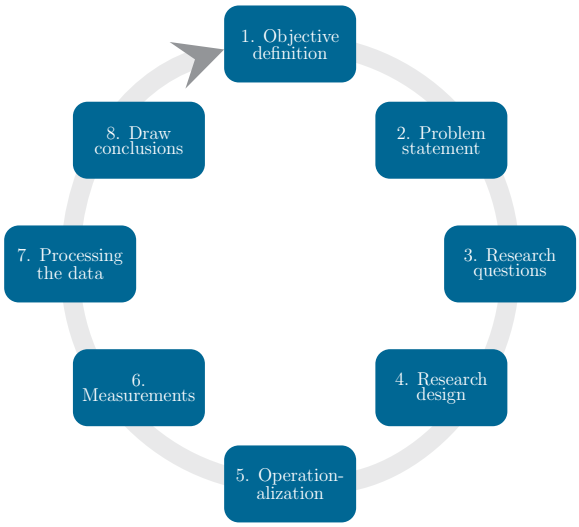


Figure 1.1: Phases of research cycle

The method proposed in the research design will be operationalized to be applicable for FS's current way of working. Subsequently, a plan will be set up on how to measure the performance of the method.

After the measurements have been executed, the validity of the measurements will be addressed. Subsequently, the results from the performance measurements will be analyzed and compared. From this analysis, the main research question will be answered and conclusions will be drawn.

2 Current system analysis

In this chapter, the current situation at Fokker Services is described. First, some background knowledge about the company is given. Subsequently, the current ways of working are elaborated upon. Additionally, a detailed description of the current tactical and operational planning is provided.

2.1 Fokker Services

At the age of 21, Anthony Fokker designed 'the Spin', a propellor airplane which he flew over the city of Haarlem. Eight years later, in 1919, Anthony registered The Netherlands Aircraft Factory. By 2025, Fokker was the largest aircraft manufacturer worldwide. After decades of both successful and difficult periods, multiple acquisitions and name changes, Fokker is now divided into different companies. This research is performed at Fokker Services.

Fokker Services is an Independent Aerospace Service Provider. Fokker Services creates tailor-made solutions for regional, commercial, and military aircraft. The mentioned services range from supplying spare parts and performing repairs to offering full maintenance and exchange programs. A large part of the customers of Fokker Services are within the Component Maintenance & Availability program.

For convenience, throughout the rest of this thesis, Fokker Services will be addressed as FS.

2.2 Supply chain

FS's supply chain is a two-echelon network. The highest level, the central warehouse, is located in Hoofddorp. The central warehouse delivers to the local warehouses, but also to customers directly. As FS operates worldwide, the local warehouses are located in Singapore and LaGrange, Georgia, U.S.. Lateral transshipments are possible between Singapore and LaGrange.

The commercial warehouse is an independent warehouse that serves the commercial market. Therefore, it has no SLA obligations. If necessary, the commercial warehouse can be used by the CMA-program to fulfill demand.

2.3 Component Maintenance & Availability program

FS provides repair services, spare parts, and reliability management for customers under the Component Maintenance & Availability (CMA) program. To enable high availability of components against reasonably low costs, FS holds a pool of spare parts of thousands of components for its CMA program.

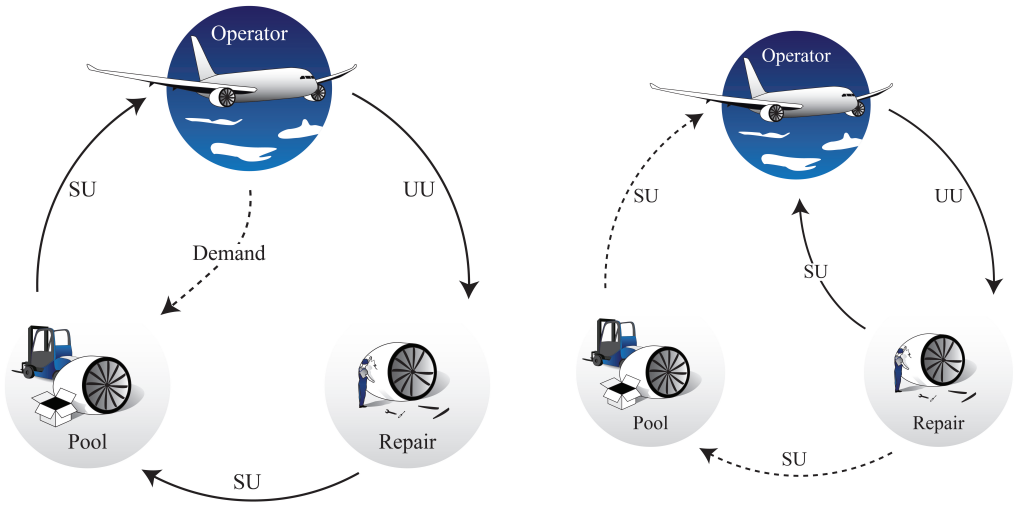
Having only one integrated service provider, customers of the CMA program benefit from relatively low prices and relatively fast delivery times. Additionally, customers benefit from predictable costs by using long-term contracts and risk-sharing frameworks. Within the CMA program, there are three types of service contracts: exchange, maintenance, and lease services. These are explained below and depicted in Figure 2.1.

The exchange service means that a customer can exchange a failed component, called an Unserviceable Unit (UU), for a functioning component from the spare parts pool, called a Serviceable Unit (SU). The UU is sent to the repair shop. When the UU is repaired, it is added to the spare parts pool.

The maintenance services contract encompasses a customer sending an UU to the repair shop and receives the component back after it is repaired. Only if the component is not repairable or if the agreed Turnaround Time (TAT) cannot be met, the spare parts pool is used to supply another component back to the customer. The latter is a special type of exchange service called performance exchange.

Some components in an aircraft are critical to flight safety and hence need to be replaced immediately upon failure, in order to prevent the grounding of the aircraft. To this purpose, FS offers to put such critical components on-site at the customer location under the so-called lease service contract. Hence, customers are assured of timely availability of such components. After replacement of the

component, the removed UU is sent to the repair shop, and an SU is sent to the customer's lease stock from the spare parts pool. For FS, the flows for the exchange and lease services are the same.



(a) Exchange/lease services

(b) Maintenance services

Figure 2.1: Component Maintenance & Availability programs

2.4 Tactical planning

Within the scope of this thesis, on a tactical level, only the base stock levels are considered.

There is a trade-off between the cost for holding inventory and acquiring a high service level. Therefore, base stock levels are carefully chosen. This is done by the tactical planners at FS. A greedy heuristic based on the METRIC framework (Sherbrooke, 1968) is used to determine these base stock levels. The tactical planning is revised every six to twelve months.

2.5 Operational planning

Decision-making on an operational level differs from the tactical planning in multiple ways (Topan & van der Heijden, 2020). First, operational decisions are made to have short-term impact. Tactical decisions are fixed for a longer time span. Second, different types of information are available. On an operational level, real-time information about current on-hand and pipeline stock can be used to make decisions, while for tactical planning often some type of aggregated historical information is utilized. Third, as the stock levels can differ daily, the system is often seen as having transient behavior instead of steady-state behavior.

2.5.1 Alerts

The OPP is alerted by tools connected to the Enterprise Resource Planning (ERP) system when it might be necessary for the planner to intervene. There are three common alert situations at FS (Petter, 2021; Ventevogel, 2020). The first is a direct backorder. In this situation, a customer requests a component that is not available. The OPP must take immediate action to get the required component to the customer as fast as possible.

Another alert is related to the Turnaround Time (TAT). The TAT is the time between the moment the failed component is removed from the aircraft until the moment the component is repaired. If the TAT of the repair of a component becomes high, it might be valuable to expedite the repair job to prevent future stockouts, and thus backorders.

Additionally, an alert is received when there is no stock on hand. This type of alert is similar to the direct backorders alert. Although it is not as urgent as direct backorders, quick actions should often be taken, as there is a high probability of future stockouts.

2.5.2 Interventions

Two categories of interventions are defined in literature (Glazebrook, Paterson, Rauscher, & Archibald, 2015; Topan & van der Heijden, 2020): reactive and proactive. Reactive interventions have to be taken when a stockout occurs. Often, the costs included for these types of interventions are high as time is limited. Proactive interventions are taken to reduce the risk of future stockouts, e.g. when an alert arises. A hybrid method proposed by Glazebrook et al. (2015) is using proactive interventions upon reactive triggers (i.e. stockouts). Similarly, interventions can either be planned (periodic reviews) or unplanned (opportunistic reviews; additional reviews in the event of a stockout) (Topan & van der Heijden, 2020).

At FS, different interventions are used. The different types of interventions are briefly explained below. For more detailed information, we refer to Petter (2021).

Do nothing When an alert arises, the OPP first decides whether or not an intervention is necessary. In some situations, no action is taken.

Discard, store or repair This intervention is triggered by alerts at the repair shop. In case an unserviceable component arrives, the OPP has to decide whether to repair it or not. This decision is based on current inventory data. There are three options: 1. The component is discarded (e.g. if repair costs are too high); 2. The component is sent to the storage location for non-operational components; 3. The component is repaired and sent back to the customer, or to the spare part warehouse, dependent on the service contract.

Prioritizing/expediting repair The operational planner is able to ask the repair shop to accelerate the repair of a component. Whether or not expediting is possible is dependent on many external factors. When expediting is possible, how much it can be expedited (how much repair lead time remains) is also dependent on many factors. In some situations, the repaired component can be ready within the same day. FS has its own internal repair shop, where less than half of the repair jobs are performed. The remaining repair jobs are outsourced to external parties. Expediting repairs is the most commonly used intervention at FS.

External sourcing In some urgent situations, parts from an external source are required. There are two (expensive) options: vendor exchange and procurement of new components. In a vendor exchange, the CMA-program leases a component from a third party vendor until the backorder is solved. If a stockout for a certain component happens regularly, it might be economically beneficial to purchase a new component (and consequently hold more stock). External sourcing is the second most commonly used intervention.

Emergency shipment When time is of the essence, e.g. when an airplane cannot fly due to a failed component, an emergency shipment can be performed. A spare part is then quickly transported from the spare part pool to the customer.
When the required component is not available in the spare part pool, a special type of emergency shipment can be performed: an emergency drop shipment. If the item is available at the repair shop, the repair can be expedited and the part can be transported to the customer immediately without first going into the spare parts pool. If no item is available at the repair shop, a new item can be purchased/leased and transported directly to the customer.
Both types of emergency shipments are rarely used.

Cannibalization This is a fast and cheap intervention, where FS uses partial components from spare parts in the warehouse to repair a component. Generally, this only happens at the internal repair shops. Although it is a cheap intervention, it is not commonly performed due to availability of components in the warehouse.
A different form of cannibalization is taking a component from an aircraft that cannot fly due to other failed components. However, as FS Services does not own aircraft themselves, this is only possible if the customer is able to perform this intervention. This happens only a few times per year.

Lateral transshipments If an alert arises due to a stockout at the CMA warehouse (i.e. the spare parts pool), parts can be relocated from one warehouse to another, or the component can be leased from another operator.

Interchangeable parts Some components are interchangeable, meaning that two different components are similar enough to replace one another (e.g. an older version of the same component). If possible, the OPP can choose to send a different item to the customer to replace the failed component.
Interchangeability can be one-way or two-way. The former means that component A can replace component B, but component B cannot replace component A. The latter means that component B can also replace component A.

Combining interventions Interventions are regularly combined, which can have multiple reasons. First, a combination of interventions increases the probability that a component will be available soon. Another reason is that it can be cheaper to combine interventions, for example expediting a repair to keep the duration of a vendor exchange at a minimum.

2.6 Interaction between the planning levels

Base stock levels are determined based on aggregated data without considering operational flexibility. Therefore, on an operational level, base stock levels might be higher or lower than optimal.

The tactical planners determine the base stock levels for all components. In approximately 10% of the cases, a correction from the operational planner is required. These corrections are reviewed during the next tactical update, where the tactical planners can either disregard or follow the corrections.

2.7 Summary & conclusion

FS's supply chain includes multiple locations in a two-echelon network. In this thesis, we only consider the central warehouse as this is the main office, and focus on the deliveries to customers directly as this is the main practice. Therefore, the supply chain considered throughout this thesis is a single-echelon, single-site network.

FS has different types of contracts with customers within the Component Maintenance & Availability program. To keep high service levels, FS determines base stock levels on a tactical level following a greedy heuristic based on the METRIC framework. On an operational level, interventions are performed to prevent and solve short-term shortages of stock. The two most common interventions are expediting repair jobs and vendor exchanges (hiring a component from a third party).

The greedy heuristic used to determine base stock levels does not consider the effect of interventions. As the operational flexibility enabled by performing interventions can prevent backorders, this effect could be taken into account when making tactical decisions, i.e. determining base stock levels. In previous research at FS, the relationship between operational and tactical decisions are described to be a black box. Therefore, we are interested in the literature on the integration of the two mentioned planning levels.

3 Literature review

Following the second sub-research question, in Section 3.1, we gather methods found in literature that integrate the tactical and operational planning. In Section 3.2, we explore what Reinforcement Learning is and its many applications. Subsequently, in Section 3.3, we explore what Deep Reinforcement Learning is and how it can be applied to inventory management.

3.1 Integrated planning

First, we perform a literature review on the relation between the two planning levels and some topics related to the integration of determining base-stock levels (tactical) and performing interventions (operational).

3.1.1 Relation between tactical and operational planning

In our literature review on the relation between the two planning levels, we build upon and extent the recent literature review of Topan et al. (2020). Our review follows the same distinction between three types of relations between the two planning levels: hierarchical, sequential and integrated.

Hierarchical relation
The first considered type of relation between the operational and tactical planning level is a hierarchical relation. Within this type of relation, only operational decisions are considered. The tactical parameters, e.g. base stock levels, are considered to be fixed.

Caggiano, Muckstadt, and Rappold (2006) consider a two-echelon reparable service parts system. Their proposed solving method is a linear programming model, with the decision variables being the number of units shipped via regular transport and the number of units shipped via expedited transport from the central to the local warehouse. Additionally, an effective heuristic is proposed, to enable the solving of larger problems. The starting inventory levels for the experiments are given. They found that significant economic value can be achieved by integrating repair and inventory allocation decisions.

Chua, Scudder, and Hill (1993) consider optimal batching of items that are waiting for repair at a (single-echelon) repair shop. Six policies have been developed and evaluated. Through a simulation experiment, the SP-WBPT (Spares weighted – Weighted Batch Processing Time) policy was found to perform best. The experiments consider different arrival rates, with no initial inventory levels.

Topan and van der Heijden (2020) research operational decision-making including both reactive and proactive interventions. These interventions include lateral transshipments and emergency shipments. The authors formulate a Mixed Integer Programming model to find the optimal solution. They find that their approach leads to a significant cost reduction.

Kazemi Zanjani and Nourelfath (2014) address external maintenance via coordination of spare part order quantities and delivery due dates. Both a deterministic and a stochastic programming model are created. The stochastic programming model is found to outperform the deterministic model. Within the experiments, the initial inventory levels are fixed. As suppliers, customers, and the maintenance department are considered, a multi-echelon model is created. However, decision-making is only done on one level.

Sequential relation
For sequential relations, first the tactical decisions are determined (optimized). Afterwards, with the now fixed tactical parameters, operational decision-making is optimized. The optimization step on a tactical level distinguishes the sequential relation from the hierarchical relation. The latter only focuses on the operational decision-making

Axsäter (2003) created a decision rule for lateral transshipments (pooling), given some reorder points and batch sizes. These tactical decisions are determined by simulation, but a reasonable simplification is determining these based on the no-pooling case. The decision rule should minimize costs under the assumption that no further transshipments will take place. This process is repeated

to approximate the optimal solution. This approach is found to be computationally efficient and to perform well. Therefore, this decision rule is particularly promising for complex situations.

Çömez, Stecke, and Çakanyildirim (2012) designed an optimal inventory sharing policy. Their policy allows for flexibility in accepting or rejecting transshipment requests. The transshipments in this research happen between independent retailers. Initial order quantities are determined for each retailer (tactical). The authors show that retailers' optimal transshipment policies are dynamic.

Integrated relation
Papers that consider an integrated relation aim for optimizing the operational and tactical parameters simultaneously or iteratively. The latter requires a feedback loop that enables two-way communication.

Frazzon, Israel, Albrecht, Pereira, and Hellingrath (2014) incorporated information from intelligent maintenance systems in optimizing lot sizes. To this end, they created a mixed integer linear programming problem (MILP) that combines the Capacitated Lot Sizing Problem and the Networks Flows Problem. No reaction protocols (i.e. interventions) are included in their research. The master plan (tactical level) is evaluated on a operational level through a discrete-event simulation, after which the master plan is adjusted. This process can be repeated until no adjustments are necessary. Hence, this is an iterative approach.

Glazebrook et al. (2015) propose hybrid lateral transshipments. In conventional policies, reactive lateral transshipments only meet the demand that has to be backordered. However, some of the costs involved in these shipments are to some extent independent of the amount of shipped stock (e.g. labor costs, fuel costs). Conventional proactive transshipments balance stock by reallocations on isolated points in time. The hybrid approach combines reactive triggers (i.e. an order cannot be met) with the assumption that no transshipments will be made until the next replenishment. This model is found to be especially effective for networks where the cost of transshipments is mostly determined by the travel time and distance, rather than the amount transported.

Grahovac and Chakravarty (2001) introduce a model for expensive, low-frequency items. The model only considers lateral transshipments as intervention. Through iteratively calculating stationary probabilities and updating the parameters after each iteration until the parameters have converged, we approximate the probabilities for backorders and emergency shipments. It is found that lateral transshipments can reduce overall costs by 20%. This is not always achieved by a reduction in inventory, which was an unexpected conclusion.

Kukreja and Schmidt (2005) consider a single echelon system in which lateral transshipments are used to enable complete pooling of parts with lumpy demand. The model starts with initial values for the reorder point and order-up-to level (both tactical decisions). It then iteratively runs the simulation to find the configuration that minimizes the costs while adhering to the required service level. Pooling is considered to influence the stock levels on a daily basis. It was found that pooling can create a significant cost reduction.

Kutanoglu and Lohiya (2008) propose a mathematical (ILP) model that combine base-stock level decisions with transportation modes decisions. The transportation modes differ in delivery time and costs. The problem is formulated as a single-echelon, multi-facility system. It is shown that significant benefits can be obtained from integrating the inventory and transportation decisions.

Ormon and Cassady (2004) formulated a model that includes setting initial stock levels, expediting repairs and cannibalization. Decision trees are created to support the logic of the model. Through a simulation, the following decision variables are determined: the initial number of spare parts, the number of machines that should be operational, and at which locations cannibalization actions are allowed. A statistical analysis is subsequently performed, e.g. to determine the average number of expedited repairs. These findings are used to determine the required maintenance investment. Thus, the operational impact is taken into account when performing tactical decisions.

Salman, Cassady, Pohl, and Ormon (2007) research the impact of cannibalization on fleet performance, balancing fleet readiness and total maintenance costs. The maintenance costs are minimized with the constraint that the average fleet readiness is above a certain value. A discrete-event sim-

ulation is used for evaluation. They found that, while not guaranteeing an optimal solution, this intervention yields a significant readiness benefit for a fleet at a relatively low cost.

3.1.2 Other related literature

Some additional literature has been obtained that is relevant to the topic of this research. These will be addressed in this section.

Integration of strategic and tactical planning
A different type of integration found in literature is the integration of the strategic and the tactical level. Candas and Kutanoglu (2007) found that integration of network design and stock levels, in comparison to the conventional sequential method, enables a cost reduction while maintaining the same service levels (or increased service levels for the same costs). Network design involves the strategic decision-making behind for example the number of facilities and the locations of the facilities. The decisions behind the stocking levels and their fill rates are on a tactical level. The benefits of integration of these two levels of decision-making increases significantly when dealing with high inventory costs (e.g. caused by expensive items and high service levels).

Wu, Hsu, and Huang (2011) attempt to solve a design problem for a spare part logistic system. It integrates logistic network design, part vendor selection, and transportation modes selection. This is a different type of integration than handled in context of our research. However, the approach could be of value to our research as well. Five algorithms have been proposed. Two algorithms simultaneously considered all design factors, whereas the remaining three first create a near-optimal logistic network and subsequently find optimal combinations for part vendor and transportation modes. The best performing algorithm is a combination of a Neural Network, Genetic Algorithm and Tabu search (NN-GA-Tabu). The authors indicate that the NN-GA-Tabu might be useful in solving other comprehensive space search problems.

Difference in objectives
There is a difference in objective between the operational and tactical planning level. The tactical planning is focused on the more long-term impacts of decision-making, with its main focus on costs. The operational planners are often focused more on the short term, where adhering to SLAs is more important than keeping the costs at a minimum. Multiple approaches to balancing different objectives have been found in literature.

An example of dealing with opposing objectives is found in Topan and van der Heijden (2020). The goal of this research is to minimize downtime. However, the authors do not want to minimize downtime at any cost. Therefore, they minimize the total shipment and downtime cost, translating the duration of downtime into costs.

Ormon and Cassady (2004) use a different approach. They solve a stochastic optimization model by optimizing one objective and converting the other objective into a restriction. In their case, there are two options. First, they minimize the investment (costs) while keeping the production capacity at a specified minimum value. Additionally, they maximize the production capacity while meeting budget constraints. This approach, also called the epsilon-constraint method (Du, Chen, Quan, Long, & Fung, 2011), is commonly used in literature. Generally, the costs are minimized while the SLA should be met.

In previous research at FS, Petter (2021) follows the example of Topan and van der Heijden (2020), linking backorders to costs. A clear cost (or reward) function is required for implementing Reinforcement Learning. More information about Reinforcement Learning is given Section 3.2.

Future research directions
In a literature review on integrated Supply Chain planning (Parreira, Pires, & Frazzon, 2021), four reoccurring directions for future research were found: 1. Including stochasticity into the model's processes to accurately approximate real life; 2. Extending the models to assess models on a larger scale; 3. Developing more efficient solving approaches to handle the complexity of SC problems; 4. Evaluating the impacts of operational decision-making on a long term (i.e. tactical) level. Petter (2021) has shown that Reinforcement Learning can benefit operational decision-making, especially

when the exact relationship between the two planning levels is unclear (i.e. a black box). He mentions that for larger problems, neural networks are necessary to solve the problems. Combining neural networks and RL is called Deep Reinforcement Learning. Vanvuchelen et al. (2020) find that Deep Reinforcement Learning (DRL) is a promising topic when solving sequential decision-making problems, where little information is available about the optimal policy structure.

