Industrial Engineering & Management

Master's Thesis

Adopting Reinforcement Learning in Operational Spare Part Management

Visualizing the Black Box of Decision-Making

Author:
Joris Petter

January 22, 2021

A thesis submitted in fulfillment of the requirements for the degree of Master of Science in the field of:
Production & Logistic Management
Supply Chain & Transport Management

Department of Industrial Engineering & Business Information Systems,
University Of Twente

Adopting Reinforcement Learning in Operational Spare Part Management

VISUALIZING THE BLACK BOX OF DECISION-MAKING

Author
Joris Petter
s1570692
January 22, 2021

Educational Institution
University Of Twente
Drienerlolaan 5
7522 NB Enschede
The Netherlands

Host Organization
Public version
All confidential information is removed from this thesis.

Supervisors
Internal Supervisors
dr. E. Topan
dr. ir. M.R.K. Mes

External Supervisor
ir. K. Alizadeh
Independent Aerospace Company

Faculty of Behavioural, Management and Social Sciences (BMS)
Dep. Industrial Engineering and Business Information Systems (IEBIS)

UNIVERSITY OF TWENTE.

Acknowledgments

This Master's thesis represents the last phase of my time as a student at the University of Twente. My time as a student in Enschede was not always without a struggle. Therefore, finishing my studies in these strange times fits the list nicely. Nevertheless, because of my student house, year club, (study-) friends, and family, I had a fantastic time as a student here. So, by submitting this thesis, a new phase begins.

First of all, I would like to thank Engin Topan and Martijn Mes from the University of Twente for their supervision and guidance during my thesis. With their feedback, input, and very interesting and challenging discussions, I was able to bring this research to the level as is.

Secondly, I would like to thank Kaveh Alizadeh as my external supervisor. Unfortunately, there was not a possibility to discuss the matter often in real-life. Nevertheless, despite all difficulties within the aerospace branch, the weekly online meetings helped me a lot. Because of the support, I stayed motivated until the end.

Last but certainly not least, I want to thank my friends, girlfriend, and family who helped and supported me during my thesis, both in real life and virtually.

I hope you enjoy reading this thesis!

Joris Petter
Utrecht
January 2021

Management Summary

As an independent aerospace service provider, Independent Aerospace Company (IAC) provides service programs for contracted customers. With a spare part pool of approximately 7.500 components, the IAC aims to keep all aircraft in the air. The Component Maintenance & Availability (CMA) program guarantees unlimited access to high-quality components such that downtime can be avoided at competitive costs. The CMA-program consists of a closed-loop inventory network, where components can move due to three events; demand (from the IAC to operator), unserviceable returns (from operator to repair shop), and returning repairs (from repair shop to the IAC).

To monitor real-time performance of the CMA-program, the IAC uses a Service Control Tower (SCT). The SCT gives an alert in the case a stock-out is likely to occur, which are tend to be solved by the Operational Planning Professionals (OPP) while taking two KPIs into account: costs and SLAs. However, currently, the IAC does not use an operational decision support system; the decisions are made based on the gut feeling and preference of the OPP. Therefore, an inadequate assessment of costs in the decision-making process is used for the decisions. Furthermore, in the current performance, we see that approximately 35% of all orders are delivered late to the customer. The currently investigated decision support systems by the IAC cannot incorporate the stochasticity and exceptions of scenarios sufficiently. The goal is to combine the stochasticity of scenarios and the KPIs to find the optimal decision-making. Therefore, the IAC is interested in the potential of artificial intelligence since it can add human reasoning to algorithms.

Furthermore, considering that strong conjunction between operational and tactical planning can lead to higher service levels and a better insight into costs, the IAC is interested in visualizing its *black box* of decision-making. This imaginary box contains all relationships between operational- and tactical planning decisions. The company's desire lies in obtaining more information, insight into the impact of operational decisions on long-term yield, and a visual and measurable decision-making process, including all possible stochasticity during the decision-making process. Therefore, a planning and learning algorithm, such as reinforcement learning, is preferred for this research. Based on this knowledge, the following objective is set for this research:

'In what way and to what extent can a reinforcement learning algorithm improve operational decision-making while incorporating long-term yields?'

To find the best solution approach for the IAC's purpose, an extensive literature study is performed to define the potential of artificial intelligence in Decision Support Systems (DSS). This study concludes that automated decision-making approaches are compelling for optimizing yield and creating operational control. Although SCTs and Reinforcement Learning (RL) are not studied jointly in the domain of service logistics, implementation of smart learning algorithms in digital control towers can support the transition towards a more effective and efficient logistics network. We see that the algorithms can make decent trade-offs between the improvement of

long-term decision-making and reducing short-term maintenance costs. Nevertheless, literature states that a decision support system should not entirely replace professional judgment.

For the sake of simplicity, we modeled the problem instance by a relatively simplistic approach. Nonetheless, this approach is still representative of the the IAC's situation. We used a Stochastic Dynamic Programming (SDP) framework in our modeling approach since this incorporates time-dependent decisions in a finite horizon. Here, we define the inventory position at a given time as the state. Within each time period, we make a decision: *do nothing*, *expedite*, or *buy*. Combinations of interventions are also possible. Based on the decision, we move to the next stage and receive a direct reward. Stochasticity in demand, returning repairs, and returning components from customers are considered within the updating procedure.

In solving the SDP model, we use three different solution approaches. The first solution approach of the SDP is an exact approach by backward recursion. The solutions found by the exact approach are used as a guideline and reference for the performance of the other two solution approaches. This approach is only applicable to relatively small problem instances, such as our current proposed problem. Second, we use a short-sighted (greedy) heuristic. The heuristic is greedy because it always chooses the intervention with the highest direct reward while ignoring the cost-to-go function. The final solution approach is the Dyna-Q algorithm, which includes both model-based as model-free aspects. Here, easy implementation, strong, and quick performance show added value for our research. Both the swiftness of implementation and the strength and speed of performance show added value for our research.

To determine the performance of the algorithms, we first learn the policies of the backward recursion and Dyna-Q. Then, we will evaluate all models in a simulation with pseudorandom numbers. For a sensitivity analysis, we perform the experiments with a preference for tactical level, operational level, and the equilibrium point between operational and tactical level. The first mentioned scenario is also the IAC's current way of working. Combining these three experiments creates insight into the relationship between operational and tactical levels.

For the initial scenario, we use a tactical preference in decision making. We see that the greedy and Dyna-Q approaches perform on aggregate comparable. It goes without saying that the exact solution by backward recursion gives the best value for the expected costs. In situations that seem simple, the Dyna-Q algorithm uses a conservative approach by using more expensive actions but having fewer backorders. However, the difference between greedy and Dyna-Q is in this scenario not significant.

Second, we test the operational preference in decision making. Again, we see that the backward recursion presents the optimal solution for this case. Further, we see that the greedy heuristic applies the same behavior as in the previous experiment; it does nothing as long as possible. Ignoring the cost-to-go function causes this behavior. The Dyna-Q algorithm shows in this experiment a significant improvement compared to the greedy heuristic.

Finally, we investigate the performance at an equilibrium point. To shift the impact of decisions from tactical to operational, we changed the turnover rate's value. Therefore, we divided this difference into ten steps in order to find the best suiting point. Contrary to the other experiments, we see that the greedy algorithm performs poorly in this scenario, which means that the greedy algorithm finds it hard to decide between tactical and operational decisions. Nonetheless, we see in this scenario that Dyna-Q is performing comparable as the exact solution in most situations. Still, we see a slightly higher cost for the Dyna-Q algorithm, but the RL algorithm has fewer backorders than the exact solution.

Overall, we see the great potential and benefit for challenging situations in decision-making. We see that a heavy computational algorithm as reinforcement learning is not beneficial for straightforward cases compared to a simple heuristic. Obviously, if the problem size is applicable for solving the problem exactly, we highly recommend this. Nonetheless, we recommend to further explore the potential and benefits of reinforcement learning in this application.