3.1.3 Discussion and conclusion

In literature, three different types of relationships between the operational and tactical planning are found: hierarchical, sequential and integrated. Within these types of relations, many different methods are used (e.g. heuristics, decision rules, linear programming). A summarizing overview of the literature described in the previous sections is given in Table 1.

Most papers only consider one type of intervention. Only a few papers consider two interventions. No papers were found that consider more than two interventions. This observation is likely due to the size and complexity of the problem when the sets of operational and tactical decisions increase. The main focus is put on lateral transshipments.

Most modelling approaches found in literature are specific to their situation. FS requested a method that is generalizable (e.g. which can be extended to other applications). Additionally, only one research considered a system with repairable spare parts, which is required for FS.

Given the two observations mentioned (few interventions and generalizability), this thesis aims to build a more generic approach to integration of the operational and tactical planning. FS prefers a data-driven approach.

Following these observations and the four research directions from Parreira et al. (2021), Deep Reinforcement Learning (DRL) seems to be a potential solution method. Artificial Intelligence allows for large and complex problems to be solved with computational efficiency. Another benefit is that no decision rules have to be formulated, as the DRL algorithm will be trained to make (optimal) decisions itself. Therefore, we aim to learn about the relation between the operational and tactical model from the actions learned by the algorithm, instead of vice versa. The model will learn the quality of decisions, regardless of the stochastic nature of the inventory system at FS. Additionally, DRL seems to be a new approach in the field of integrated decision-making in repairable spare part systems, adding to the literature within this field. To conclude, this research will further research the potential of DRL for creating an integrated inventory planning.

Table 1: Summary literature review on relation between tactical and operational planning

Ref.	Relation	Intervention(s)	Method	System
[1]	Hierarchical	Expediting	LP model + heuristic	Two-echelon, multi-item
[2]	Hierarchical	Batching	Heuristic approach for batching policies	Single-echelon, single item (multi-indenture)
[3]	Hierarchical	Transshipments, emergency shipments	Mixed Integer Programming	Two-echelon, multi-item
[4]	Hierarchical	-	Deterministic and stochastic programming for order quantity and equipment delivery decisions	Single-echelon, single-location, single-item
[5]	Sequential	Transshipments	Decision rule	Single-echelon, multi-location, single-item
[6]	Sequential	Transshipments	Optimal inventory sharing policy	Single-echelon, multi-location, single-item
[7]	Integrated (iterative)	-	MILP for lot sizes	Multi-echelon, multi-location, multi-item
[8]	Integrated	Hybrid lateral transshipments	Dynamic programming model. Reactive triggers with the assumption that no trans-shipments can be made in (near) future	Single-echelon, multi-location, multi-item
[9]	Integrated	Transshipments	Iterative calculation of stationary probabilities	Multi-echelon, multi-location, single-item
[10]	Integrated (iterative)	Pooling (transshipments)	Iteratively reducing inventory while staying above SLA	Single-echelon, multi-location, single-item
[11]	Integrated	-	ILP for assigning transportation modes to customers	single-echelon, multi-location, single-item
[12]	Integrated	Expediting, cannibalization	Simulation model using decision trees	Multi-echelon, single-item (multi-indenture)
[13]	Integrated	Cannibalization	Simulation	Single-echelon, single-item (multi-indenture)
[14]	Integrated	Expediting, vendor exchange	Deep Reinforcement Learning	Single-echelon, multi-item

[1] (Caggiano et al., 2006); [2] (Chua et al., 1993); [3] (Topan & van der Heijden, 2020); [4] (Kazemi Zanjani & Nourelfath, 2014); [5] (Axsäter, 2003); [6] (Çömez et al., 2012); [7] (Frazzon et al., 2014); [8] (Glazebrook et al., 2015); [9] (Grahovac & Chakravarty, 2001); [10] (Kukreja & Schmidt, 2005); [11] (Kutanoglu & Lohiya, 2008); [12] (Ormon & Cassady, 2004); [13] (Salman et al., 2007); [14] This thesis

3.2 Reinforcement Learning

Dynamic Programming (DP) is a method for solving sequential problems to optimality. However, complex, realistic problems quickly become too large to find the optimal solution for within reasonable computation times. The *curse of dimensionality* refers to the size that a problem quickly grows to when the number of variables increases.

Often, a guaranteed optimal solution is not necessary. Many methods can be used to approximate the optimal solution of a mathematical problem. These methods are gathered under the term Approximate Dynamic Programming (ADP). Reinforcement learning is one of these methods.

Reinforcement Learning (RL) is a class of Machine Learning (ML), which is a class of Artificial Intelligence (AI). AI enables computers to perform tasks that usually require human intelligence. ML is the part of AI that involve computer algorithms that improve automatically through using data and experience. There are three types of ML (van Heeswijk, 2022):

- **Supervised Learning:** computer algorithms learn patterns from data labeled by humans;
- **Unsupervised Learning:** computer algorithms learn patterns from unlabeled data;
- **Reinforcement Learning:** computer algorithms describe how to take actions in an environment to maximize some cumulative rewards.

RL is about learning what to do to maximize a reward (Sutton & Barto, 2018; Geevers, 2020). In contrast to the other types of ML, the learner is not explicitly told which action to perform in each state. In this section, RL will be explained.

3.2.1 Markov Decision Processes

To understand RL, we need to understand what a Markov Decision Process (MDP) is. An MDP is a framework to break down sequential decisions (van Heeswijk, 2022). These decisions are taken by a so-called *agent* (Sutton & Barto, 2018). At every time step t, where $t = 0, 1, 2, ..., T$, this agent interacts with its *environment* (also called interaction space) through an *action*, denoted by A_t. Through these actions, the *state* (S_t) of the environment changes. For all taken actions, the agent receives a *reward*, denoted by R_t. The goal of the agent is to maximize the total sum of rewards earned.

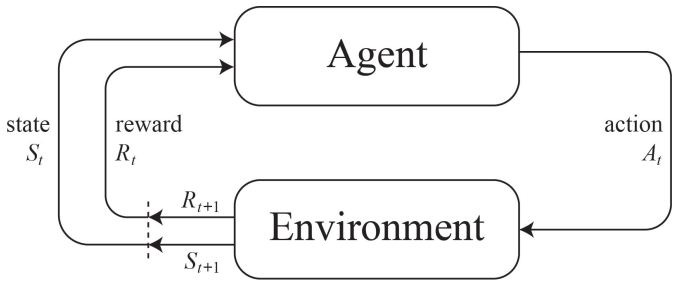


Figure 3.1: Agent-Environment interaction (Sutton & Barto, 2018)

Often, the sets of states (S), actions (A) and rewards (R) are not infinitely large. Within these *finite* MDPs, the future states ($s' \in S$) and future rewards ($r \in R$) have a certain probability of occurring at time t, given the current state and action. This leads to the following function:

$$p(s', r|s, a) \doteq Pr(S_t = s', R_t = r|S_{t-1} = s, A_{t-1} = a) \tag{1.1}$$

for all $s', s \in S$, $r \in R$, $a \in A$, where p defines the dynamics of the MDP. Additionally:

$$\sum_{s' \in S} \sum_{r \in R} p(s', r|s, a) = 1 \quad \forall s \in S, a \in A. \tag{1.2}$$

The points in time at which decisions are made are called *decision epochs* or periods (Puterman, 2014). After a decision, the system transitions from one state to another. We need multiple consecutive periods to learn how one action affects future states. A group of sequential decision epochs is called an *episode* (Sutton & Barto, 2018). An episode can end in a terminal state (e.g. the end of a game) or an episode can have a fixed length (a fixed number of periods) when there is no clear terminal state. The set of decision epochs is denoted by T, which can be either discrete or continuous, and either finite or infinite. An MDP can be denoted by a tuple $\langle S, A, R, T, \gamma \rangle$, where the latter denotes the discount factor which will be addressed in the next section.

Another characteristic of an MDP is the Markov property, which is also called the *memoryless property* (van Heeswijk, 2022). The Markov property means that decisions and transition probabilities do not depend on past events. They only use information from the current state. This characteristic enables the decomposing of sequential decisions.

3.2.2 Bellman Equation

The Markov property (see Section 3.2.1) enables the solving of finite optimization problems with sequential decisions (i.e. finite MDPs) to optimality through Dynamic Programming (DP). Richard Bellman introduced DP in 1953 with his Principle of Optimality: the idea that at whatever decision at period t, the subsequent decisions should proceed optimally for the sub-problem starting at $t+1$ (Dixit & Sherrerd, 1990). The Bellman optimality equation for deterministic environments is given by:

$$v_*(s) = \max_a \{r(s,a) + \gamma v_*(s')\} \tag{1.3}$$

indicating that the optimal value v_* of the current state s equals to the maximum of the sum of the direct reward for taking action a in state s and the value of the state s' resulted from taking action a in state s, multiplied by a discount factor γ. v is called the *state-value function*.

Due to the future rewards being dependent on the future actions that will be taken, decision policies are used to determine these future actions. A *policy* is a mapping from states to probabilities of selecting each possible action (Sutton & Barto, 2018). Following policy π at time t, $\pi(a|s)$ denotes the probability of taking action a given the agent is in state s. The state-value function under a certain policy is denoted by v_π. When the optimal policy is found, $v_\pi = v_*$.

3.2.3 Q-learning

There are many methods within RL literature that can be used to solve an MDP. One of these methods is Q-learning: an off-policy, model-free algorithm. Instead of the state-value function ($v(s)$) described in the previous section, Q-learning uses the *action-value function* denoted by $q(s,a)$, which are the values for taking a certain action in a certain state. These so-called q-values are stored in a table. As we store q-values ($q(s,a)$), not state-values ($v(s)$), we need to determine an action a' to find the value for the future state $q(s',a')$. Within Q-learning, we take the action a' for which $q(s',a')$ is highest. There are other methods in literature that use a policy for determining a'. One of these methods is SARSA, which is explained in Appendix A.

In Figure 3.2, the differences between the optimal state-value and optimal action-value functions are schematically depicted. The open circles represent a state and the black circles represent state-action pairs. The word *max* is added to the diagrams to indicate that the maximum value is taken. For the SARSA algorithm, the max operator would be replaced by a policy.

The q-values are found by iteratively visiting states and finding the effect (reward and state transition) of the actions taken. Often, states and actions are "visited" (i.e. their reward and state transition are found) by simulating the effect of taking an action in a given state. Through a high number of episodes and exploration, all states and actions are visited repeatedly to approximate their real value. Every time $q(s,a)$ is simulated, its value is updated. The formula for updating the values in the Q-table is given by

$$q(s,a) \leftarrow q(s,a) + \alpha * [r + \gamma \max_{a'} q(s,a') - q(s,a)] \tag{1.4}$$

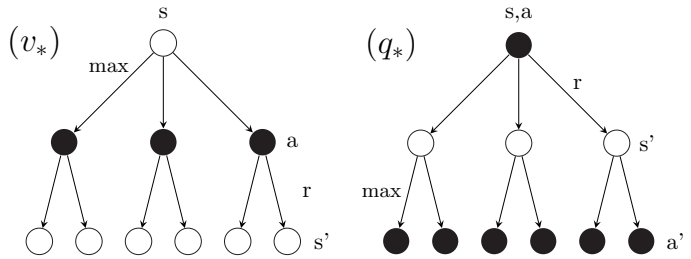


Figure 3.2: Backup diagrams v_* and q_* (Sutton & Barto, 2018)

where α is the updating parameter and can take values between [0,1]. If $\alpha = 1$, we replace the previous q-value with the new q-value, which means that we never converge (i.e. full exploration). The lower value α takes, the less we update the q-values and thus, the more we exploit the knowledge we have gained so far. By decaying α and using a high number of iterations, the q-values approximate their true values, after which $q*$ can be found. The initial values for the Q-values can be chosen arbitrarily (Sutton & Barto, 2018). Often, the initial Q-values are set to zero.

Both finite and infinite horizon MDPs can be used for Q-learning. For infinite horizons, γ has to be smaller than 1. In the rest of this thesis, we consider finite horizons.

3.2.4 Summary

Reinforcement Learning is an efficient approach to approximate the optimal solution for relatively complex problems. Using the Markov Decision Process framework and the Markov Property, sequential decision-making problems are broken down into subproblems. The MDP framework includes states S, actions A, rewards R, transitions T and an optional discount function γ. By iteratively visiting states, the Q-value of a state-action pair can be approximated and a policy that maximizes the total reward is created. Q-learning is a method that updates the values in a Q-table by the direct reward of action A_t and the value of the maximum Q-value for the future state-action pair $Q(S_{t+1}, A_{t+1})$.

The size of the Q-table exponentially grows with the number of possible states and actions. Including all possible (combined) interventions for all possible states is thus not only complex to model, but also computationally intractable.

Combining Reinforcement Learning with artificial neural networks, called *Deep Reinforcement Learning*, might provide a solution to this dimensionality problem. In the next section, the literature on this method is explored.

3.3 Deep Reinforcement Learning

In reinforcement learning, state-action combinations are repeatedly visited (i.e. taking an action in a given state is simulated such that reward and state transition are found) to update their expected action-values (q-values). Often, simulation is used to find the reward and the transition to the new state corresponding to taking an action from a certain state. The expected values of all state-action pairs are captured in a table. However, when solving large, realistic problems, the state-action space becomes too large to store all this information in a table. Additionally, a lot of computation time is required to visit all state-action pairs at least once, while multiple visits are required to make good estimations of their values. These limitations of RL are known as the generalization problem (Geevers, 2020).

Value Function Approximation (VFA) can be used to overcome the generalization problem. Instead of using a table, we can use functions that represent the value. Only the parameters of these functions have to be stored. Additionally, these functions can also estimate the values of unvisited states. Both mentioned limitations, related to the generalization problem, can be solved through VFA. There are many approaches to VFA, but one promising approach is Deep Learning, which uses Neural Networks to estimate the value of a state and action.

3.3.1 Neural Networks

When referring to a neural network in an AI context, we are talking about artificial neural networks, not biological neural networks (da Silva, Spatti, Flauzino, Liboni, & dos Reis Alves, 2016). An Artificial Neural Network (ANN) consists of multiple layers of artificial neurons: one input layer, one or multiple hidden layer(s) and one output layer. An adapted version of an artificial neural network from Geevers (2020) is given in Figure 3.3.

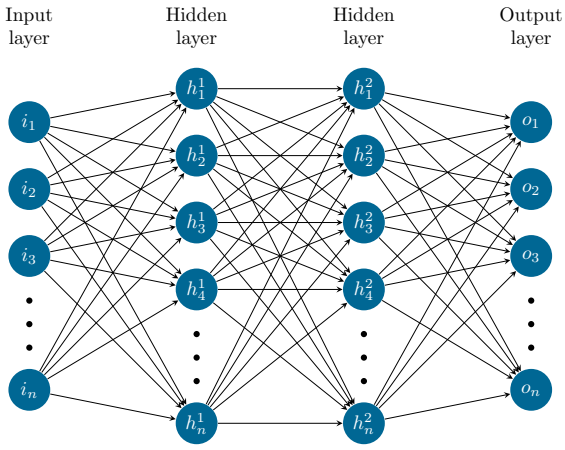


Figure 3.3: Artificial Neural Network

These neurons, or nodes, are nonlinear and perform simple functions. Figure 3.4 visualizes the relationship between neurons and is an adapted version of Geevers (2020) and da Silva et al. (2016). Note that, in this figure, the input x_i and output y can be between the input and a hidden layer, between two hidden layers, and between a hidden layer and the output layer.

A neuron receives input values x_i from the layer before, multiplies these with their corresponding weights w_i and takes the sum. A certain bias θ is often incorporated to artificially create a threshold to trigger an output. An activation function, f, can be included to limit the given output to a certain range of values, often $[0, 1]$. The result of this activation function is the output value of the neuron, y. The expression corresponding to the artificial neurons is:

$$y = f\left(\sum_{i=1}^{n} w_i * x_i - \theta\right). \tag{1.5}$$

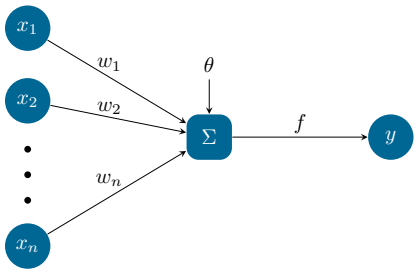


Figure 3.4: Neurons in a neural network

If Figure 3.4 represents neuron h_1^1 from Figure 3.3, the x_i's represent the input values (or in DRL: the state features). The output y represents the value of that neuron for the given inputs. If Figure 3.4 represents neuron o^1 from Figure 3.3, y likely represents the expected reward for a certain action, given the state features.

A neural network is trained by providing inputs with corresponding outputs. Based on the training data, the weights and biases are adjusted such that the neural network provides the right output for a given input. Training is done by minimizing a loss function (e.g. mean squared error). Within Deep Reinforcement Learning (DRL), the goal is to train the neural network to match state features (as input) with the optimal action (as output). For a discrete action space, we can let each node from the output layer represent one action from the action space. The node with the highest value, and thus the corresponding action, is chosen. For continuous action spaces, a node can be trained to approximate the value of an action.

3.3.2 Algorithms

There are different types of Deep Reinforcement Learning algorithms. Similar to Reinforcement Learning, DRL knows *value-function-based* or *policy-based* classifications. In value-function-based methods, the objective is to learn the value of an action given a state. Using a Neural Network, this is done by minimizing a certain loss function between the predicted and the target values. Often, simulation is used to generate the values that the neural network should learn. In policy-based methods, the objective is to maximize the expected reward of a policy. In DRL, the policy generally is a neural network that takes the state as input and outputs a probability distribution across the actions. Some algorithms combine value-function- and policy-based features. This combination is often referred to as the Actor-Critic architecture.

In this section, we will explain Deep Q-learning. For other algorithms, like PPO, A3C and DCL, we refer to Appendix B.

Deep Q-learning
In many RL methods, such as Q-learning, tables are used to store and update Q-values for all state-action combinations. Mnih et al. (2015) propose to use neural networks instead of tables to estimate the performance of certain actions. They developed the method Deep Q-learning (DQL), that uses a Deep Q-Network (DQN), which is found to outperform other DRL methods in several Atari 2600 games. The method is model-free, off-policy and value-function-based.

Value-function-based methods are often unstable. To increase stability, Mnih et al. (2015) include *experience replay* and iterative updates of the target network.

For each period in an episode, relevant information (e.g. state, action, reward, the new state) for the state transition is stored in a memory list. Experience replay uses the observations in the memory list as follows: when an update of the parameters (weights and biases) of the DQN is performed, a sample of random past experiences (i.e. observations from the memory list) is drawn to remove correlations in the sequence of observations. Experience replay requires that the laws of the environment are not changed such that past experiences are irrelevant (Geevers, 2020).

For the second improvement, the parameters θ^- (the weights and biases) of the target network $\hat{Q}$ are updated periodically (after a number of decision epochs or number of episodes). Through this

procedure, a delay is created between taking actions from the neural network and updating the parameters of that neural network. In the full pseudo code on DQL from Mnih et al. (2015) (see Appendix C), two networks are described: a target and an online network. The online network is used to sample actions from (e.g. taking the action corresponding to the output node with the highest value) and we update the parameters of the online network every step. The parameter updates are done by using the data from the random samples (as part of the experience replay). When we transition from one state s to state s', we find the direct reward r but we also want to take future rewards into account. The target network is used to find these future rewards, by taking the action that maximizes the action-value q from taking that action in state s'. See 1.6 for the equation for finding the updated action value for one observation.

$$y_{s,a}^{DQN} = r + \gamma \max_{a'} \hat{Q}\left(s', a'; \theta^{-}\right) \tag{1.6}$$

$Q\left(s', a'; \theta\right)$ and $\hat{Q}\left(s', a'; \theta^{-}\right)$ are found by using s' as input of the neural network and take the action a' that corresponds to the output node with the highest value (i.e. choose a' that maximizes $\hat{Q}\left(s', a'; \theta^{-}\right)$).

Note the similarities with Equation 1.4. The main difference is that in Equation 1.4 we subtract the old value of $Q(s, a)$ to update $Q(s, a)$. In DQL, we perform this update by minimizing a loss function (e.g. mean squared error) over $(y_{s,a}^{DQN} - Q(s, a; \theta))^2$.

3.3.3 Summary & conclusion

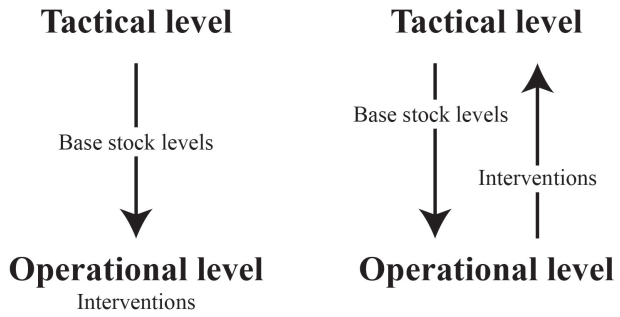
Reinforcement Learning uses tables to store Q-values for all state-action combinations. For systems with large state and action spaces, using tables becomes untenable. Therefore, function approximation methods can be used, such as artificial neural networks. Combining neural networks and Reinforcement learning is called Deep Reinforcement Learning.

There are many different algorithms withing DRL, which can have value-function-based and/or policy-based characteristics. Deep Q-Learning is a relatively simple value-function-based algorithm. Using experience replay and iterative updates of the target network improves the stability of DQL. The Actor-Critic architecture has a value-function-based critic and a policy-based actor. Asynchronous Advantage Actor Critic (A3C) and Proximal Policy Optimization (PPO) are two algorithms that are based on the AC architecture and generally perform well. Deep Controlled Learning (DCL) is a policy-based algorithm that performs especially well in stochastic environments.

We opt to implement Deep Q-learning for its relatively simple implementation. The goal of this thesis is to find the relation between operational flexibility and determining base stock levels. To reach this goal, implementing the most advanced algorithms is not necessary.

4 Model formulation

This thesis aims to create a method that integrates operational and tactical decision-making for the inventory planning of a repairable spare parts system. Within our scope, we look at interventions and base stock levels for the two planning levels respectively. Specifically, we want to determine the base stock levels (BSLs) that are (near-)optimal on an integrated level. Generally, the available methods in current literature (such as the METRIC framework (Sherbrooke, 1968)) determine base stock levels without taking the possibility of performing interventions into account (Figure 4.1a). Therefore, the number of backorders are generally over-estimated in systems where interventions are possible. This might lead to higher costs than necessary to reach a predetermined service level. Our model should determine the base stock levels while taking operational flexibility into account (Figure 4.1b). We do so by first optimizing when interventions should be performed for a wide range of inventory levels, and subsequently using these near-optimal interventions to determine what set of base stock levels minimize the total costs while adhering to service level agreements.



(a) Conventional (b) Integrated

Figure 4.1: Conventional versus integrated planning

From our literature review on the relationship between operational and tactical planning (see Section 3.1), we have found that Deep Reinforcement Learning is a promising method to approximate the optimal solution to complex problems. In this chapter, we describe our modelling approach.

4.1 General model description

As we are dealing with two different levels of decision-making (and in real life: different decision makers), we have created two models: an operational and a tactical model. First, we determine a range of base stock levels for each stock keeping unit (SKU) that the model should consider. Subsequently, the operational model is trained to optimize interventions for all SKUs and the corresponding base stock levels. With these interventions, the tactical model determines the base stock levels optimal on a integrated level. For both models, we use the Deep Q-Learning (DQL) algorithm to train the neural networks.

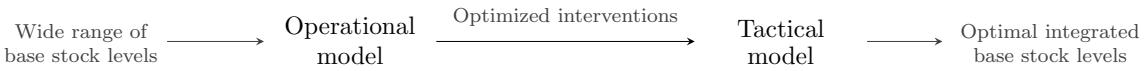


Figure 4.2: Overview model formulation

Both models consider the same single-echelon, single-indenture, single-site system. We consider the exchange services contract (see Section 2.3). To limit stochasticity in our model, repair lead times are assumed to be fixed. As interventions cannot influence the probability of a component being repairable, we assume that all components can be repaired. Our model therefore is a closed-loop, where customer demand that is met is subtracted from the on hand stock and added to the repair pipeline. In the first place we consider expediting as sole intervention as it is the most-used intervention at FS. Additionally, we demonstrate the possibility for model extensions by introducing temporarily hiring components from external parties.

Operational model
On an operational level, we optimize the timing of performing interventions for a wide range of inventory levels and SKUs. We train a neural network to choose interventions and use discrete event simulation to simulate the state transitions and rewards used to train the model. The model takes an action (intervention) for a single SKU at a time, but is trained to take actions for different SKUs based on their characteristics (e.g. demand rate, acquisition cost, and repair lead time). These characteristics are assumed to be fixed.

In real life, the operational planner is able to perform interventions on a daily basis. However, the aim of this thesis is not to optimize interventions (for this we refer to (Petter, 2021)). The objective of this thesis is to determine base stock levels that are optimal on an integrated level. Therefore, we consider the effect of performing interventions on a more aggregate level. We consider periods with a length of 14 days each (i.e. time units of 14 days), at the start of which a decision is made (i.e. an intervention is performed). Within a period, no decisions are made. An overview of the operational planning model is given in Figure 4.3.

Excluding daily interventions is a limitation of our model, but it is one chosen carefully, to limit the state space and computation time. We expect this period length to provide enough moments to simulate the effect of performing interventions for the purpose of integrating the planning levels.

At the start of each period, the decision is made how many repair jobs should be expedited. Expediting repair jobs is a complex process subject to exogenous factors such as material availability. Hence, how much repair lead times can be reduced by the intervention is modeled as a random variable with a probability distribution, which is dependent on the remaining lead time of the repair job.

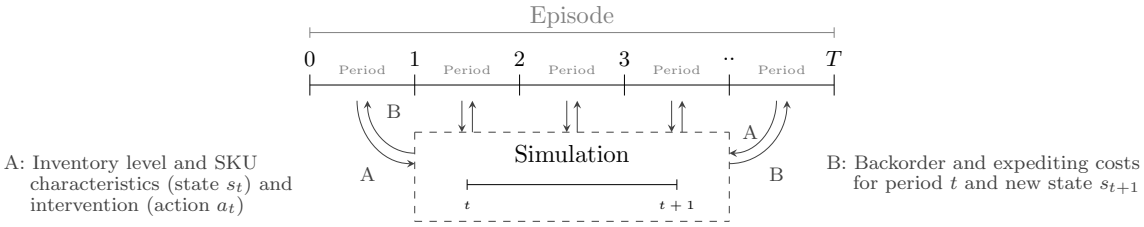


Figure 4.3: Overview operational planning model

Tactical model
After the operational model is trained, and thus the interventions are optimized, we train the tactical neural network. On a tactical level, we optimize the base stock levels such that the sum of the operational costs (i.e. backorder and intervention costs) and tactical costs (i.e. holding costs and penalty costs for not meeting service level agreements) are minimized. Operational decisions (interventions) are now fixed and given by the operational model, but the costs are dependent on the base stock levels (determined tactically). In every episode, we simulate multiple years, to also include the costs for slow-moving products that are sold less than once per year. At the start of each episode, the base stock levels are set. Subsequently, a number of periods is run to find the backorder, intervention and holding costs. The same simulation as for the operational model is used to determine what the optimal operational costs are for a set of BSLs, as shown in Figure 4.2. Note that we sequentially train the models as the operational model is trained for a wide range of BSLs and SKUs.

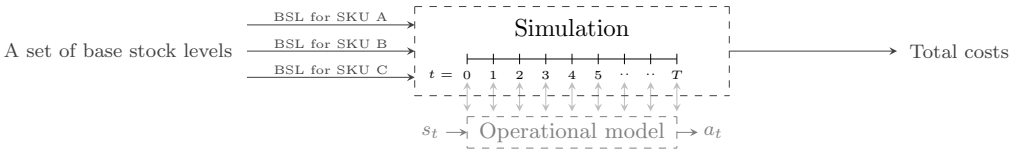


Figure 4.4: Overview tactical planning model

4.2 Operational model

The operational model will be used to simulate operational decision making: to find what interventions (action) should be taken for a given scenario (state).

4.2.1 State space

The state vector of the operational model should include all relevant information of a component's state. A list of the state features and a short explanation about each feature is provided below. We include two types of state features: inventory and stationary state features. The stationary state features include the component's acquisition cost, repair lead time and demand characteristics. These are assumed to be fixed for each component. The inventory state features give information about the current inventory position of a component. These are the stock on hand, backorders and repair pipeline. In the list below and throughout the rest of this section, index n is used to indicate that the features can differ per component. Index t is added to indicate which state features are stationary and which are non-stationary (about the inventory position at that time). The latter can change between periods in an episode. The former only change when a new component is selected at the start of an episode.

- **Stock on hand** $(IL_{n,t})^+$**:** The number of items of component type n that is directly available to ship at time t. In the notation, we use IL for inventory level, similarly to (Oroojlooyjadid, Nazari, Snyder, & Takáč, 2022). Note that the $^+$ is added as (physical) on hand inventory cannot be negative.
- **Backorders** $(IL_{n,t})^-$**:** The number of demand items of component type n at the start of period t that have been ordered by customers, but which could not be fulfilled due to a lack of stock on hand. As we follow a first-come-first-serve policy, the number of backorders cannot be positive unless the stock on hand equals zero, and vice versa.
- **Repair shop pipeline** $[z^{repair}_{n,t,l=0}, z^{repair}_{n,t,l=1}, ...]$**:** A vector of the number of items of component type n that are in the repair shop at time t and expected to arrive at the start of period $t+l$, where $l \in [0, ..., l_n^{repair}]$. When demand is met or backorders are solved, new repair jobs are added to $z^{repair}_{n,t,l_n^{repair}}$ (see Section 4.2.3). Repair jobs in $z^{repair}_{n,t,l=0}$ arrive at the start of the period.
- **Demand during period** d_n^{period}**:** The average demand during one period for component type n.
- **Acquisition costs** P_n**:** The costs for buying one item of type n.
- **Repair lead time** l_n^{repair}**:** The number of periods it regularly (without performing interventions) takes to repair a failed component of type n.
- **Expediting success probability** $Pr^{expedite}$**:** The probability of successfully expediting a repair job by one period. See Section 4.2.3 for more information.

The state vector for the operational model is given in Equation 1.1.

$$S^{operational}_{n,t} = \Big[(IL_{n,t})^+, (IL_{n,t})^-, [z^{repair}_{n,t,l}], d_n^{period}, P_n, l_n^{repair}, Pr^{expedite}\Big] \quad \text{for } l \in [0, ..., l_n^{repair}] \tag{1.1}$$

Recall from Section 4.1 that the model will be trained for one single component per episode. Therefore, for every period in an episode, the state vector $S^{operational}_{n,t}$ has a length of $v = 6 + (l_n^{repair} + 1)$ state features. The four stationary state features are fixed per episode, but will change if a new component n is selected at the start of the next episode.

4.2.2 Action space

The actions that the operational planner (OPP) can take are the interventions mentioned in 2.5.2. To limit our model's complexity, we only include expediting repairs as a possible intervention. Thus, within our operational model, we limit the action space to be the number of repair jobs of SKU n to expedite at time t, denoted by $x^{expedite}_{n,t}$. How many periods a repair job is expedited (e.g. from

$z^{repair}_{n,t,l=2}$ to $z^{repair}_{n,t,l=1}$) is dependent on several external factors for the repair shop and is modelled with a certain probability, explained in Section 4.2.3. We only include costs for the number of repair jobs that are expedited, not for the number of periods a repair job is expedited, as explained in Section 4.2.4. If the action is to expedite more than one repair job, the repair jobs in the pipeline are expedited starting from the repair job that has the smallest remaining lead time.

In Figure 4.5, the neural network of our operational model is schematically shown. Here, $[f_1, f_2, ..., f_v]$ denote the state features from Equation 1.1 and A denotes the intervention taken. Thus, $x^{expedite}_{n,t} = A$, where $A \in [0, 1, ..., max_{exp}]$. max_{exp} is the maximum number of repair jobs that can be expedited in a period.

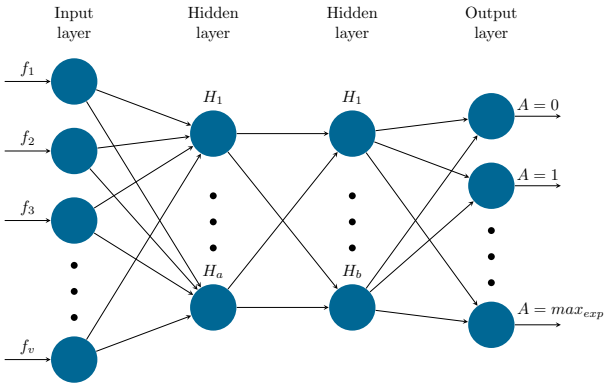


Figure 4.5: Neural network of operational model

The operational neural network thus finds one action (intervention) per component and is considered single-item. However, we train the neural network for different products by providing the relevant stationary state features as explained in Section 4.2.1.

4.2.3 Simulation

To generate rewards and find state transitions (required for DQL to train the neural network), we use Discrete Event Simulation. The simulation uses the state and action for one SKU as inputs, and outputs the reward and the next state for that single SKU. Every simulation step resembles one period (time units of 14 days), at the start of which an action is taken. In this section, we first provide an overview of the simulation and subsequently explain each step in more detail.

Simulation overview

For each period, we perform the following order of events:

1. Update repair pipeline: move every repair job one period ($l \rightarrow l - 1$).
2. Perform interventions. It is possible that a repair job with remaining lead time l is finished at the start of the current period. See Section 4.2.3 for more information.
3. Process arriving replenishments. Here, we solve potential backorders or increase the stock on hand. Penalize remaining backorders.
4. Generate and process demand. Penalize new backorders.

The transition functions for the stock on hand and backorders are given in Equation 1.2. Arriving customer orders are fulfilled following a first-come-first-serve policy.

$$\begin{aligned} IL_{n,t+1} &= (IL_{n,t})^+ - (IL_{n,t})^- - d_n + z^{repair}_{n,t,l=1} \\ (IL_{n,t+1})^+ &= \max(0, IL_{n,t+1}) \\ (IL_{n,t+1})^- &= \max(0, -IL_{n,t+1}) \end{aligned} \tag{1.2}$$

Replenishments (i.e. arriving repair pipeline) arrive at the end of a period. For example, $z^{repair}_{n,t,l=0}$ arrives at the start of period t (at the end of period $t-1$) and $z^{repair}_{n,t,l=1}$ arrives at the end of period t (at the start of period $t+1$).

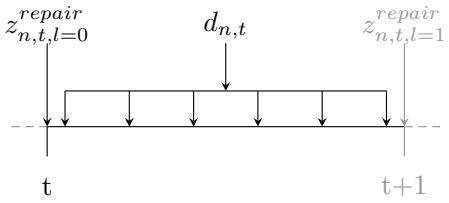


Figure 4.6: Order of events

1) Update repair pipeline At the start of the period, the repair pipeline is updated, e.g. $z^{repair}_{n,t,l=0} = z^{repair}_{n,t-1,l=1}$.

2) Process interventions Directly after updating the pipeline, the interventions are processed. Recall that the success of expediting is dependent on many external factors, which we model as probability. Repair jobs with high remaining lead times have a larger probability that the repair can be expedited. However, the probability that the repair job will be finished at the start of the current period (at $z^{repair}_{n,t,l=0}$) decreases when the lead time increases. These probabilities are schematically illustrated in Figure 4.7. To illustrate, if we expedite a repair job with a remaining lead time of two periods and $Pr^{expedite} = 0.5$, there is a $\frac{1}{4}$ probability that the repair job is available at the start of the current period, a $\frac{1}{2}$ probability that it will be available next period, and a $\frac{1}{4}$ probability that the repair job cannot be expedited.

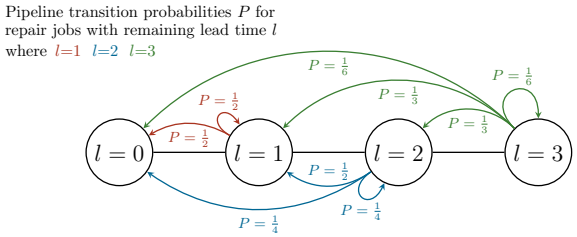


Figure 4.7: Expedite success probabilities (transitions with respect to remaining lead times)

To include this stochasticity into our model, we use the binomial probability distribution as it is a simple approach for modelling the transitions described above. For the binomial distribution, we require two variables, in literature denoted by n and p. In our case, n is the number of periods of the remaining repair lead time (thus $n = l$) and p the probability of successfully decreasing the repair lead time with one period ($p = 0.5$ in previous example). $P(X = x)$ denotes the probability that a repair job can be expedited for x periods. For every action $x^{expedite}_{n,t}$ we run this random expediting process, following the probability equation below.

$$\mathrm{P}(X = x) = \binom{l}{k} p^x (1-p)^{l-x} \tag{1.3}$$

For example, if a repair job scheduled for $l = 3$ is expedited, there is a $P(X = 2) = \binom{3}{2} p^2 (1-p)^1 = 1/3$ that the repair is expedited by two periods, to the pipeline for $l = 1$. Similarly, there is a probability of $1/3$ of expediting one period, $1/6$ for expediting three periods, and $1/6$ for expediting zero periods (intervention is unsuccessful). The random variable for how many periods a repair job is expedited (moved in the pipeline) is denoted by X. The mathematical expression for this example, with $l = 3$ is given in Equation 1.4.

$$\begin{aligned} z^{repair}_{n,t,l=3} &\leftarrow z^{repair}_{n,t,l=3} - 1 \\ z^{repair}_{n,t,l=3-X} &\leftarrow z^{repair}_{n,t,l=3-X} + 1 \end{aligned} \tag{1.4}$$

Repair jobs $z^{repair}_{n,t,l}$ where $l = 0$ cannot be expedited, as these are already available at the start of the current period. Similarly, repair jobs where $l = l^{repair}_n + 1$ cannot be expedited as these have only recently be added to the repair pipeline. This restriction is added to include the time it takes to ship a serviceable unit to the customer and receive the unserviceable unit back from the customer.

Within one period, it is not possible to try to expedite a repair job multiple times. However, it is possible that a repair job is expedited multiple times given that these interventions are performed in different periods. For example, if we try to expedite a repair job at $l = 2$ but this was not successful, we can try to expedite it next period (when $l = 1$). In these situations, we assume that e.g. material availability can change during a period.

3) Process arriving replenishments Afterwards the interventions are processed, the replenishments arriving at t (indicated by $z^{repair}_{n,t,l=0}$) are used to solve potential backorders and are added to

the stock on hand. Backorders that have been solved are added to the repair pipeline. Remaining backorders are penalized, as explained in Section 4.2.4.

4) Generate and process demand When the replenishments are processed, we generate the demand during the period. Orders (customer demand) arrive randomly. These demand occurrences are assumed to be uniformly distributed, such that backorder costs can be calculated (see Section 4.2.4). The accumulated size of the orders for component n during a period is Poisson distributed.

All demand that can be met by the stock on hand, is added to the repair pipeline, denoted by $z^{repair}_{n,t,l^{repair}_n+1}$. The products are added to the pipeline of $l^{repair}_n + 1$, instead of l^{repair}_n, to account for time between the occurrence of the demand and the end of the period. Thus, as demand occurrences are distributed uniformly over a period, repair jobs from solved demand on average stay in the pipeline for $l^{repair}_n + 1$ for half a period. If the demand is higher than the stock on hand, backorders occur. How backorders are penalized is explained in Section 4.2.4.

4.2.4 Reward function

We define two types of costs for our operational model: backorder and intervention costs.

Backorder costs
We define $costs^{backorder}_n$ to be the costs for holding one backorder for one period. Thus, backorders are penalized per unit of time, not per occurrence.

Backorders occur when the demand during a period is higher than the on-hand stock at the beginning of that period. Backorders can only be solved by replenishments (items returning from the repair shop). These arrive at the beginning of a period. As backorders at t can only be solved at the beginning of period $t + 1$, the backorders at the beginning of the period, $(IL_{n,t})^-$ are multiplied with the backorder costs for one complete period (see the first line in Equation 1.5).

Additionally, we have to include the time between the occurrence of a backorder and the beginning of the next period. As demand occurrences are distributed uniformly over a period, we assume that on average, backorders that occur during a period stay for half a period. Therefore, these are multiplied with half the backorder costs (see the second line in Equation 1.5).

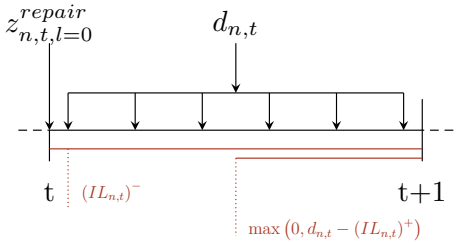


Figure 4.8: Timeline backorders

Expediting costs
Performing interventions disrupts the planned repair activities and requires time from the operational planner. For this reason, costs for performing expedited repairs are included into the reward function. These costs represent the efforts of the OPP and the repair shop employees for discussing possibilities. As the repair job has to be performed anyway, these costs are not included in the the expediting costs. Thus, the expediting costs are independent of the component type and the number of periods the repair job is expedited. Also when an intervention is unsuccessful, these costs are included as manpower has gone into investigating the possibility of performing the intervention.

Total operational cost function
The total operational cost function for component n at period t is formulated as follows:

$$\begin{aligned} Cost^{operational}_{n,t} &= (IL_{n,t})^- * costs^{backorder}_n \\ &+ \max\left(0, d_n - (IL_{n,t})^+\right) * 0.5 * costs^{backorder}_n \\ &+ x^{expedite}_{n,t} * costs^{expedite} \end{aligned} \tag{1.5}$$

Note that holding costs are only included in the tactical model (see Section 4.3).

Within (Deep) Reinforcement Learning, the model is trained to maximize a certain reward function. Therefore, we multiply our cost function with -1.

4.2.5 Model extension for operational model

To get our model working properly, we have started formulating our model considering only one type of intervention. However, a wide range of interventions are possible at FS. Therefore, we want to investigate the impact of having multiple interventions. We therefore add FS's second-most used intervention to our model: external sourcing (Section 2.5.2).

As explained in Section 2.5.2, the OPP can either buy or rent an item from an external party. For our model extension, we focus on renting components. We disregard the option to buy a component, as this would result in a permanent deviation from the base stock policy. We aim to find the base stock levels that are optimal on an integrated level, and thus from which the OPP should not have to deviate (permanently).

Description of model extension
In our model, we only allow the action of renting a component to solve backorders. A maximum of one item can be rented per period. When the action is taken, one backorder is solved. We assume that it is always possible to rent a component. For the complete repair lead time, the rented component is at the customer. When the failed component is repaired, the renting stops. No other interventions are performed for these components. The costs for this action include the renting costs for the complete repair lead time. We assume that these lead times are fixed and deterministic. With this assumption, we do not model the repair of these components, as after repair, these components will not go to the stock on hand but directly back to the customer.

State feature
No changes are made in the state vector.

Action space
We add the action $x_{n,t}^{rent}$ to our model, which can take on value 0 and 1. As mentioned in Section 2.5.2, interventions can be combined. Therefore, our action space increases to the list below.

- Do nothing: $x_{n,t}^{expedite} = 0$ and $x_{n,t}^{rent} = 0$
- Only expedite repair jobs: $x_{n,t}^{expedite} = [1, ..., max_{exp}]$ and $x_{n,t}^{rent} = 0$
- Only rent a component: $x_{n,t}^{expedite} = 0$ and $x_{n,t}^{rent} = 1$
- Perform both actions: $x_{n,t}^{rent} = 1$ and $x_{n,t}^{expedite} = [1, ..., max_{exp}]$

The size of our action space thus becomes: $1 + max_{exp} + 1 + max_{exp}$.

Cost function
The cost for renting a component includes all costs for renting the item for the complete repair lead time. These costs depend on the type of component it is. Adding these costs to the cost function, we find the total cost function of our model extension to be:

$$\begin{aligned} Cost_{n,t}^{operational} = & (IL_{n,t})^{-} * costs_n^{backorder} \\ & + \max\left(0, d_n - (IL_{n,t})^{+}\right) * 0.5 * costs_n^{backorder} \\ & + x_{n,t}^{expedite} * cost^{expedite} \\ & + x_{n,t}^{rent} * cost_n^{rent} \end{aligned} \tag{1.6}$$

4.3 Tactical/integrated model

After the neural network of the operational model is fully trained, the tactical model can be trained. Where the operational model is focused on the short-term impact of interventions for one component at a time (single-item), the tactical model aims to find the base stock levels that minimize the total costs on a system level (multi-item). The total costs include both the long-term (tactical) and short-term (operational) costs. For any set of base stock levels, we iterate over the SKUs and find the operational costs by simulating the impact of performing the trained (optimized) interventions with the stock on hand initialized to the given base stock level. The number of periods is set to represent at least one year. Thus, for each SKU, we simulate one year of consecutive operational decisions and take the sum over these costs. These costs are added to the cost function of the tactical model. Therefore, this model aims to find the BSLs that minimize the total operational and tactical costs, and thus are optimal on an integrated level. Recall that we optimize the BSLs for a single-site model.

On a tactical level, the base stock levels for a group of products are determined. SKU characteristics are generally considered to be fixed during the horizon for which the BSLs are determined. We thus do not have to train our model for different state features, as these features are not expected to (significantly) change. We still opt to build and train a neural network, as the main components have already been set up for the operational model, and a neural network is expected to perform well in finding actions. Examples of alternative solution approaches are metaheuristics that are capable of handling stochastic environments.

4.3.1 State space

The input data for the model should include product information that is relevant for determining these BSLs. As we are interested in reaching service level agreements on a system level, the base stock level of one SKU is dependent on that of other SKUs. Thus, we include state features from all SKUs in the state vector. This vector includes the stationary state features mentioned in Section 4.2.1, which are all state features that represent the components' characteristics. The state vector for the tactical model thus becomes:

$$S^{tactical} = \left[d_n^{period}, P_n, l_n^{repair}, Pr^{expedite}\right]_{n \in N}. \tag{1.7}$$

Here, N denotes a selection of Stock Keeping Units (SKUs). The length of the state vector is thus dependent on the number of SKUs in N, where the number of state features equals $4 * N$. The state features do not change while either training or testing the model.

4.3.2 Action space

The tactical model should output the BSLs for the SKUs in N that are near-optimal on an integrated level. Thus, the model will be trained to find the BSLs that minimize the total costs for a given selection of SKUs (or rather for the characteristics of a given selection of SKUs).

To limit our action space, we set a BSL-range for each product. The BSL-range is a predefined range of levels that the model should consider. For example, if the demand during lead time for an arbitrary component equals 3 units, it is unnecessary to waste time training our model on having 0, 1, or 2 items as base stock levels as these will lead to very low fill rates. Therefore, the minimum BSL we include in our model is the demand during lead time.

As we expect the optimal BSLs to lay around the base stock levels found by heuristics that do not consider interventions, we set the maximum BSL to be the BSL from a heuristic + the demand for one period. Thus, for each SKU, the BSL_n^{range} becomes $\left[\lceil d_n^{period} * l_n^{repair} \rceil, BSL_n^{heuristic} + \lceil d_n^{period} \rceil\right]$.

In our operational neural network (Figure 4.5[1]), one action for one SKU has to be chosen. From the output layer, the best action (represented by the node with the highest value) is taken. On a tactical level, we have to choose one action (i.e. a BSL) for every SKU in N.

[1]Note that this is a simplified figure. See Appendix E for more information.

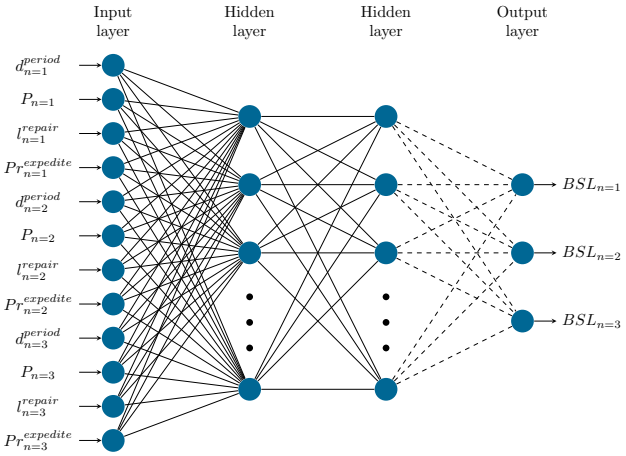


Figure 4.9: Neural network for tactical model

To conclude, the action space (all possible actions) for the tactical/integrated model is defined as

$$[BSL_1, BSL_2, ..., BSL_N]\,. \tag{1.8}$$

4.3.3 Reward function

All costs relevant for determining the base stock levels should be included in our costs function. First, we include the operational costs. The operational model was trained for a range of randomly selected BSLs for all SKUs. The operational cost function was calculated at every period in an episode. For the tactical costs function, we look at the costs over a complete episode instead of just one period within an episode. The length of the periods and the number of episodes are the same as for the operational model (2 weeks and 100 periods respectively), as the operational model is trained with these settings and we use this model to find the operational costs. We take the sum of the operational costs (Equation 1.5) for all periods in an episode and for all products in the selection of SKUs. These costs are computed by the same Discrete Event Simulation as used for the operational model. However, now the BSLs, represented by the initial stock on hand levels, are selected by the neural network instead of taken randomly (as done when training the operational model).

Next to the operational costs of every period in an episode, we include the holding costs of every period. These costs are relevant to accurately incorporate the impact of a BSL. For the holding costs, the costs per year are set to 20% of the acquisition cost of an item, as generally accepted in literature. The holding cost per period thus is equal to $P_n * 0.20/26$. Following Vanvuchelen et al. (2020) and Geevers (2020), the backorder cost per period are (approximately) equal to 19 times the holding cost.

Similar to the budget constraint, costs can be added for not reaching a predetermined service level. FS aims to have a demand-weighted-average fill rate of at least 95%. Thus, service level penalty is given when the simulated weighted-average fill rate over all products, fr_N, is lower than 0.95.

The total costs function on a tactical (or rather integrated) level is given in Equation 1.9, where T_{ep} denotes the number of periods in an episode.

$$\begin{aligned} TotCost^{tactical} = & \sum_{n \in N} \sum_{t=1}^{T_{ep}} Cost^{operational}_{n,t} \\ & + \sum_{n \in N} \sum_{t=1}^{T_{ep}} \left(s^{onhand}_{n,t}\right)^{+} * costs^{holding}_{n} \\ & + \max(0, 0.95 - fr_N) * cost^{penalty} \qquad \text{(Service level penalty)} \end{aligned} \tag{1.9}$$

Similar to the operational costs function, we multiply the costs function with -1 to find our reward function.

5 Experiments

In Chapter 4, we have formulated two models. This chapter starts with relevant information for both models. Subsequently, we describe the experimental design of the operational model, alongside with some interesting results from running experiments with this model. Last, we explain the design and results of experiments run with the tactical (integrated) model.

5.1 General experimental design

Before performing experiments, our model needs to be trained and validated. In this section, we briefly introduce our sample data, a heuristic for enabling performance comparison and the deep learning algorithm that is used. We use the Tensorflow (Abadi et al., 2015) and Keras (Gulli & Pal, 2017) packages to create, train and test our DRL models.

5.1.1 Sample SKU data

To train and test our model, a selection of 10 stock keeping units is created. The selection, shown in Table 2, represents the wide range of demand rates and acquisition costs of the SKUs that FS keeps on stock. For the first experiments, the repair lead time and expedite success probability are identical for all SKUs. In later experiments, we will consider lead times of 1 to 4 periods and expediting success probabilities of 0.3, 0.5, and 0.7.

Table 2: SKU characteristics for experimentation

SKU	**Demand per year**	**Acquisition costs**	**Repair leadtime**	**Expedite probability**
A	0.67 pcs	45.000 euros	2 periods	0.5
B	1.33 pcs	4.140 euros	2 periods	0.5
C	4.22 pcs	1.845 euros	2 periods	0.5
D	4.89 pcs	12.500 euros	2 periods	0.5
E	5.56 pcs	5.790 euros	2 periods	0.5
F	8.22 pcs	22.370 euros	2 periods	0.5
G	8.67 pcs	5.690 euros	2 periods	0.5
H	11.56 pcs	6.500 euros	2 periods	0.5
I	17.56 pcs	13.200 euros	2 periods	0.5
J	36.89 pcs	37.000 euros	2 periods	0.5

pcs = pieces

5.1.2 Framework and heuristic for validation and performance comparison

To validate our model, we compare the performance results of our simulation to expected performances. Following Gerrits, Topan, and van der Heijden (2022), we analytically determine expected fill rates for different stock levels. As the demand for all SKUs is generated via a Poisson distribution, the fill rate for stock level s is given by Equation 2.1, where mT is the average number of items in the repair pipeline (calculated by multiplying demand rate m with repair lead time T).

$$fr(s) = \sum_{n=0}^{s-1} \frac{(mT)^n e^{-mT}}{n!} \tag{2.1}$$

The expected number of backorders (EBO) for a stock level s can subsequently be derived by following Equation 2.2.

$$EBO(s) = mT \sum_{n=s}^{\infty} \frac{(mT)^n e^{-mT}}{n!} - s \sum_{n=s+1}^{\infty} \frac{(mT)^n e^{-mT}}{n!} \tag{2.2}$$

$$EBO(s) = mT(1 - fr(s-1)) - s(1 - fr(s)) \tag{2.3}$$

To compare our model on a system level, we need to have a set of base stock levels as our base setting. For this purpose, we use a greedy heuristic. The heuristic follows three simple steps. First, we initialize all stock levels to zero ($s_n = 0 \quad \forall n$). Secondly, we calculate the marginal expected backorder reduction per invested euro for each SKU, using Equation 2.4. Third, we select SKU n for which Δ_n is maximal and increment the stock level for that SKU by 1 (i.e. $s_n \leftarrow s_n + 1$). We repeat step 2 and 3 until we reach a stopping criterion. FS uses a weighted average fill rate, with the yearly demand as weights, as their service level agreement. Therefore, we use a demand-weighted average fill rate larger than 0.95% as our stopping criterion.

$$\Delta_n = \frac{EBO_n(s_n) - EBO_i(s_n + 1)}{P_n} \tag{2.4}$$

To acquire a demand-weighted average fill rate of at least 95%, we find the following BSLs: 1, 2, 3, 2, 3, 3, 4, 4, 5, and 7 for SKU A to J respectively. This corresponds to a total inventory investment of €542.055,-.

5.1.3 Algorithm

As mentioned, we have chosen to use Deep Q-learning to train Deep Q-Networks (DQN) for our DRL models. We have chosen this algorithm as it is relatively easy to implement and still quite efficient and stable due to the use of experience replay. The goal of this thesis is to find how the ability of performing interventions can influence the tactical planning (i.e. BSLs). To reach this goal, it not necessary to have the most advanced algorithm possible.

5.2 Operational model: Experimental design

As explained in Section 4.2, we first train a model that optimizes the decision-making on an operational level. Afterwards, we test the model to assess its performance. The only intervention that our model includes is the possibility to expedite repairs.

5.2.1 Hyperparameter tuning

Within the field of machine learning, parameters that influence the learning process are often referred to as hyperparameters. There are many hyperparameters that can be tuned to optimize training. For some parameters, we consult literature to find appropriate values. These are briefly described below.

- **Algorithm/optimizer** An optimizer is a method that changes the attributes of a deep learning model to reduce a loss function and therefore train the model. The Keras package offers a selection of different optimizers. One example is Standard Gradient Descent. In previous research at FS (Ventevogel, 2020), the Adam optimizer was found to perform best. Additionally, this optimizer is most commonly used in machine learning literature. Therefore, we choose the Adam optimizer.
- **Number and types of layers** We use two hidden layers in our neural network. The number of nodes per layer will be tuned. A dropout layer is added between the input layer and the first hidden layer. A dropout layer randomly sets input units to zero, with a certain rate, to prevent overfitting. The dropout rate will be tuned as well.
- **Number of episodes** The number of episodes for training is set to 2.000. This can be run in approximately an hour and is expected to be enough to properly show the convergence required to assess the quality of the hyperparameters in the tuning process.
- **Number of periods per episode** One period (i.e. step in our simulation) represents two weeks. FS updates its tactical planning every year. However, as the demand rate for SKU A is less than one (see Table 2), setting the number of periods per episode to 26 is too small, as often no demand will be generated for this SKU. To on average generate at least two demand occurrences for all SKUs, we set the number of periods per episode to 100 (simulating nearly 4 years).
- **Exploration rate** ϵ The rate of exploration versus exploitation is indicated by the value of epsilon (ϵ). At the start of the learning process, we want to fully explore ($\epsilon = 1$). While training, ϵ decays to a minimum of 0.01. For the 2.000 episodes, we take $\epsilon^{decay} = 0.995$. For more information about where epsilon is used and updated (decayed) in our training process, see the pseudo code in Appendix C.
- **Memory size** N At every period in an episode, the action and corresponding reward are saved into a memory list. This memory list has a length of 5,000.
- **Replay frequency** Random samples out of the memory list are taken to speed up the learning process by increasing data efficiency. We perform a replay after every episode.
- **Discount rate** The discount rate (γ) is the weight that future rewards are given. We use a discount rate of 0.9. For example, at time t, the reward function for $t + 6$ is given a weight of $0.9^6 = 0.531$. Thus, the costs for backorders that occur over six periods are valued approximately half of the costs of a backorder in the next period, at $t + 1$.

For some remaining parameters, we do manual hyperparameter tuning, i.e. we run experiments for different settings to find suitable values. We have run several experiments with different settings for the parameters described below. These were manually tuned in no specific order.

- **Hidden layer sizes** To determine the sizes of the hidden layers, we have run experiments with the configurations 128-64, 64-64, 64-32, 64-16, 32-32, 32-16, and 16-16. We found that with configuration 32-16, our model showed the best convergence of the reward function.
- **Learning rate** The learning rate controls the step size at each iteration while moving towards minimizing the loss function. With a high learning rate, the model will take large steps.

Outliers therefore have a large impact on the learning capability of the model. With a low learning rate, moving towards the minimum loss function might take very long. We decide to start with a learning rate of 0.001 and decay this value after every episode with rate 0.999, until a minimum learning rate of 0.0001 is reached. See pseudo code in Appendix C.

- **Dropout rate** A dropout layer aims to prevent overfitting by randomly setting outputs to zero. We have inserted a dropout layer between the input layer and the first hidden layer and have experimented with several dropout rates in the range 0 to 0.5. In our case, using a dropout rate seems to have a negative effect on the learning process. Therefore, we set our dropout rate to zero, which is the same as deleting the dropout layer altogether. That a dropout layer does not improve our learning process might be due to the limited number of state features in our model.
- **Batch size** M We use a (mini)batch size of 64, which means that every time the replay function is called, 32 random observations are taken from the memory list. We have experimented with larger batch sizes, but these did not improve the learning process while increasing computation times.
- **Update target network frequency** As explained in Section 3.3, Deep Q Learning can use two networks to increase stability: the online network and the target network. The parameters of the online network are updated every period (or in our case, every episode). As can be seen in the pseudo code for Deep Q Learning (Appendix C), after every C periods or episodes, the parameters of the online network are copied to the target network. We have set C to 100 episodes.
- **Activation function** The activation function determines the output of a node based on the given inputs. We have set the activation function for the two hidden layers to be Parametric Rectified Linear Unit (PReLU). The output layer has a linear activation function.
- **Normalization factor** We have divided all costs by a factor of 10,000 to improve the learning process. Without a normalization function, it might take a very long time for the model to approximate the real q-values.

5.2.2 Training operational model (expediting only)

With the hyperparameters determined in the previous section, we train the operational model for 2.000 episodes with 100 periods per episode. Recall that at the start of every episode, a random SKU and random BSL is chosen (stock on hand level is set equal to BSL, other inventory state features are set to zero), as the operational model should optimize interventions for a wide range of base stock levels. The total costs for every episode are saved and a moving average (over 500 episodes) is given in Figure 5.1. To increase stability while training the model, costs are clipped to a certain range to avoid excessive costs influencing the learning process. In our case, we use a clipping value of 10.000. Thus, the total cost for each period (not episode) that has a total cost higher than 10.000 is reduced to 10.000. This results in 0.3% of all rewards being clipped.

Note that the data still fluctuates due to the random selection of SKU and BSL at the start of every episode. However, we do see convergence after approximately 1,000 episodes.

5.2.3 Testing operational model

After the model is trained, we test the performance of the model. In the context of this thesis, testing means that we simulate a number of consecutive periods and for each period, we take the action learned from the neural network. We use the same discrete event simulation as used while training the neural network. To ensure an accurate depiction of the performance data, we first determine a warm-up period and a minimum number of replications. Each replication is an episode run with the same starting information, but a different random number stream.

A warm-up period is chosen to minimize the effect of starting with full stock on the performance data. The length of the warm-up period is determined visually. After the warm-up period, a predetermined number of consecutive periods is simulated. A number of replications is chosen to ensure stochasticity cannot significantly influence the performance results. This is chosen based on

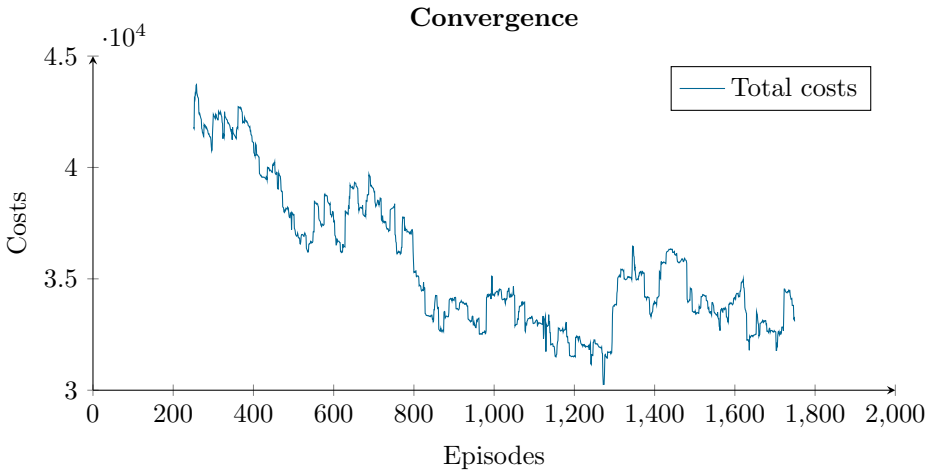


Figure 5.1: Convergence operational model

the confidence-interval half-length relative to the mean not exceeding a certain value. For more information, see Appendix D. For every replication, we run one episode of 100 periods for every SKU.

The warm-up period is set to 10 periods. The number of periods per episode is set to 100. The number of replications is set to 132 for our model with interventions (learned actions). We find that for the model without performing interventions, 339 replications are required to reach the set threshold. Thus, for validating our model, we use 339 replications. For further experiments, we use 132 replications.

Simulation validation
First, we want to validate our simulation to see if it works as expected. Recall that for all SKUs, the demand follows a Poisson distribution, which is also the distribution we use in the greedy heuristic. For this purpose, we run our model without performing interventions with the BSLs found by the greedy heuristic (1, 2, 3, 2, 3, 3, 4, 4, 5, and 7 respectively). The fill rates obtained from our simulation should approximate the analytical fill rates used in the heuristic. The analytical and simulated fill rates are presented in Table 3.

We find that the demand weighted average fill rates (from the greedy heuristic and from simulation without interventions) differ quite a bit (nearly one percent point). The average absolute difference in individual fill rates is 0.5%. We identify three reasons for the found differences in fill rates.

- Our discrete event simulation (used to find the state transitions and rewards) has an impact on the average repair lead time. When a component fails during a period, it is sent to the repair shop when a replacement is received. As demand occurrences are distributed uniformly over a period, the average repair time is the repair lead time plus half a period (of 14 days), which adds up to 5 weeks. However, if the demand during a period is larger than the stock on hand, the actual average repair lead time increases. For example, if demand during a period equals two and there is one item on stock, the first demand order can be fulfilled and the failed component will be sent to the repair pipeline. However, as we no longer have stock left, the second item will not be sent to the repair shop (failed components are only sent to the repair shop when a replacement is received; see Section 2.3). Therefore, on average, items will be in the repair pipeline a little longer than $l_n^{repair} + 0.5$ periods. We find that if we set the average repair lead time to 5.25 week, the analytical weighed average fill rate for the BSLs of the heuristic decreases to 94.80%, approximating the analytical fill rates.
- Similar to the previous point, when at the start of a period a backorder is solved by incoming replenishments, the item is sent to the repair shop. The failed component is added to the repair pipeline for one period more than the repair lead time to include the time for transporting first the replacement component from the warehouse to the operator and subsequently the failed component from the operator to the repair shop. Thus, an item with a repair lead time of 5

weeks will be in the repair shop for 6 weeks (3 periods).

- Although we have defined a minimum number of replications to limit the influence of random variables, the stochasticity of the model might still impact the fill rates, especially for slow movers (e.g. SKU A).

To summarize previous points, for SKUs with a low demand rate, the average repair lead time of our simulation will lie more around 4 weeks. For SKUs with high demand rates, the average repair lead time will lie somewhere around 5.5 weeks. This explains the higher fill rates for SKUs with lower demand rates.

The goal of this thesis is to find out how to integrate operational and tactical decision-making. In other words, we want to determine base stock levels considering operational flexibility. To this purpose, we will compare the performances of our simulation with and without performing interventions. Perfectly tuning the repair lead times of our analytically determined fill rates is not relevant in this comparison and considered beyond the scope of this thesis. Therefore, we conclude that the model's performances approximate the expected performances enough for the model to be considered valid, despite of the mentioned differences.

Table 3: Fill rates for model validation

SKU	**BSL**	**Analytical**	**Demand per year**	**Simulated without expedite**	**Simulated demand****
A	1	93.81%	0.69	96.45%	0.68
B	2	99.25%	1.33	99.25%	1.23
C	3	99.18%	4.22	99.55%	4.12
D	2	91.91%	4.89	91.56%	4.75
E	3	98.30%	5.56	98.45%	5.76
F	3	95.42%	8.22	94.46%	8.35
G	4	98.97%	8.67	98.87%	8.83
H	4	97.37%	11.56	97.29%	11.65
I	5	97.14%	17.56	96.91%	17.53
J	7	93.18%	36.89	91.19%	36.74
All*		95.62%		94.77%	

*demand-weighted average

**demand per episode multiplied by 26/100

Performance comparison

Now we want to compare the performance of our trained model to the performance without interventions. We test the model with the expediting actions that our model has learned while training. We use the same random number stream for the generated demand in both experiments (thus simulated demand is identical). The simulated fill rates with expediting should be higher than the simulated fill rates without interventions, while reducing costs. Note that we use 132 replications.

From Table 4, we find that indeed the fill rates improve together with a reduction of the total costs. The decrease in total costs is 14.8%. The increase in the weighted average fill rate amounts to 2.6%, which translates to a reduction of the sum of backorders of 46.7% . Additionally, we find a significant reduction of 48.5% in the standard deviation of the total costs over all SKUs per episode, which means that performing interventions ensures more stable operations. The reduction in backorder costs outweighs the intervention costs made.

Table 4: Fill rates for performance comparison

SKU	**BSL**	**Average demand****	**Simulated without expedite**	**Simulated with expedite**	**Number of actions**
A	1	0.68	96.45%	98.46%	3.18
B	2	1.23	99.25%	99.25%	0.36
C	3	4.15	99.55%	99.58%	0.197
D	2	4.76	91.56%	95.66%	8.364
E	3	5.75	98.45%	99.29%	4.371
F	3	8.35	94.46%	97.96%	34.78
G	4	8.83	98.87%	99.12%	1.04
H	4	11.65	97.29%	98.01%	2.62
I	5	17.53	96.91%	98.12%	4.12
J	7	36.7	91.19%	95.47%	10.72
All*			94.77%	97.22%	
Average total costs			€68,923.89	€58,735.70	
StDev total costs			66,797.05	34,413.16	
Average backorder costs			€68,923.89	€28,928.89	
Average intervention costs			-	€29,806.82	

*demand-weighted average

**demand per episode multiplied by 26/100

5.3 Operational model: Experiment results

Throughout this section, we present results found from performing experiments for the operational model with one intervention.

5.3.1 Standard data set

To learn about the expediting behaviour of our model, we want to explore when (for which states) our model takes actions. We first look at the correlation between the expediting actions and the different state features (variables in state vector). We loop over all 10 SKUs (presented in Table 2), a wide range of inventory levels and a wide range of pipeline configurations. The results are given in Table 5.

We find that the stock on hand is negatively correlated with the number of interventions, which is logical as the probability of a stock out increases when the stock on hand decreases (and vice versa). Following the same reasoning, it is not surprising that the correlation with the number of backorders is positive. We find that the intervention actions are more heavily correlated with the backorders than the stock on hand. This is an expected result, as a higher number of backorders would result in higher backorder costs, making performing interventions more attractive.

Judging from the correlations between the interventions and the repair pipeline state features, we see that only the pipeline for $l = 2$ has a positive correlation with the interventions. This is expected as the other two variables are scheduled to arrive soon, thus expediting is not (as) useful.
The values in Table 5 indicate that the pipeline for $l = 0$ and $l = 1$ are hardly correlated. This is likely due to that if there is pipeline for both $l = 0$ and $l = 2$, we still perform interventions. To confirm this, we test the model for each pipeline l while setting the other pipelines to zero. We find the new correlations for the repair pipeline to be -0.30, -.10, and 0.65, for $l = 0$, $l = 1$, and $l = 2$ respectively. Thus, our model learned to mainly expedite when there is pipeline scheduled for $l = 2$.

Finally, we see that the two stationary state features (demand rate and acquisition cost) are positively correlated with the actions. This means that with an increasing demand rate, the number of performed interventions increase as well. The same holds for the acquisition costs, albeit with a slightly larger correlation.

Table 5: Correlation state features with expediting actions

State feature	**Range per SKU**	**Correlation**
Stock on hand	[0, BSL^{max}*]	-0.36
Backorders	[0, 2]	0.43
Pipeline $l = 0$	[0, 0.5 BSL^{max}]	-0.09
Pipeline $l = 1$	[0, 0.5 BSL^{max}]	-0.05
Pipeline $l = 2$	[0, 0.5 BSL^{max}]	0.55
Demand rate	d_n^{period}	0.08
Acquisition cost	P_n	0.13

* as determined in Section 4.3.2

In the following paragraphs, we will explore the mentioned observations in terms of correlation between the actions and the state features in more detail.

Impact of inventory levels We expect that our model expedites more for lower inventory levels. We loop over the same ranges as shown in Table 5 to find the expediting behaviour of our model. The averages of the actions found by our model are depicted in a heat map in Figure 5.2. Lighter colors indicate that the model expedites more in these states. We take the average over a number of pipeline configurations to find the average number of repair jobs expedited per SKU per inventory level.

For each SKU, we see a gradual decrease in the average number of interventions (maximum step of 0.5) when the inventory level increases. Thus, we find that indeed more interventions are taken for lower inventory levels, with the largest average number of actions when there are two backorders.

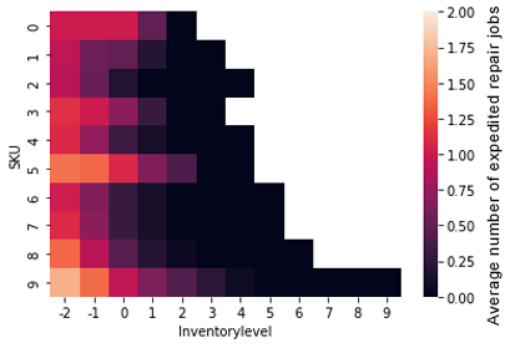


Figure 5.2: Expediting actions for all SKUs (0-9 = A-J)

Impact of pipeline configurations From the correlations in Section 5.3.1, we find that the pipeline for $l = 2$ has a strong positive impact on the number of interventions, whereas $l = 1$ and $l = 0$ have a negative effect.

We find that for all SKUs, the model mainly expedites if the sum of the pipelines for $l = 1$ and $l = 2$ is equal to or larger than 2. In Figure 5.3, the average number of actions for different sizes of pipeline $l = 2$ are plotted against the size of the pipeline for $l = 1$. To leave out the effect of the inventory level, the stock on hand is set to 1. We see that the average number of actions decreases when the size of pipeline $l = 1$ increases. This confirms the negative correlation. Additionally, the number of actions increases when the pipeline for $l = 2$ increases. Thus, if there are more items in the pipeline for $l = 2$, the model expedites more, confirming the positive correlation between the expediting and pipeline $l = 2$.

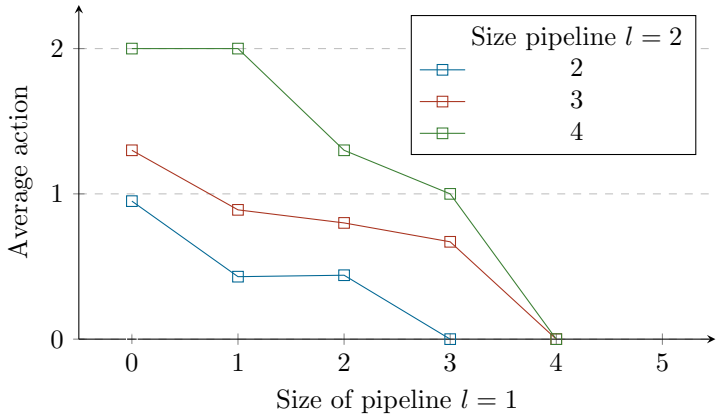


Figure 5.3: Expediting actions for different pipelines (stock on hand = 1)

Impact of stationary state features As mentioned in Section 5.3.1, the demand rate and acquisition cost are positively correlated with the number of interventions taken. Also when looking at Figure 5.2, we see that more is expedited for SKUs with higher acquisition costs. As our model is trained in a generic way, we should be able to input different combinations of values for both state features (i.e. deviating from the combinations for the SKUs in Table 2) to still find appropriate actions. We again input several inventory levels and pipeline configurations and take the average of the actions outputted by the neural network. The results are given in a heat map in Figure 5.4, where the lighter colors indicate that more expediting actions are taken.

We find that indeed the number of interventions is positively correlated with both the demand rate and the acquisition costs, depicted by a gradual increase in the average number actions for both axes.

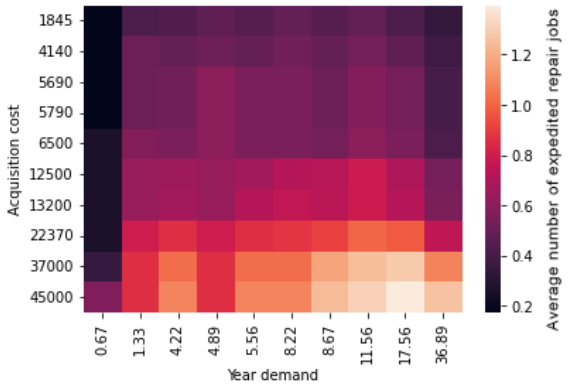


Figure 5.4: Heatmap of actions for different demand rate and acquisition cost levels

Some minor irregularities are found. For example, the average number of actions temporarily decreases around a year demand of 4.89 and 36.89. This is likely due to overfitting for the data of the selected SKUs presented in Table 2.

5.3.2 Repair lead times

In previous section, the expediting behaviour for the 10 SKUs in Table 2 is described. We found that for the stationary state features, both the yearly demand and the acquisition cost have a positive effect on the number of expediting actions. The repair lead time for all SKUs was set to 2 periods. In this section, we explore the impact of longer repair lead times.

From the ten SKUs in Table 2, we create 40 SKUs with repair lead times of respectively 1, 2, 3, and 4 periods. Before training the operational model, we run the greedy heuristic with this new data set. If we compare the heuristic base stock levels with the base stock levels found for the 10 SKUs in Table 2, we find no differences in base stock levels for the SKUs with a repair lead time of 2 periods. For the SKUs with a lead time of 1 period, we find an average BSL reduction of 0.9. For the SKUs with higher lead times (i.e. 3 and 4 periods), we find an average BSL increase of 0.6 and 1.3 respectively. Clearly, the repair lead time has a significant impact on the results of the greedy heuristic.

We train the operational model with this new data set. In Figure 5.5, we plot the average actions given by our trained operational model. Note that we take the average number of repair jobs expedited for the considered states. See Table 5 for the states we consider. We find that for every inventory level, the average number of actions is higher for SKUs with higher lead times. For example, we see that for items with a repair lead time of 2 periods and an inventory level of 2, our model expedites in $\pm$ 25% of the states.

The SKUs with a repair lead time of either 2 and 3 periods have a similar expediting behaviour. Especially for SKUs with a repair lead time of four periods, we see a large increase in the expediting behaviour of the model. For the SKUs with a lead time of only one period, the model rarely expedites.

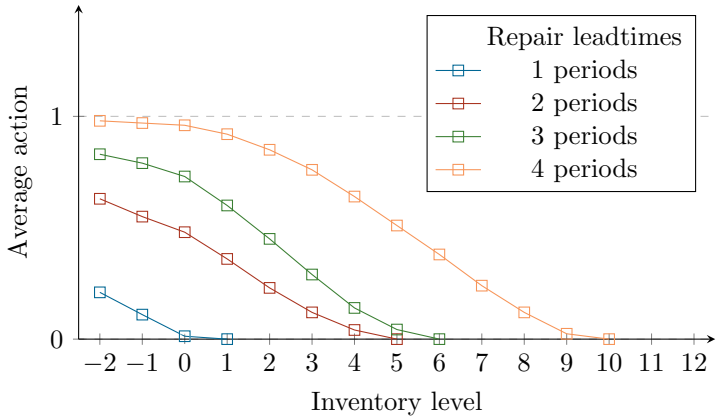


Figure 5.5: Expediting actions for different repair lead times

The described behaviour can also be seen in the performance results, when comparing the simulated model with and without expediting. On average, the fill rates increase by 0.15%, 0.89%, 1.01%, and 2.61% for SKUs with lead times 1, 2, 3, and 4 periods respectively. In general, expediting seems to enable more improvement for SKUs with a higher lead times. This follows the logic that for longer lead times, there are generally more items in the repair pipeline, and thus more jobs that can be expedited. Additionally, the repair jobs have a longer remaining lead time, which means that more lead time reduction is possible when performing interventions.

5.3.3 Expedite success probabilities

In previous sections, we have seen the impact of the demand rate, acquisition cost and repair lead time on the expediting behaviour for an SKU. Now, we are interested in seeing the impact of different expediting success probabilities on the actions learned by our model. We create a new data set (of 30 SKUs), where we consider each SKU in Table 2 with probabilities 0.3, 0.5 and 0.7. After training, we find the behaviour as described below.

We see that our model performs more interventions for SKUs with a lower success probability. For example, for an inventory level of 3 and success probability of 0.7, we see that our model expedites on average $\pm$ 0.2 times (i.e. for 1 in 5 considered states). Examples of when our model expedites are states where the demand rate is high and where pipeline $l = 0$ and $l = 1$ are zero. Thus, the model "expects" that the demand for the following two periods (before pipeline $l = 2$ arrives) is larger than 3.
For a success probability of 0.3, our model expedites in $\pm$ 4 out of 5 states. Now, our model learned to also perform interventions for lower demand rates and/or other pipelines. Consider the same scenario as mentioned above, but for a lower demand rate. For a success probability of 0.7, our model might decide not to expedite. If the demand during period t is high, our model can decide to expedite at $t + 1$ with a high probability of success. However, with a lower probability of success, it might be better to already try and expedite at period t, as it is likely not successful to intervene at period $t + 1$.

This expediting behaviour makes sense when looking at individual cases (e.g. preventing one backorder), but is controversial when considering long term impact. The expediting costs will become very high, likely outweighing the decreased backorder costs. This expediting behaviour is expected to have an impact on the integrated base stock levels, which will be addressed in Section 5.5. We expect that our model learns to hold more stock for low success probabilities, and vice versa.

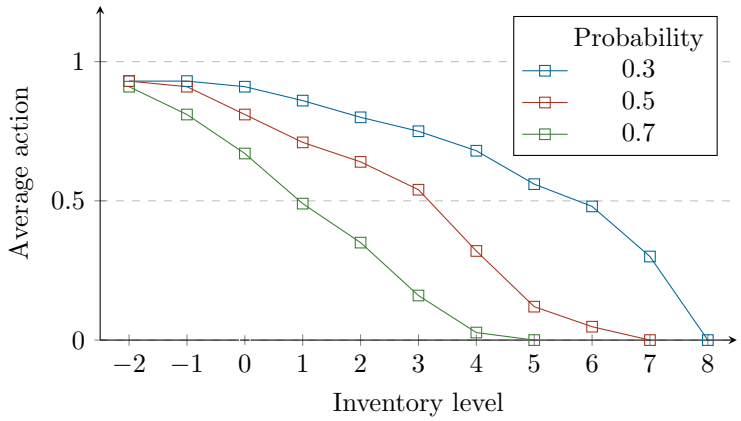


Figure 5.6: Expediting actions for different expediting success probabilities

Our model was not able to learn properly when using low, decimal input values (values between 0 and 1). Therefore, we multiplied the probabilities with a factor 10 when inputting the success probabilities as state feature into the neural network. With this slight modification, the learning process of our model improved.

5.3.4 Sensitivity analysis

Discount factor In all mentioned experiments, the discount factor has been set to 0.9. We expect that if we use a lower discount rate, our model tries to expedite more, and vice versa. This follows the reasoning that if we use a very high discount factor, many future rewards will be taken into account while only the first few future rewards can be influenced by the action taken at time t. The effect of one action will therefore be marginal and difficult to learn. When using a low discount factor, we mainly consider the current and the next period (or next few periods). Therefore, more effort will be taken to prevent a backorder.

To confirm this expectation, we experiment with discount factors 0.5, 0.75, and 0.95. We find that, indeed, the average number of actions decreases for an increasing discount factor. Performing fewer interventions results in a higher number of backorders, and therefore a reduction in fill rate. However, the total intervention costs increase drastically, outweighing the reduction in backorder costs. For a discount rate of 0.9, we find the lowest total costs, as depicted in Figure 5.7.

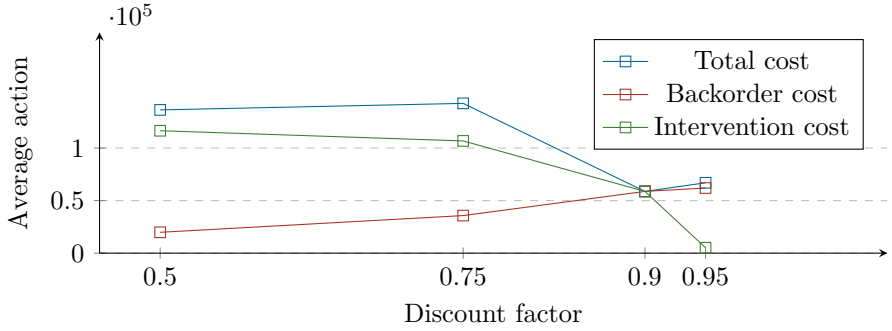


Figure 5.7: Expediting actions for different expediting success probabilities

Backorder costs calculation In our operational cost function, we assume that backorders that occur during a period should be penalized for half a period, to account for the discreteness of our simulation (i.e. time units of 14 days). This assumption is based on the situation that there is no on hand stock at the start of the period, and that demand occurrences on average happen at the middle of the period (as demand is assumed to be uniformly distributed over time). For example, if there is no stock on hand at time t and there are two demand occurrences spread uniformly over the period, the first demand arrives at $t+0.33$ and the second demand arrives at at $t+0.67$. Thus, on average, both backorders have to be penalized for $(0.33+0.67)/2 = 0.5$ period. However, if there is one item on stock, only the second demand becomes a backorder. This backorder should only be penalized for 0.33 period (time between t+0.67 and t+1).

As a sensitivity analysis, we are interested in the impact of this more accurate calculation of backorder costs. We denote the number of (temporary) backorders occurring during a period as TBO. Thus, $TBO = \max(0; d_{n,t} - (IL_{n,t})^+)$.

The time remaining after the first stockout is calculated by $1 - \frac{(IL_{n,t})^+ + 1}{(IL_{n,t})^+ + TBO + 1}$. We multiply the time after first stockout with $1 + 0.5 * (TBO + 1)$ to find the total time for which the occurred backorders should be penalized. The total backorder costs should decrease when using this calculation approach.

To test this, we run the same experiments from Table 4 with the new way of calculating the backorder costs. The results are given in Table 6. We see that indeed, the new approach of calculating costs significantly impacts the total backorder costs. However, the conclusions drawn in this chapter remain the same. The reduction in costs is significantly larger for the simulation run with expediting. This implies that even a larger cost reduction is possible than was concluded from Table 4.

Table 6: Sensitivity analysis backorder costs

Backorder calculation	**Old**	**New**	**Change**
Simulated without expediting	€68,923.89	€59,641.22	-13.49%
Simulated with expediting	€28,928.89	€22,295.21	-22.93%

5.3.5 Discussion, summary & conclusion

From the experiments result presented throughout this section, we make some observations.

Simulation validation Due to the period length of two weeks, we find that there is some difference between the simulated and expected (analytical) fill rates. For calculating the expected fill rates, we take a repair lead time of 5 weeks, as demand occurrences are spread uniformly over the period of two weeks. Thus, the average simulated repair lead time is expected to be 5 weeks. However, we find that the simulated repair lead time is dependent on the demand rate. For SKUs with a low demand rate, rarely more than one item is ordered within a period. As a consequence, due to the discreteness of the simulation, the average realized repair lead time is about 4 weeks instead of 5.
On the other hand, for items with a high demand rate, the simulated fill rate is lower than the expected fill rate. This is a result of that potential backorders are solved at the start of the period, and the items that go to the repair shop directly after a backorder is solved stay in the repair shop for 3 periods (6 weeks). On average, the realized simulated lead repair time for these SKUS is higher than 5 weeks. As a result, we find that the simulated fill rates for SKUs with a high demand rate are lower than the expected fill rates.
This is a limitation of the discrete event simulation used to find the reward and state transitions. However, the discrete event simulation does allow us to make accurate comparisons between simulations with different actions. As the latter is the main goal of this thesis, we accept the mentioned limitation.

Training model for multiple SKUs We train our operational model to take actions for different SKUs, based on their characteristics like demand rate and acquisition cost. This allows us to train a single model to learn optimal actions for multiple SKUs. Judging from the described expediting behaviour, which we deem to be logical, this approach seems to work as intended.
We did notice that when using the values 0.3, 0.5, and 0.7 for the expediting success probabilities, the model found it difficult to take learn to take different actions. When multiplying them by 10 (only when inputting the values as state feature into the neural network, not when used as probability), such that the same order of magnitude is used as most other state features, the model did learn better.

Instability Although we have found promising results, we found it to be difficult to find consistent behaviour between training sessions. The instability issues of the Deep Q-Learning algorithm, even after the implemented improvements, still result in (sometimes significant) differences in expediting behaviour. The instability is a result of that DQL learns a deterministic policy (a Q-value per action), which is known to be less robust than stochastic policies. The latter are trained to learn the probability of taking an action, which reduces overfitting (Poupart, 2020).

Expediting behaviour The model is trained to take the best action in a given state. Logically, the weighted-average fill rates improve significantly when comparing our simulation without and with learned interventions.

We find that our model logically decides to expedite more for lower inventory levels. Additionally, we find that our model expedites most for items with high acquisition costs and a high demand rate. The reasoning behind this observation is that the probability of a stock-out happening in next period is larger for SKUs with a high demand rate, and backorder costs are higher for SKUs with a high acquisition cost.

When considering a repair lead time of 2 periods, we see that our model mainly expedites when there are two or more items in repair. If the size of the pipeline that is scheduled to arrive next period increases, we see that our model performs significantly less interventions.

To further explore the expediting behaviour for different repair pipelines, we have created a data set with different repair lead times. The model learns to expedite more and sooner (i.e. for higher inventory levels) for longer repair lead times.

Lastly, we have experimented with different expediting success probabilities. We see that more interventions are performed for SKUs with lower probabilities. Additionally, the model starts expediting sooner for these SKUs.

Summary & conclusion We use discrete event simulation to generate rewards and state transitions for every period. Although we find some limitations of the discreteness of our simulation when comparing it to analytical results, we conclude that the simulation allows for reliable comparisons between different experimental settings.
We conclude that although the algorithm has some instability issues, our model performs well in making good, logical decisions. We are interested in seeing how the learned expediting behaviour might influence the base stock levels, as will be addressed in the following sections.

5.4 Integrated model: Experimental Design

After we have trained the operational neural network and see the impact of performing interventions, we are ready to train the tactical model. Here, we determine the base stock levels for a selection of SKUs (a tactical decision) while taking both operational and tactical costs into account (an integrated approach). This model can thus be considered our integrated model.

5.4.1 Model settings

For training our integrated model, we again use the Deep Q-Learning algorithm (as presented in Appendix C). Before training, we again have to determine the hyperparameters. Most hyperparameters are equal to the ones described in Section 5.2.1. Below, we only mention the changes.

- **Discount factor** As there is only one decision moment per episode and episodes are independent, no discount factor is required.
- **Hidden layers** As the number of output nodes (action) increased significantly, we test our tactical model with larger number of nodes. We found that 128 nodes for both hidden layers worked best.
- **Number of episodes** In contrast to our operational model, we now only have one decision per episode (the base stock levels), and thus only one observation per episode to add to the memory list. We perform a memory replay after every episode. In our operational model, we train our model with 100.000 memory replays (100 periods * 1000 episodes). For our tactical model, we find that 30.000 episodes (30.000 memory replays) suffices.
- **Exploration rate** With a larger number of episodes, we want the exploration rate ϵ to decay slower. We start with $\epsilon = 1$ and decay it with rate 0.999 until it reaches 0.01.
- **Memory size** As we only save one action per episode (i.e. the base stock levels), a smaller memory size suffices. We set our memory size to 1000.

Costs matrix It is computationally heavy to compute the costs corresponding to a set of BSLs by simulating 100 periods for every SKU. As the backorder, intervention and holding costs for one SKU are independent of the BSLs of other SKUs, we compute these costs separately. Before training, we simulate and store these costs into a matrix. For every SKU and every BSL, we run an episode and save the relevant costs into the mentioned matrix. Once this matrix is filled, we can start training the integrated model. During training, the operational costs are drawn from the matrix. To include stochasticity into our model, we perform a number of iterations (each iteration is one episode) for each SKU-BSL combination and store these total costs separately. During training, a random iteration is chosen for each SKU-BSL combination, such that we can find the total costs for all SKUs.

5.4.2 Training integrated model

We train the integrated model by performing the following steps for each episode. We start by selecting initial stock on hand levels (representing the base stock levels) for a selection of SKUs. For each SKU, we take a random BSL with probability ϵ and take the BSL from the neural network with probability $1 - \epsilon$. Afterwards, we calculate the total inventory investment cost (sum of acquisition costs). Subsequently, we take a random iteration from the operational cost matrix for each SKU separately. The sum of the costs found in the matrix are added to the total acquisition costs (as shown in Equation 1.9). This is the negative reward function that the model tries to maximize.

To enable comparison of the performance of the integrated model, we compare it to the base stock levels found by using the greedy heuristic. We run 132 iterations of the operational model with a slight modification to find both the operational costs (i.e. backorder and expediting costs) and the holding costs corresponding to the BSLs found by the greedy heuristic. We find a total cost of €308,999,99.

After performing 20,000 episodes, we find that our model finds a set of BSLs that outperform the heuristic BSLs in terms of both total costs and holding costs. Our model seems to find sets of BSLs

that match the performance of the heuristic BSLs after only 4,000 episodes (in less than 2 minutes). After 12,000 episodes, our model seems to have found an even better set of BSLs. The final set of base stock levels found by our integrated model is: 1, 1, 2, 3, 2, 2, 4, 3, 5, and 7 for SKU A to J respectively.

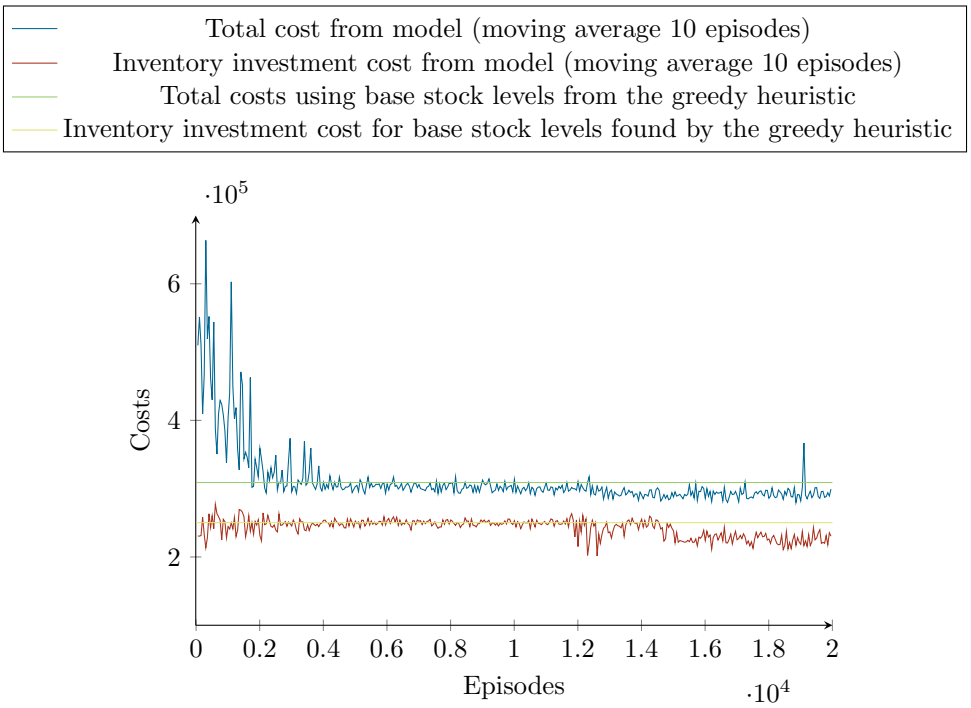


Figure 5.8: Training integrated model

5.4.3 Testing integrated model

After the integrated model is trained, and thus a set of BSLs is found, we perform a number of replications to test the performance of these BSLs. After simulating 132 replications, we find an average total cost of €305,661.96 (see Table 7). This is a reduction of 1.1% when compared to the costs found by using the BSLs from the greedy heuristic. This reduction in total costs is enabled by a decrease in holding cost of 8.4%. As expected, both the backorder and expediting costs increase, with 47.4% and 13.5% respectively. The increase in backorder costs seem significant, but is not surprising considering that the greedy heuristic ended up with a high simulated fill rate of 97.22%, resulting in very low backorders costs. The simulated weighted average fill rate of the new set of base stock levels is 95.36%. The total inventory investment costs of the new set of base stock levels is €513,910.00, a reduction of 5.2% compared to the heuristic BSLs.

The lower base stock levels have a significant, negative impact on the fill rate. The fill rate is directly taken into account in the cost function when the SLA of 95% is not reached. If the weighted average fill rate is higher than 95%, the fill rate is only indirectly taken into account in the cost function. Backorder costs are added for the time that a backorder is held. The fill rate only considers the number of backorders. Even though the total number of backorders increases with 67.01%, the total backorder costs increase with 47.36%. This indicates that backorders are held for shorter periods and for less expensive SKUs (i.e. items with lower backorder costs).

Note that for some SKUs the base stock level remained the same but the fill rates slightly changed (see SKU A, G, I, and J). This is caused by the randomness in expediting probabilities.

Table 7: Performance comparison of integrated base stock levels (standard data set)

		Heuristic		Model	
SKU	**Demand**	**BSL**	**Fillrates**	**BSL**	**Fillrates**
A	0.68	1	98.46%	1	98.14%
B	1.23	2	99.25%	1	90.65%
C	4.15	3	99.58%	2	95.89%
D	4.75	2	95.66%	3	99.58%
E	5.76	3	99.29%	2	91.74%
F	8.35	3	97.96%	2	87.40%
G	8.83	4	99.12%	4	99.09%
H	11.65	4	98.01%	3	93.85%
I	17.53	5	98.12%	5	98.16%
j	36.74	7	95.47%	7	95.45%
All*			97.223%		95.362%
Average total costs			€308,999.99		€305,661.96
Average backorder costs			€28,928.89		€42,629.90
Average intervention costs			€29,806.82		€33,829.55
Average holding costs			€250,264.28		€229,202.51
Inventory investment costs			€542,055.00		€513,910.00

*demand-weighted average

5.5 Integrated model: Experiment results

From the previous section, we find that our model outperforms the greedy heuristic for a small sample of SKUs by taking operational flexibility into account when determining base stock levels. We can thus conclude that the model's formulation is appropriate and Deep Reinforcement Learning can be used to solve it. To further test the performance of our integrated model and to inspect the behaviour of our model considering SKUs with different characteristics, we compare our model's performance to other optimization techniques and perform experiments with different, larger sets of sample data.

As mentioned in Section 4.3, the state of our integrated model does not change. Therefore, we do not have to link different states to different actions. Other optimization techniques, such as local search methods, can be used to solve the formulated model. We implement Simulated Annealing to compare our method's performance to.

Simulated Annealing Simulated Annealing is a local search method that is able to escape local optima by accepting worse solutions (Maan-Leeftink, 2021). However, finding a global optimum cannot be guaranteed. For stochastic environments such as ours, it is important that the value of a solution accurately resembles the quality of that solution. For example, if a set of base stock levels finds very low costs because of being "lucky" with low generated demand, the algorithm might falsely identify that solution as the best solution. Therefore, we take the average over multiple samples to minimize the influence of stochasticity on the value of a solution. This approach is inspired by L. Wang and Zhang (2006), where hypothesis testing is used to determine the quality of a solution. Instead of hypothesis testing, we determine a minimum number of replications to ensure the found value is close to the solution's real value. See Appendix C for more information and the pseudo code of our algorithm. Similar to our DQL approach, we use the costs matrix to calculate the total cost (according to Equation 1.9) for each replication.

5.5.1 Standard data set

As mentioned before, our model outperforms the greedy heuristic in terms of costs. Now, we want to test our model's performance with the performance of simulated annealing. Additionally, as we are dealing with a small sample of ten SKUs, we can loop over all possible combinations of base stock levels to find a global best solution. The found results are presented in Table 8.

We find that simulated annealing (SA) outperforms our DQL approach. It finds a set of base stock levels that enables a cost reduction of 3.6% (compared to our model's integrated BSLs), which is

found to be close to the optimal solution.

Note that both the SA solution and the global best solution do not completely adhere to the minimum weighted average fill rate of 95%. We expect that our model learns to never accept a solution that does not meet the SLA agreements, while accepting a small penalty might reduced total cost.

Table 8: Performance comparison of integrated base stock levels (standard data set)

SKU	**Heuristic**	**DQL**	**SA***	**Global best**
A	1	1	1	1
B	2	1	1	1
C	3	2	3	4
D	2	3	2	2
E	3	2	2	2
F	3	2	2	2
G	4	4	4	3
H	4	3	4	4
I	5	5	4	4
J	7	7	7	7
Weighted average fill rate	97.223%	95.362%	94.959%	94.753%
Average total costs	€308,999.99	€305,661.96	€294,493.41	€294,174.53
Average backorder costs	€28,928.89	€42,629.90	€44,882.02	€46,210.34
Average intervention costs	€29,806.82	€33,829.55	€32,045.45	€34,822.73
Average holding costs	€250,264.28	€229,202.51	€217,360.94	€211,905.45
Inventory investment costs	€542,055.00	€513,910.00	€498,400.00	€490,865.00
SLA penalty	-	-	€205.00	€1,235.00

*SA = Simulated Annealing

5.5.2 Repair lead times

The repair lead time is an important factor in the greedy heuristic, as shown in Section 5.3.2. Therefore, we are interested in seeing how our integrated model behaves for different lead times. We use the created data set and trained operational model from mentioned section, where we consider four repair lead times (1, 2, 3, 4 periods) for all ten SKUs of Table 2. We first run the greedy heuristic and subsequently train our model. The base stock levels from the greedy heuristic are slightly adjusted such that all SKUs have a minimum base stock level of 1.

After training our integrated model, we have found a new set of base stock levels, which should be near-optimal on an integrated level. The greedy BSLs determine that a total of 146 items have to be held on stock, which corresponds to a total inventory investment of €2,378,325. The BSLs found by our model reduce the sum of items with 6 items, which amounts to a reduction in inventory investment costs of 2.0%. These lower base stock levels enable a 1.2% total cost reduction, while still adhering to an overall weighted average fill rate larger than 95%. The reduction in holding and acquisition costs outweigh the increased backorder and intervention costs (see Table 9).

Again, simulated annealing seems to outperform our model by reducing total costs and attaining a higher weighted average fill rate.

Both solving techniques enable a cost reduction by reducing base stock levels. We find a similar behaviour: both approaches mainly reduce the base stock levels for SKUs with higher repair lead times. This behaviour is a result of that expediting repair jobs has the largest impact for longer lead times. Therefore, our model and the simulated annealing approach learn to reduce the base stock levels for SKUs with longer lead times.

Table 9: Performance comparison of integrated base stock levels (different lead times)

	Heuristic	**DQL**	**SA***
Weighted average fill rate	95.84%	95.15%	95.29%
Average total costs	€1,518,047.70	€1,500,501.34	€1,481,525.36
Average backorder costs	€225,282.89	€244,602.52	€211,092.71
Average intervention costs	€238,464.00	€237,534.00	€236,688.00
Average holding costs	€1,054,300.81	€1,018,364.82	€1,033,744.66
Inventory investment costs	€2,378,325.00	€2,331,435.00	€2,351,095.00

*Simulated Annealing

5.5.3 Expediting success probabilities

In the previous section, we have seen that our model learns to decrease the base stock levels for SKUs with longer lead times. For these items, expediting has the largest impact. We expect similar behaviour for SKUs with different expediting success probabilities. We test this by experimenting with the operational model as mentioned in Section 5.3.3.

After training both the operational and tactical model with the mentioned data set, we find the results in Table 10. The sum of items of the base stock levels found by our model is 91 items, which is 11 less than the greedy heuristic. With these new BSLs, we find a total cost reduction of 1.7%. The average base stock level over all SKUs found by the greedy heuristic is 3.4 for all three expediting success probabilities. Following our expectation, we find that the average base stock level for probabilities 0.3, 0.5 and 0.7 are 3.3, 2.9, and 2.9 respectively. Thus, for SKUs with higher success probabilities, lower base stock levels can be held, while still reducing total costs.

In contrast to previous sections, the Simulated Annealing method did outperform the greedy heuristic, but did not perform better than our DQL model.

Table 10: Performance comparison of integrated BSLs (different expediting success probabilities)

Base stock levels from	**Heuristic**	**DQL**	**SA**
Weighted average fill rate	96.47%	95.50%	95.28%
Average total costs	€1,123,393.70	€1,104,076.48	€1,117,808.75
Average backorder costs	€134,310.38	€138,960.61	€141,446.71
Average intervention costs	€237,911.36	€242,843.18	€234,264.00
Average holding costs	€751,171.96	€722,272.69	€742,098.04
Inventory investment costs	€1,626,165.00	€1,586,465.00	€1,617,990.00

5.5.4 General observations

We have performed experiments with our model using different data sets. In this section, we highlight some general observations.

First, we find that our model enables a total cost reduction between 1.1% and 1.7% for all three data sets when compared to the base stock levels of the greedy heuristic. This total cost reduction is enabled by that the decrease in holding costs outweigh the increases in backorder and intervention costs. See Table 11 for all changes in the cost terms that are included in the

Table 11: Changes in costs: comparing greedy heuristic with our model

Dataset	**Total**	**Backorder**	**Intervention**	**Holding**
Standard data set (Table 2)	-1.1%	+47.4%	+13.5%	-8.2%
Different repair lead times	-1.2%	+ 8.6%	-0.3%	-3.41%
Different expediting probabilities	-1.7%	+ 3.5%	+2.1%	-3.85%

Lower base stock levels generally result in lower total inventory investment costs: the sum of all acquisition costs multiplied by their base stock level. The total inventory investment costs are not directly included in the cost function of our model, but are indirectly taken into account through the holding costs. In Table 12, the cost reductions enabled by the integrated BSLs compared to the greedy heuristic are given. Additionally, the average acquisition cost of the units on stock held are presented.

Interestingly, we find that for all experiments, the average acquisition cost of the integrated BSLs (from both our DQL model and simulated annealing) is higher than for the heuristic BSLs. This is also the case for the global best solution found for the experiment with a small number of SKUs (average acquisition cost is €16,926.38). Apparently, our model learns that holding more stock for more expensive items reduces total costs. This observation is likely a result of (a combination of) two things: **1)** The backorder costs in our model are dependent on the acquisition costs, which makes preventing backorder for expensive items more important, resulting in higher base stock levels for these SKUs. **2)** The greedy heuristic implemented might focus too much on minimizing inventory investment costs, which results in relatively low base stock levels for expensive SKUs.

The observation of holding more expensive items on stock also has practical advantages in real life. In general, expensive parts are more crucial to a customer and thus, high availability is important. Additionally, ad hoc procurement of expensive parts is more difficult than for "cheap" items.

Table 12: Inventory investment cost comparison

		Average acquisition costs		
Dataset	**Inv. cost***	**Heuristic**	**DQL**	**SA**
Standard data set (Table 2)	-5.19%	€ 15,942.79	€ 17,130.33	€16,077.42
Different repair lead times	-3.41%	€ 16,289.90	€ 16,653.11	€17,415.20
Different expediting probabilities	-3.85%	€ 15,942.79	€ 17,433.68	€16,680.31

* change in total inventory investment cost from greedy heuristic to our DQL model

A similar observation is found for the demand rate. We see that the average demand rate for the integrated BSLs (both found by our model and by simulated annealing) is higher than for the BSLs from the greedy heuristic. Thus, when taking operational flexibility into account, we should mainly reduce the base stock levels for low demand items. This follows the reasoning that expediting costs increase significantly for high demand rates, as there are more items to be expedited.

Table 13: Demand rate comparison

Dataset	**Heuristic**	**DQL**	**SA**	**Global best**
Standard data set (Table 2)	14.54	15.59	15.02	15.60
Different repair lead times	15.03	15.40	16.39	
Different expediting probabilities	14.53	15.75	15.00	

5.5.5 Model extension: multiple interventions

A limitation of the current literature available on integration of operational and tactical spare part inventory planning is that generally, only one intervention is used. As a model extension, we have added the second most-used intervention at FS, externally renting a component, to our model.

On an operational level (when comparing simulation with and without interventions) we see an increase in total cost when performing interventions. Apparently, our model learns to take so many actions that on the long term, this has a negative impact on the total cost (also when disregarding holding cost). This might be a result of two things: **1)** On the operational level, we minimize costs considering a short horizon (through the discount factor of 0.9). Our model learns to perform interventions to minimize short term costs, but this might result in higher costs in the long run. **2)** Due to the instability issues mentioned, the learned actions from our model might not be optimal.

Although the interventions might not be optimal, we can still experiment with the model's extension on an integrated level. We see that our model reduces the total costs by 3.5%. Additionally, we find that allowing a combination of the two interventions performs even better than solely taking expediting actions, as for the same set of SKUs, our model found a total cost of €305,661.96 (see Table 8. This shows using a combination of interventions is fruitful.

Table 14: Performance comparison model extension

	Heuristic without interventions	**Heuristic with interventions**	**DQL**	**SA**
Weighted average fill rate	94.77%	96.58%	95.28%	96.15%
Average total costs	€304,995.66	€312,945.67	€302,017.59	€308,494.00
Average backorder costs	€68,923.89	€28,527.50	€22,915.00	€29,090.88
Average expediting costs	-	€3,518.18	€3,734.09	€3,677.27
Average sourcing costs	-	€38,067.89	€33,054.07386	€38,924.05
Average holding costs	€236,071.77	€242,832.11	€242,314.43	€236,801.80
Inventory investment costs	€542,055.00	€542,055.00	€542,700.00	€533,970.00

5.6 Summary, discussion & conclusion

Solving approach & instability We use both Deep Q-Learning and Simulated Annealing to solve our formulated model. Simulated Annealing outperforms DQL in two of the three experiments. This is likely a result of the instability issues of the DQL algorithm, similar to the issues mentioned in Section 5.3.5.

Model behaviour We find similar behaviour for both solving methods (Deep Q-Learning and Simulated Annealing). When taking operational flexibility into account, our integrated model learns to reduce total cost by balancing holding, backorder and intervention costs. Generally, the decrease in holding costs outweigh the increased backorder and intervention cost. Holding costs are reduced by following lower base stock levels. We find that generally, base stock levels are reduced most for SKUs with low acquisition costs, low demand rates, higher repair lead times and higher expediting success probabilities.

From the experiments run for the operational model, we find that our model performs more interventions for SKUs with high acquisition costs, high demand rates, high repair lead times and low expediting success probabilities. For the acquisition costs, demand rate and expediting success probability, we see that we lower the base stock levels for items for which we (need to) perform more interventions. This follows the reasoning that for items for which more interventions are required, it might be worthwhile to hold more stock to reduce the need for interventions.
When looking at the repair lead time experiment, however, we decrease the base stock level for items for which we perform less interventions. We expect this to be a result of the greedy heuristic that we use to determine base stock levels that do not consider interventions. The greedy heuristic already does take the repair lead time into account and determines higher base stock levels for higher repair lead times. Additionally, expediting repair jobs has a larger impact for longer lead times. This combination of larger expediting impact and the higher base stock levels as comparison result in more room for improvement for these items.

Model extension We have experimented with a model that considers two interventions: expediting repair jobs and renting components externally. Although the interventions learned by our operational model increase costs for the base stock levels from the greedy heuristic, we find that using a combination of the two interventions allows for a larger total cost reduction when compared to solely using expediting.

Fill rate versus costs We include costs for the number of periods a backorder is held, instead of including costs per backorder. A low fill rate is therefore not directly penalized. For example, two backorders that are both held for half a period is penalized equal to one backorder held one period. Total backorder costs might therefore better represent the performance of a set of base stock levels than the demand weighted average fill rates. However, as FS uses the latter as KPI, we have added them here as well.

6 Discussion & conclusion

6.1 Conclusion

The main research question of this thesis is as follows:

In what way can the spare part inventory planning on a tactical level (i.e. base-stock levels) and on an operational level (i.e. interventions) be integrated?

We found that in literature on integration of operational and tactical inventory planning, at most two interventions are considered, while FS has a wide range of interventions that are possible. Additionally, FS prefers a generalizable, data-driven approach. To this purpose, we found that Deep Reinforcement Learning is a promising solution technique.

Within Deep Reinforcement Learning, a model is trained to take actions for given states based on learned rewards. The environment of states, actions and rewards is formulated as a Markov Decision Problem. As long as the decisions and relevant inventory information can be written as an MDP, this DRL approach is possible.

We have formulated two models: one for operational and one for tactical (integrated) decision-making. The operational model is trained to perform interventions based on the inventory data of a stock keeping unit. Additionally, we add SKU characteristics (such as demand rate and acquisition costs) to the state vector. This enables our single model to take actions for different SKUs, which makes our operational model very scalable. On an tactical, or rather integrated, level, we determine the base stock levels considering both tactical and operational costs. Here, we balance backorder and intervention costs with holding costs and penalty costs for not adhering to Service Level Agreements. We expect that our model learns to hold less inventory for SKUs for which interventions have the largest impact.

We use Deep Q-Learning for training both models and experiment with different data sets. On an operational level, we find that our generic approach works well. Our model learns to perform more interventions for SKUs with higher demand rates and higher acquisition costs. Next to that, we find that higher repair lead times and lower probabilities of interventions being successful increase the number of interventions taken by our model.

With the learned interventions, we train our integrated model. We find that for all experiments, a total cost reduction of 1% to 2% is enabled when compared to our greedy heuristic. Generally, this cost reduction is enabled by that the decrease in holding costs outweighs the increased backorder and intervention costs. The decreased holding costs indirectly impact the total inventory investment costs, which are shown to be reduced by 3% to 5.5%. We find that the average demand rate and average acquisition cost of the base stock levels found by our model are higher than for the greedy heuristic. Thus, our model learns to keep less stock for cheaper and low-demand items. Additionally, we find a decrease in base stock levels for SKUs with longer repair lead times and higher expediting success probabilities, following our expectation.

Finally, we find that allowing a combination of interventions enables an even larger cost reduction of 3.5%.

As the states do not change on a tactical level, we compare our DQL model to a common local search method: Simulated Annealing. As we are dealing with a stochastic environment, we use a confidence interval half width relative to the mean as a threshold to ensure that the simulated costs of a set of base stock levels are close to the true value of that set of base stock levels. As depicted in Table 15, in two out of four experiments, the simulated annealing approach outperforms our DQL model. This is likely due to instability of the Deep Q-Learning algorithm. Robustness is a known issue for DRL algorithms with deterministic policies.

Table 15: Overview of changes in total costs and inventory investment costs

	Model (DQL)		Simulated Annealing	
Dataset	**Total**	**Inv* cost**	**Total**	**Inv* cost**
Standard data set (Table 2)	-1.1%	-5.2%	-4.7%	-8.1%
Different repair lead times	-1.2%	-3.41%	-2.4%	-1.1%
Different expediting probabilities	-1.7%	-3.85%	-0.5%	-0.5%
Model extension: renting components	-3.5%	+0.1%	-1.4%	-1,5%

*Inventory investment cost

Despite some limitations of our model and solution approach, we find that the model we have formulated is successful in finding integrated base stock levels. To answer the main research question, we conclude that Deep Reinforcement Learning is a suitable approach for integrating operational and tactical decision-making.

6.2 Contribution to literature

In terms of scientific relevance, we find three main contributions to literature.

Integration of operational and tactical decisions This thesis is the first research to propose Deep Reinforcement Learning for integrating operational and tactical decision-making. Although some stability issues have been encountered, we find that DRL is effective in determining integrated base stock levels: base stock levels that keep operational flexibility into account.

Possibility of adding interventions As described in Section 3, one downside of the current literature onthe integration of operational and tactical planning decisions is that most papers only consider one type of intervention. No paper was found that considers more than two interventions. Due to time limitations, this research mainly focuses on one intervention. However, we have shown that it is possible and relatively easy to add an intervention to our model. Our modelling approach allows interventions to be added as long as these can formulated as an MDP.

Generic approach of optimizing interventions In literature about Deep Reinforcement Learning in spare part management, most papers train their models for one component at a time (Vanvuchelen et al. (2020), Oroojlooyjadid et al. (2022), Geevers (2020)). In our operational model, we have introduced a new approach that enables a single model to learn to take different (near-optimal) actions for different SKUs. By providing SKU characteristics (i.e. demand rate, acquisition cost, repair lead time) as state features and training our model with these as well, the model learns to take the specific actions suitable for different SKUs. The main benefit of this approach is that it is scalable. A larger number of SKUs should not impact the training process of our operational model. Only a wider range of SKU characteristics will have a (slight) impact on the computation times.

6.3 Limitations

Choice of algorithm This thesis describes an exploratory research into using deep reinforcement learning for integration of operational and tactical decisions in repairable spare part management. We have decided to implement the Deep Q-learning algorithm as it is relatively easy to implement. For this research goal and the simple discrete action space that we have, a more advanced algorithm seemed to be superfluous. Additionally, we expected the algorithm with the two improvements (delayed update of target network and random batches for replay function) to be stable enough for the goal of this research.

Reflecting back on our choice of algorithm, the stochastic nature of our model (as modeled in the simulation) might be too difficult for DQL. Although the experiment results presented in this thesis are logical and conclusive, training the models was found to be difficult and often unstable (resulting in deviating results amongst different training sessions). For future research, we recommend to use an algorithm that learns stochastic policies instead of a deterministic policies, to improve robustness (Poupart, 2020).

Period time units To limit computation time and the size of our state vector (number of state features for keeping track of the pipeline), we have set one period (i.e. one simulation step) to represent two weeks. Only at the start of these two weeks, interventions can be taken. This is a limitation of our model, as in real life, interventions can be taken on a daily basis. We have chosen to accept this limitation as we expect that simulating and optimizing daily actions is not required for determining integrated base stock levels.

Another drawback of the two-week time units is that the average simulated repair lead time is difficult to determine and appears highly dependent on the demand rate. For more information, see Section 5.2.3.

6.4 Recommendations

Different formulation of repair pipeline As previously mentioned, the period length of our model is a limitation of this research. This period length was chosen to limit the state space and computation times. In our formulation of the repair pipeline (expected arrival dates as different state features), decreasing the period length to one week (or even one day) would increase the state vector to twice (or even 14 times) its current size.
For future research, we recommend to formulate the repair pipeline differently, such that the mentioned issues are minimized or prevented completely. Additionally, one might want to include stochasticity in lead times, where our model now assumes fixed lead times. A possible implementation would be formulating arriving pipeline using a probability distribution. This way, only the parameters can be stored as state features.

Stochastic policy algorithm Another limitation of our model is the instability of our DRL algorithm. For future research, we recommend to implement an algorithm that uses stochastic policies, where a neural network is used to learn the probability of taking an action instead of always taking the action with the highest value. This should improve the robustness of the model (L. Wang & Zhang, 2006).

Single model formulation We have decided to create, train and test two different models for the two planning levels, such that we could first test the operational decision-making before determining integrated base-stock levels. For future research, it might be worthwhile to consider creating a single model.

Include more interventions We have found some preliminary results from experimenting with different interventions. Although training our operational planning model for multiple interventions was found to be difficult, we still identify the improvement potential of including multiple interventions. Therefore, we recommend future research to consider formulating an MDP that includes three or more interventions.

References

Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., ... Zheng, X. (2015). *TensorFlow: Large-scale machine learning on heterogeneous systems.* Retrieved from `https://www.tensorflow.org/` (Software available from tensorflow.org)

Arup Kumar Sadhu, & Amit Konar. (2021). *Multi-Agent Coordination - A Reinforcement Learning Approach* (Vol. 9780521862).

Axsäter, S. (2003). A New Decision Rule for Lateral Transshipments in Inventory Systems. *Management Science*, *49*(9). doi: 10.1287/mnsc.49.9.1168.16568

Buşoniu, L., Babuška, R., & De Schutter, B. (2010). Multi-agent reinforcement learning: An overview. *Studies in Computational Intelligence*, *310*. doi: 10.1007/978-3-642-14435-6{_}7

Caggiano, K. E., Muckstadt, J. A., & Rappold, J. A. (2006). Integrated real-time capacity and inventory allocation for reparable service parts in a two-echelon supply system. *Manufacturing and Service Operations Management*, *8*(3). doi: 10.1287/msom.1060.0107

Candas, M. F., & Kutanoglu, E. (2007). Benefits of considering inventory in service parts logistics network design problems with time-based service constraints. *IIE Transactions (Institute of Industrial Engineers)*, *39*(2). doi: 10.1080/07408170600729218

Chua, R. C., Scudder, G. D., & Hill, A. V. (1993). Batching policies for a repair shop with limited spares and finite capacity. *European Journal of Operational Research*, *66*(1). doi: 10.1016/0377-2217(93)90212-6

Çömez, N., Stecke, K. E., & Çakanyildirim, M. (2012). In-season transshipments among competitive retailers. *Manufacturing and Service Operations Management*, *14*(2). doi: 10.1287/msom.1110.0364

da Silva, I. N., Spatti, D. H., Flauzino, R. A., Liboni, L. H. B., & dos Reis Alves, S. F. (2016). *Artificial neural networks: A practical course.* doi: 10.1007/978-3-319-43162-8

Dixit, A. K., & Sherrerd, J. J. (1990). *Optimization in economic theory.* Oxford University Press on Demand.

Du, Y., Chen, Q., Quan, X., Long, L., & Fung, R. Y. (2011). Berth allocation considering fuel consumption and vessel emissions. *Transportation Research Part E: Logistics and Transportation Review*, *47*(6). doi: 10.1016/j.tre.2011.05.011

Frazzon, E. M., Israel, E., Albrecht, A., Pereira, C. E., & Hellingrath, B. (2014). Spare parts supply chains' operational planning using technical condition information from intelligent maintenance systems. *Annual Reviews in Control*, *38*(1). doi: 10.1016/j.arcontrol.2014.03.014

Geevers, K. (2020, December). *Deep reinforcement learning in inventory management.* Retrieved from `http://essay.utwente.nl/85432/`

Gerrits, B., Topan, E., & van der Heijden, M. C. (2022). Operational planning in service control towers – heuristics and case study. *European Journal of Operational Research*. doi: 10.1016/j.ejor.2022.01.025

Gijsbrechts, J., Boute, R. N., Van Mieghem, J. A., & Zhang, D. (2019). Can Deep Reinforcement Learning Improve Inventory Management? Performance and Implementation of Dual Sourcing-Mode Problems. *SSRN Electronic Journal*. doi: 10.2139/ssrn.3302881

Glazebrook, K., Paterson, C., Rauscher, S., & Archibald, T. (2015). Benefits of hybrid lateral transshipments in multi-item inventory systems under periodic replenishment. *Production and Operations Management*, *24*(2). doi: 10.1111/poms.12233

Grahovac, J., & Chakravarty, A. (2001). Sharing and lateral transshipment of inventory in a supply chain with expensive low-demand items. *Management Science*, *47*(4). doi: 10.1287/mnsc.47.4.579.9826

Gulli, A., & Pal, S. (2017). *Deep learning with keras.* Packt Publishing Ltd.

Heerkens, J., & van Winden, A. (2012). *Geen probleem, een aanpak voor alle bedrijfskundige vragen en mysteries.* Business School Nederland. (Boekredactie)

Kazemi Zanjani, M., & Nourelfath, M. (2014). Integrated spare parts logistics and operations planning for maintenance service providers. *International Journal of Production Economics*, *158*. doi: 10.1016/j.ijpe.2014.07.012

Kukreja, A., & Schmidt, C. P. (2005). A model for lumpy demand parts in a multi-location inventory system with transshipments. *Computers and Operations Research*, *32*(8). doi: 10.1016/j.cor.2004.01.007

Kutanoglu, E., & Lohiya, D. (2008). Integrated inventory and transportation mode selection: A service parts logistics system. *Transportation Research Part E: Logistics and Transportation Review*, *44*(5). doi: 10.1016/j.tre.2007.02.001

Law, A. M. (2015). *Simulation modeling & analysis* (5th ed.). New York, NY, USA: McGraw-Hill.

Littman, M. L. (2001). Value-function reinforcement learning in Markov games. *Cognitive Systems Research*, *2*(1). doi: 10.1016/S1389-0417(01)00015-8

Luo, S. (2020). Dynamic scheduling for flexible job shop with new job insertions by deep reinforcement learning. *Applied Soft Computing Journal*, *91*. doi: 10.1016/j.asoc.2020.106208

Maan-Leeftink, G. (2021). *Lecture 11 - improvement heuristics ii.* Retrieved from `https://canvas.utwente.nl/courses/9505/pages/lecture-11-improvement-heuristics-ii?module_item_id=299762`

Mnih, V., Badia, A. P., Mirza, L., Graves, A., Harley, T., Lillicrap, T. P., ... Kavukcuoglu, K. (2016). Asynchronous methods for deep reinforcement learning. In *33rd international conference on machine learning, icml 2016* (Vol. 4).

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., ... Hassabis, D. (2015). Human-level control through deep reinforcement learning. *Nature*, *518*(7540). doi: 10.1038/nature14236

Ormon, S. W., & Cassady, C. R. (2004). Cannibalization policies for a set of parallel machines. In *Proceedings of the annual reliability and maintainability symposium.* doi: 10.1109/rams.2004.1285503

Oroojlooyjadid, A., Nazari, M. R., Snyder, L. V., & Takáč, M. (2022). A Deep Q-Network for the Beer Game: Deep Reinforcement Learning for Inventory Optimization. *Manufacturing and Service Operations Management*, *24*(1). doi: 10.1287/MSOM.2020.0939

Parreira, R., Pires, M., & Frazzon, E. M. (2021). Integrated Operational Supply Chain Planning in Industry 4.0. *International Journal of Integrated Supply Management*, *14*(1). doi: 10.1504/ijism.2021.10033722

Petter, J. (2021, January). *Adopting reinforcement learning in operational spare part management : visualizing the black box of decision-making.* Retrieved from `http://essay.utwente.nl/85598/`

Poupart, P. (2020, Jun). *Cs885 reinforcement learning module 2: June 6, 2020.* Retrieved from `https://cs.uwaterloo.ca/~ppoupart/teaching/cs885-spring20/slides/cs885-module2.pdf`

Puterman, M. L. (2014). *Markov decision processes: discrete stochastic dynamic programming.* John Wiley & Sons.

Salman, S., Cassady, C. R., Pohl, E. A., & Ormon, S. W. (2007). Evaluating the impact of cannibalization on fleet performance. *Quality and Reliability Engineering International*, *23*(4). doi: 10.1002/qre.826

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

Shapley, L. S. (1953). Stochastic games. *Proceedings of the national academy of sciences*, *39*(10), 1095–1100.

Sherbrooke, C. C. (1968). Metric: A multi-echelon technique for recoverable item control. *Operations Research*, *16*(1), 122–141. Retrieved 2022-07-05, from `http://www.jstor.org/stable/168407`

Sutton, R., & Barto, A. (2018). *Reinforcement learning, second edition: An introduction.* MIT Press. Retrieved from `https://books.google.nl/books?id=uWV0DwAAQBAJ`

Topan, E., Eruguz, A. S., Ma, W., van der Heijden, M. C., & Dekker, R. (2020). *A review of operational spare parts service logistics in service control towers* (Vol. 282) (No. 2). doi: 10.1016/j.ejor.2019.03.026

Topan, E., & van der Heijden, M. C. (2020). Operational level planning of a multi-item two-echelon spare parts inventory system with reactive and proactive interventions. *European Journal of Operational Research*, *284*(1). doi: 10.1016/j.ejor.2019.12.022

van Heeswijk, W. (2022). *Rl course - lecture 1.pdf.* Lecture slides from course: Reinforcement Learning with Applications (2021-2B). (University of Twente)

van Jaarsveld, W. (2020). Deep controlled learning of dynamic policies with an application to lost-sales inventory control. *arXiv preprint arXiv:2011.15122*.

Vanvuchelen, N., Gijsbrechts, J., & Boute, R. (2020). Use of Proximal Policy Optimization for the Joint Replenishment Problem. *Computers in Industry*, *119*. doi: 10.1016/j.compind.2020.103239

Ventevogel, P. (2020, November). *Construction of a proactive alert management model by using artificial intelligence.* Retrieved from `http://essay.utwente.nl/85228/`

Wang, E., Kurniawati, H., & Kroese, D. P. (2019). Inventory Control with Partially Observable States. In *23rd international congress on modelling and simulation - supporting evidence-based decision making: The role of modelling and simulation, modsim 2019.* doi: 10.36334/modsim.2019.b1.wang

Wang, L., & Zhang, L. (2006, 03). Stochastic optimization using simulated annealing with hypothesis test. *Applied Mathematics and Computation*, *174*, 1329-1342. doi: 10.1016/j.amc.2005.05.038

Wong, A., Bäck, T., Kononova, A. V., & Plaat, A. (2021). Multiagent deep reinforcement learning: Challenges and directions towards human-like approaches. *arXiv preprint arXiv:2106.15691*.

Wu, M. C., Hsu, Y. K., & Huang, L. C. (2011). An integrated approach to the design and operation for spare parts logistic systems. *Expert Systems with Applications*, *38*(4). doi: 10.1016/j.eswa.2010.08.088

Zipkin, P. (2008). Old and new methods for lost-sales inventory systems. *Operations Research*, *56*(5). doi: 10.1287/opre.1070.0471

A Additional literature on Reinforcement Learning

Model-free versus model-based Within RL, a model of an environment refers to anything an agent can use to predict how the environment will react to the agent's actions (Sutton & Barto, 2018). Monte Carlo is classified as a *model-free* method, as it only requires experience of interacting with the environment. As explained in the previous section, either real life of simulated data can be used to in a model-free approach. Thus, we can use a simulation model to sample transitions, but we do not store information about this model.

Model-based methods use a function to predict state transitions and rewards. Dynamic Programming is such a method, as it uses the state transitions probabilities and expected to calculate optimal actions. When transition probabilities are used in a function, this often indicates that it is a model-based algorithm.

Although these two types of methods seem distinctive, their aspects are regularly combined. This is however beyond the scope of this research. We focus on a model-free approach, as transition probabilities are not readily available and limited data is available to derive these probabilities.

On- or off-policy Methods can be either *on-policy* or *off-policy* (Sutton & Barto, 2018). An on-policy approach uses a policy to evaluate and improve itself. Off-policy approaches evaluate or improve a policy different from the policy that is used to generate the data. The *behavior policy*, which generates the data, can be unrelated to the *target policy*, which is evaluated. For example, if we take a random action a_t (for exploration) but determine a_{t+1} to be the action for which q is maximum, we are off-policy, as a different method is used to determine a_{t+1}. If

Monte Carlo prediction In many situations, complete knowledge of the environment is unrealistic. For example, because of the stochastic nature of customer demand at FS, we cannot with certainty say what the impact of a decision will be. To overcome this obstacle, we use Monte Carlo methods to predict what the reward (or cost) of an action will be. Monte Carlo only requires experience of interacting with the environment (Sutton & Barto, 2018). From experience, averages of samples are taken to approximate the real expected values. Either actual or simulated interaction can be used to create experiences: sample sequences of states, actions and rewards. In this research, we create a model to simulate rewards from state and action combinations.

In our situation, the reward of an action is dependent on actions chosen in the future. For example, to include the long term impact of a decision at time t, we do need to know what action is taken at time $t+1$. Therefore, Sutton & Barto (2018) consider episodic tasks, meaning that all experience is divided into episodes and all episodes eventually terminate. Value estimates and policies can only be updated after the completion of an episode, as only then all rewards of future steps are found. Returns ($G_t(s,a)$), the sum of all (discounted) future rewards, are sampled for each state-action pair. Often, a discounting factor (γ) is used to differentiate between the short- and long-term impact of an action, as shown in:

$$G_t \doteq R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + ... = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \tag{15.1}$$

Temporal Difference Learning Temporal difference (TD) learning is a combination of Monte Carlo methods and Dynamic Programming (Sutton & Barto, 2018). Similar to Monte Carlo (MC) methods, TD methods learn from experience. Similar to DP, estimates of TD methods are based on other estimates (also referred to as bootstrapping). For example, the value of a state is estimated by using the estimated values of other states. By repeating this process numerous times, the estimates should approximate the real values of states.

Contrary to MC, TD methods do not have to wait until the end of an episode to update the value of a state. After every time step t, $V(S_t)$ can be updated based on the perceived reward and state:

$$V(S_t) \leftarrow V(S_t) + \alpha * [R_{t+1} + \gamma V(S_{t+1}) - V(S_t)] \tag{15.2}$$

where α is an updating parameter. This TD method is called one-step TD, or TD(0). There are multiple TD(λ) methods where $\lambda \neq 0$, for which we refer to Sutton and Barto (2018).

SARSA One on-policy TD method is called SARSA, which refers to the following sequence: given state S_t and action A_t, we find reward R_{t+1} and state S_{t+1}. From here, we can follow our policy and take action A_{t+1}. The policy here can be fairly simple, such as an ϵ-greedy method, where we take a random action with probability ϵ and we take the action with the highest action-value (q_a) with probability 1-ϵ. All the values for state-action combinations are captured in a *Q-table*. Each value is updated after every visit of a state-action combination through the following formula (Sutton & Barto, 2018):

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha * [R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t)] \tag{15.3}$$

where α is the updating parameter and can take values between [0,1]. If $\alpha = 1$, we replace the previous Q-value with the new Q-value, which means that we never converge (i.e. full exploration). The lower value α takes, the less we update the Q-values and thus, the more we exploit the knowledge we have gained so far.

Multi-Agent Reinforcement Learning So far, we have discussed RL in which there is only one agent, the decision-maker. However, in the situation at FS, two decision-makers can be identified: the tactical planner and the operational planner. In literature, this is described as Multi-Agent Reinforcement Learning (MARL).

Frameworks
In a multi-agent RL, autonomous agents interact with the environment to learn how to maximize a reward (Wong, Bäck, Kononova, & Plaat, 2021). There are multiple frameworks that model systems with interacting agents. The oldest and most-famous framework is the *Stochastic Game*, or *Markov Game*, as first introduced by Shapley (1953).

A Markov Game can be formulated as a tuple $\langle I, S, A, R, T, \gamma \rangle$, where I denotes the set of N agents. The other variables are the same as explained in Section 3.2.1. In a Markov Game, agents can take actions simultaneously. The action space can therefore be formulated as $A = A_1 \times A_2 \times ... \times A_N$.

When agents take decisions sequentially, the model is modelled as an *extensive-form game* (Wong et al., 2021). An adapted version of the schematic representation of how the extensive-form game and the Markov Game relate to each other, and to an MDP, is given in A.1.

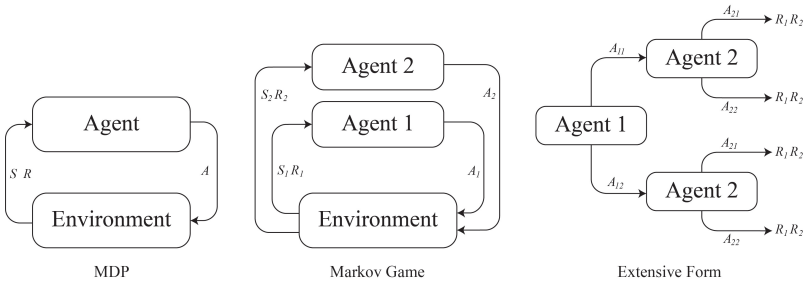


Figure A.1: Schematic representation of MDP, Markov Game and Extensive-Form Game

Benefits and challenges
Buşoniu, Babuška, and De Schutter (2010) and Wong et al. (2021) describe some benefits and challenges of MARL. One of the advantages of MARL is experience sharing. Agents that have similar tasks can learn from each other through communication. Additionally, it is possible (in some scenarios) to decrease the computation time due to parallel computation.

A major challenge of MARL is related to the curse of dimensionality. Similar to the curse of dimensionality of the single-agent RL, the number of agents increases the size of the state-action space exponentially.

Due to non-stationarity, as a consequence of simultaneous interacting with the environment, defining a goal, i.e. a value function, is an additional challenge. The expected rewards of actions are correlated and can therefore not be optimized independently. Different approaches to handling non-stationarity are found in literature.

Different agents might have different information available. This incomplete or asymmetric information across agents complicates the training of agents. Another challenge related to the learning process is the difficulty in determining the individual contribution to a joint reward. Local rewards can be used to approach this challenge, but this stimulates selfish behavior of agents.

A final relevant challenge is described by Littman (2001). An optimal policy π_* maximizes the expected sum of discounted rewards. This policy is *undominated*. For multi-agent systems, it is difficult to define a policy that is undominated as its performance depends on the behavior of other agents. One approach to defining an agent's optimal behavior is defining the optimal policy at *Nash Equilibrium*. A system is at Nash Equilibrium if there is no incentive for an agent to deviate from the current policy (Buşoniu et al., 2010). A system can have multiple Nash Equilibria.

Classifications
Agents in MARL can be divided into three classes: cooperative, competitive and mixed (Arup Kumar Sadhu & Amit Konar, 2021). In cooperative systems, all agents have the same reward function. In competitive systems, the agents have conflicting goals. A competitive agent maximizes their own reward while minimizing the opponent's rewards. In mixed systems, the reward functions are not identical, nor directly opposing.

The mixed MARL is most interesting for FS, as the operational and tactical planners do not have identical rewards functions, nor do the planners aim to minimize the other agent's reward function. There are two types of systems: static and dynamic systems. In *static* systems, or *state-less* systems, the agents do not have state transitions. *Dynamic* systems do include state transitions.

Partially Observable MDP Supply chain and logistic systems are generally subject to a lot of stochasticity (van Jaarsveld, 2020). Demand and lead times are for example randomly distributed. In a way, these problems can be considered to be partially observable. A framework for such systems is the Partially Observable MDP (E. Wang, Kurniawati, & Kroese, 2019). The POMDP can be modeled as the tuple $\langle S, A, O, T, Z, R, \gamma \rangle$. Note that we have added γ for notational consistency within this thesis. The O denotes the set of observations. An observation o is drawn from function $Z(s^{'}, a, o) = f(o|s^{'}, a)$.

In a POMDP, the exact state is never known. Only a *belief* b can be obtained, which is an estimate of the current state. The belief is updated at the end of every step based on an observation and the belief of the previous step. The policy of a POMDP is a mapping from beliefs to actions. The optimal value function for a belief is denoted as

$$V^*(b) = \max_{a \in A} \left[\sum_{s \in S} R(s,a) b(s) + \gamma \sum_{o \in O} f(o, |b, a) V^*(\tau(b, a, o)) \right] \tag{15.4}$$

where $\tau(b, a, o)$ represents the belief of the agent after performing action a and observing o.

B Additional literature on Deep Reinforcement Learning

Double DQN
Standard DQN, as described above, has one problem: the max operator in Equation 1.6 uses the same values for both action selection and evaluation (Luo, 2020). Double DQN, or DDQN, solves this by action-selection using a online network Q and estimating the values using a target network $\hat{Q}$. The training target formula of DQN is adjusted to

$$y_t^{DDQN} = r_t + \gamma \hat{Q}\left(s_{t+1}, \arg\max_{a'} Q(s_{t+1}, a'; \theta); \theta^-\right) \tag{15.1}$$

Both neural networks are trained to have its output values approximate the target (or actual) values. This makes Double DQN only a value-function-based algorithm.

Actor-Critic
Actor-Critic (AC) methods are a combination of value-function-based and policy-based methods. It learns both value functions and policies, where the actor refers to the policy and the critic to the value function (Sutton & Barto, 2018). The critic evaluates the actor's policy through a TD algorithm (see Section A) and sends a TD error, δ, to the actor. Positive values of δ indicate that an action (from the actor's policy) resulted in a state that returned a value better than expected. See Figure B.1 for a schematic overview of the interactions between the actor, the critic and the environment.

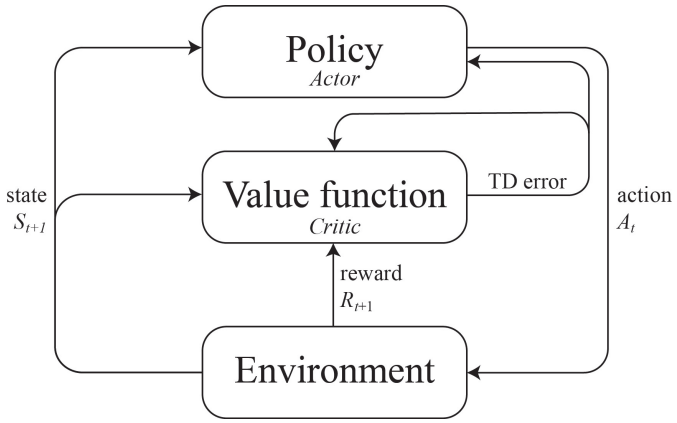


Figure B.1: Actor-Critic architecture

A special Actor-Critic method, called *Asynchronous Advantage Actor-Critic* (*A3C*) was proposed by Mnih et al. (2016). Instead of using experience replay to increase stability, as used in DQN, the A3C algorithm uses multiple parallel actors with different exploration policies. Herewith, the parameter updates are less correlated than when the updating is performed by a single agent.

Proximal Policy Optimization (PPO)
Proximal Policy Optimization (PPO), introduced by Schulman, Wolski, Dhariwal, Radford, and Klimov (2017), uses the AC architecture and combines policy gradients with trust-region methods. Similar to trust-region methods, PPO clips the distance between the old and the new policy (Vanvuchelen et al., 2020). This ensures that updates are not too large, which improves stability. Like policy gradient methods, PPO samples decision sequences.

PPO is known for its simplicity, good-sample complexity and computational efficiency. Schulman et al. (2017) applied PPO to a number of the Atari 2600 games and outperformed DQN in several cases. As PPO uses the Actor-Critic architecture, this algorithm uses both value-function-based and policy-based features.

Deep Controlled Learning (DCL)
As mentioned in Section A, high stochasticity (and thus high variance) are common in supply chain

and logistic systems. For example, a stockout could be expected but does not occur. This state-action combination (including the purchase of a new component to prevent the stockout) will thus receive a very high cost, considering both purchase and holding costs. This action will be remembered as unfavourable, and an RL model might not visit it again to change the expected value of this action.

Deep Controlled Learning (DCL) was introduced by van Jaarsveld (2020) to deal with variable trajectories. The approach improves the ability to distinguish between good and mediocre actions, which yields high-quality labelled samples. Similar to PPO and A3C, neural networks are used to represent policies. In constrant to PPO and A3C, the DCL algorithm does not use value approximators, making this a policy-based algorith.

The tuning of the neural network parameters is a relatively simple task for DCL. DCL was developed for problems in the operations management field and the method was found to outperform A3C on the lost sales inventory problem by Zipkin (2008). The performance improvement in comparison to A3C might be due to the use of supervised learning with stable targets or to the fact that no value approximators are used.

B.0.1 DRL models in inventory management literature

Many different approaches to using DRL for solving inventory control problems can be found in literature. Some relevant approaches found in literature are highlighted in the following paragraphs.

A typical reoccurring inventory control problem is the beer game. It is a multi-echelon serial supply chain including a retailer, a warehouse, a distributor, and a manufacturer. The four agents (echelons) place replenishment orders to minimize costs, without having complete information about the system as a whole. Different approaches have been implemented in the beer game, like PPO (Geevers, 2020) and DQN (Oroojlooyjadid et al., 2022).

Geevers (2020) applied PPO to the beer game, where the neural network included two hidden layers of 64 nodes. The output layer represents the recommended action to be taken, in this case, the order quantity for each agent. The input layer consists of some state features. Two state features are agent-independent: the total inventory ti and the total backorders tb. Additionally, for every agent $i \in [1, 2, 3, 4]$, the inventory h_i, the backorders C_i and the demand of the previous period d_i is captured. The final state features are both agent- and time-dependent and are related to the number of items ordered. The lead time of a replenishment order can be larger than one time period. Therefore, the number of items that were in transit in the past few periods is relevant to the state of the system. Thus, the final state features are the items in transit $tr_{i,t}$ and the items in transit that arrive at t $o_{i,t}$, where t has a horizon of four periods as the maximum lead time equals four. Both the input and output nodes are normalized to the range $[-1, 1]$. The DRL model outperformed the benchmark in the beer game. The model was also implemented in two other supply chains, where DRL outperformed in one case, but was not consistent in the other.

Oroojlooyjadid et al. (2022) applied DQN to the beer game. The action space is similar to the action space in the previous paragraph, where the output layer of the neural network denotes the order quantity. The authors included information on the past m timesteps into the set of state features: the positive and negative inventory level $\left(IL_j^i\right)^+$ and $\left(IL_j^i\right)^-$, the items on order OO_j^i, the arriving order (i.e. demand) AO_j^i and the arriving shipment AS_j^i for each agent i and for $j \in [t - m + 1, ..., t]$. The shaped-reward DQN (SRDQN) agent was found to perform well on real-world data sets, regardless of the behaviour of the other agents. When the other agents follow a base-stock policy, the SRDQN agent achieves nearly-similar costs as following a base-stock policy. If the other agents behave irrationally, the SRDQN agent outperforms following a base-stock policy.

Vanvuchelen et al. (2020) consider a system with one agent coordinating the replenishments of n shippers. The action vector is the order quantity per agent, which should sum up to a multiple of a full truckload. The state vector contains the inventory position per shipper. A tanh activation function is used to normalize the output of nodes to the range $[-1, 1]$. The PPO algorithm was found to outperform two often-used heuristics in a small-scale experiment. In larger experiments, the algorithm performs similar to the heuristic when shippers have similar cost structures, but outperforms the heuristics when the cost structures differ.

Gijsbrechts, Boute, Van Mieghem, and Zhang (2019) consider a single-item, periodic review, lost sales inventory replenishment system and applied A3C to find a near-optimal solution. Again, the action equals the order quantity for the item. The state vector has a length equal to the lead time, and is denoted as $S_t = (s_0 = I_{t-1} + q_{t-l}, s_1 = q_{t-l+1}, s_2 = q_{t-l+2}, ..., s_{l-1} = q_{t-1})$. Orders placed at $t - l$ arrive the end of period $t - 1$, such that they are available at t. After some experiments, the authors conclude that DRL can effectively solve large inventory problems. However, it is noted that the A3C algorithm did not outperform all heuristics and remains a black-box, which nuances the authors' conclusion.

C Pseudo codes

C.1 Deep Q Learning

An adapted version of (Mnih et al., 2015)

Algorithm 1 Deep Q-learning with experience replay

Initialize replay memory D to capacity N
Initialize action-value function Q with random weights θ
Initialize target action-value function $\hat{Q}$ with weights $\theta^- = \theta$
for episode $= 1, E$ **do**
  Initialize sequence $s_1 = x_1$
  for $t = 1, T$ **do**
    With probability ϵ select a random action e
    otherwise select $a_t = \arg\max_a Q(s_t, a; \theta)$
    Execute action a_t in simulation and observe reward r_t and state x_{t+1}
    Set $s_{t+1} = s_t, a_t, x_{t+1}$
    Store transition (s_t, a_t, r_t, s_{t+1}) in D
    Sample random minibatch of size M of transitions (s_j, a_t, r_t, s_{j+1}) from D
    if episode terminates at step $j + 1$ **then**
      $y_j = r_j$
    else
      $y_j = r_j + \gamma \max_{a'} \hat{Q}(s_{j+1}, a'; \theta^-)$
    end if
    Perform a gradient descent step on $(y_j - Q(s_j, a_j; \theta))^2$ with respect to the network parameters θ
  end for
  After every C episodes reset $\hat{Q} = Q$
  Decay epsilon ϵ and learning rate
end for

Note that the experience replay is incorporated by sampling a minibatch of transitions from the memory list. The purpose of the gradient descent step is similar to the updating rule for Q-learning (see Equation 1.4), but instead of using a table with individual values to update, the parameters of the neural network are updated. Similar to Q-learning, the learning rate is used in this updating process.

C.2 Simulated Annealing

Simulated Annealing is a local search method that is able to escape local optima by randomly accepting worse solutions (Maan-Leeftink, 2021). This heuristic is however not very suitable for stochastic environment, as it could store an outlier in cost as best solution even if the average cost for that solution is bad. As inspired by L. Wang and Zhang (2006) (where they use hypothesis testing to identify solution quality), we overcome this obstacle by making sure that when we generate a neighbour solution, the value of that neighbour solution is close to the true cost value of that solution. We use the same approach for determining the number of replications, as described in Appendix D.1. See Algorithm 2 for the simulated annealing pseudo code and Algorithm 3 for the pseudo code for finding the value of a neighbour solution.

Algorithm 2 Simulated annealing under stochastic environment

Determine starting temperature T^{start}, stopping temperature T^{stop}, decrease factor α and Markov Chain Length M
Initialize current temperature T: $T = T^{start}$
Initialize current solution $Sol^{current}$
Initialize best solution Sol^{best}: $Sol^{best} = Sol^{current}$
while $T > T^{stop}$ **do**
  for m = 1 to M **do**
    Find neighbour solution $Sol^{neighbour}$
    if $Sol^{neighbour} < Sol^{current}$ **then**
      $Sol^{current} = Sol^{neighbour}$
      if $Sol^{neighbour} < Sol^{best}$ **then**
        $Sol^{best} = Sol^{current}$
      end if
    else
      Take a random number X in [0,1]
      if $X < e^{(Sol^{current} - Sol^{neighbour})/T}$ **then**
        $Sol^{current} = Sol^{neighbour}$
      end if
    end if
  end for
  $T = \alpha * T$
end while
Return Sol^{best}

Algorithm 3 Simulated annealing; find neighbour solution

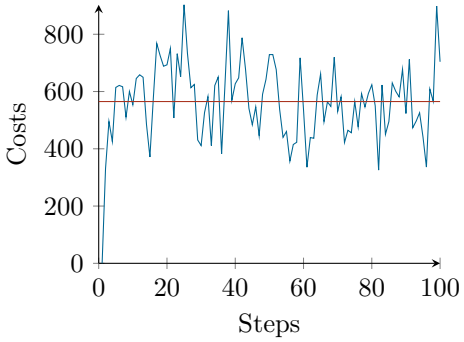
Initialize/empty reward list R
while loop = True **do**
  Find total cost for all SKUs
  Add the sum of costs over all SKUs to R
  Set n to length of R, S^2 to the variance of R and $\bar{X}$ to the mean of R
  Determine $\delta(n, 0.05) = t_{n-1,1-0.05/2} * \sqrt{\frac{S^2(n)}{n}}$
  if $\delta(n, 0.05)/\bar{X} < 0.05$ **then**
    loop = False
  end if
end while
Return $Sol^{neighbour}$

D Experimental design operational model

D.1 Warmup period and number of replications

Warm-up period At the start of every episode, the model resets. This means that the stock on hand is set equal to the base stock level, and the backorders and pipeline are set to zero. To minimize the influence of this starting position on the performance data, we first determine a warm-up period. During the warm-up period, no data to measure performance is gathered.

To determine the warm-up period, 250 episodes have been run with a length of 100 periods per episode and interventions (only expediting) enabled. Taking the average of each period over all episodes, we find the results as shown in Figure D.1a. Judging from the averages, there seems to be a short warm-up period, shown as a steep incline of costs in the first few periods. The average cost per period over the 100 periods is approximately 565. In less than 10 periods, the average cost per period reaches the average. Therefore, we set the warm-up period to 10 periods. Thus, for the first 10 periods, no performance data is registered. For each episode, we run 10 + 100 = 110 periods.



(a) Average of period costs

Figure D.1: Warm-up period

Number of replications To ensure that the randomness of demand does not influence the performance data too much, we do not just test our model for only a single episode. Instead, we perform a number of episodes and take the average of these. To determine the minimum number of replications required, we follow Law (2015). To approximate the true mean of a KPI (in this case of the total cost per episode), we need a number of samples (replications). We want to have a number of replications such that the confidence-interval half-length relative to the mean is smaller than some predetermine error γ. The confidence-interval half-length is denoted by $\delta(n, \alpha)$ and computed by

$$\delta(n, \alpha) = t_{n-1,1-\alpha/2} * \sqrt{\frac{S^2(n)}{n}} \tag{15.1}$$

, where n is the number of replications, α is the confidence (or significance) level, and S^2 the sample variance. $t_{n-1,1-\alpha/2}$ is a value from the T-student distribution corresponding to $n-1$ degrees of freedom and a probability of $1-\alpha/2$.

Law (2015) recommends using a relative error smaller than or equal to 0.15. We set the relative error to be 0.10 and set $\alpha = 0.05$. We find that after 132 replications, the relative error is smaller than 0.10, as shown in Figure D.2.

Note that we have determined the number of replications for the model with interventions. The number of replications for the model without interventions is much higher, 339. This is a result of that performing interventions decreases both the average and the variance of period rewards. For the experiments, we will mostly use our model with interventions, and thus 132 replications.

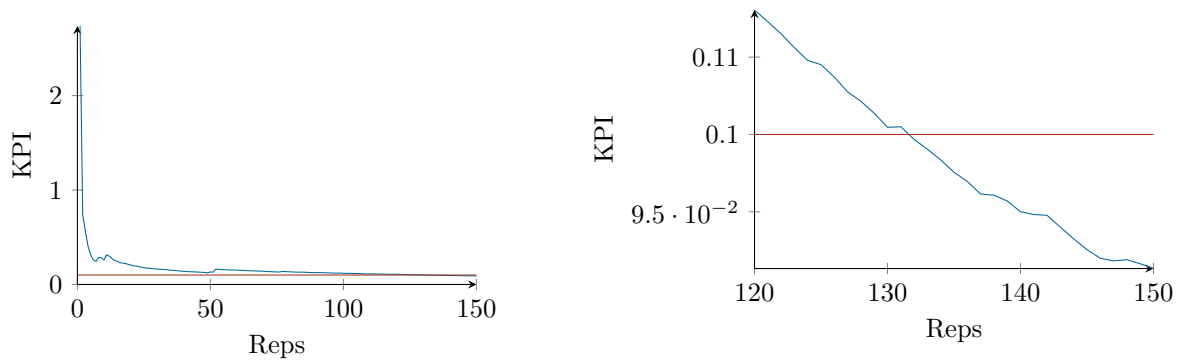


Figure D.2: Warm-up period

E Neural Network of tactical model

A Deep Q-Network requires a discrete action space. Every output node of the neural network represents one action. The value of these output nodes are trained to approximate the q-value of taking the corresponding action in a given state. For our tactical model, we want to find a base stock level for each SKU. An output node can however not store both the q-value and the inventory level for one SKU at the same time.

Therefore, we define different base stock levels as different actions. Thus, each SKU gets multiple output nodes, each representing a different base stock level for that SKU. For example, if we have 5 SKUs and we give each of them a range of 5 BSLs (e.g. 1 to 5), our tactical model has 25 outputs. Considering the random example values in Figure E.1, we would find the following base stock levels: 3, 2, 3, 5, and 3 for SKU A to E respectively.

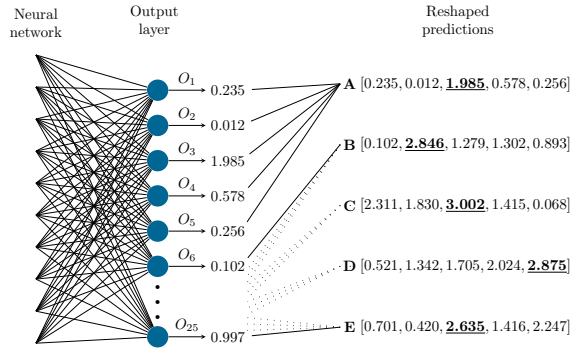


Figure E.1: Example output of tactical neural network

For future research, we recommend using a different algorithm that does allow for the output nodes to represent inventory levels. This allows for a significant reduction in the number of output nodes the model needs to train.