In conclusion, we can summarize our results as follows:

Aggregate cost and backorder evaluation results of three solving methods in three scenarios

	Exact		**Greedy**		**Dyna-Q**	
Preference	Cost	EBO	Cost	EBO	Cost	EBO
Tactical	49.674,99	1.01934	73.964,67	2.11055	72.713,82	0.41588
Operational	81.234,29	0,27511	129.816,66	0,75386	118.789,64	0,30912
Equilibrium	70.685,31	0.35177	153.811,09	1.54363	92.401,19	0.30035

The solving approach, as used in this research, is suitable for small problem instances. In case we are interested in solving, e.g., multi-item multi-echelon instances, state- and action space will increase. Using a nonstationary Value Function Approximation (VFA) or Deep Neural Networks (DNN) might suffice for these problems. In case the amount of data becomes even larger, approximating the optimal solution by combining sub-optimal policies requires lots of computational power. Therefore, advanced methods are required for solving these problem instances.

After solving the scalability issue for an actual problem instance, it is exciting to determine the possibilities of implementing an alert generation tool and the DSS into a Digital Twin (DT). The IAC can benefit from this because the Digital Twin brings ERP data and, for example, strategic insights in order processing together. In other words, the visibility leads to better insight and control in cost behavior of interventions. This can help understand and monitor the supply chain's behavior by continuous improvement. Therefore, the relationship between the operational and tactical level becomes clearer.

In conclusion, a reinforcement learning algorithm can benefit operational decision-making by creating insight into the operational planning horizon's expected costs. Because of the adaptive learning abilities of planning and learning algorithms, stochasticity in demand, returning repairs, and returning customer components can be processed without performance loss. Although the current experimental setting was relatively small, we see great potential in the application of reinforcement learning in service control towers. Therefore, we recommend researching the possibilities of solving more complex problem instances. We expect that by implementing a similar DSS, the *black box* of decision-making becomes more transparent.

Table of Contents

List of Figures

List of Tables

List of Abbreviations

A2C	Synchronous Advantage Actor-Critic
A3C	Asynchronous Advantage Actor-Critic
AC	Actor-Critic
ADP	Approximate Dynamic Programming
AI	Artificial Intelligence
AOG	Aircraft on Ground
CMA	Component Maintenance & Availability
CMRO	Component Maintenance Repair & Overhaul
DL	Deep Learning
DNN	Deep Neural Network
Double DQN	Double Deep Q-Network
DQN	Deep Q-Network
DRL	Deep Reinforcement Learning
DSS	Decision Support System
DT	Digital Twin
DTSC	Digital Twin Supply Chain
EBO	Expected Backorder
EP	Exchange Program
Fmv	Fair Market Value
IAC	Independent Aerospace Company
IID	Independent and Identically Distributed
IoT	Internet of Things
IP	Inventory Position
IPC	Illustrated Part Catalog
MAB	Multi-Armed Bandit
MDP	Markov Decision Process

ML	Machine Learning
MRO	Maintenance Repair and Overhaul
OEM	Original Equipment Manufacturer
OH	On-hand
OPP	Operational Planning Professional
PPO	Proximal Policy Optimization
RL	Reinforcement Learning
SCT	Service Control Tower
SDP	Stochastic Dynamic Programming
SLA	Service Level Agreement
SU	Serviceable Unit
TAT	Turnaround Time
TD	Temporal Difference
TRPO	Trust Region Policy Optimization
UU	Unserviceable Unit
VFA	Value Function Approximation

1 | Introduction

This Master's thesis includes a research for an Independent Aerospace Company (IAC). The company offers services to regional, commercial, and military aircraft. With Component Maintenance & Availability (CMA) programs, Maintenance Repair and Overhaul (MRO) components, and defense-programs is IAC aiming for excellence. This first chapter will further explain the company structure, the content of CMA-programs, and the associated challenges. As a result of the challenges raised by the IAC, this chapter presents the research approach and methodology.

1.1 The Company

This information is removed from the public version of this Master's thesis

1.1.1 Component Maintenance & Availability Programs

The CMA-program of the Independent Aerospace Company (IAC) provides repairs, spare parts, and reliability management for contracted customers. Through different service contracts, customers have unlimited access to approximately 7.500 components in a so-called *pool* of spare parts. These components are necessary to keep the aircraft in the air. To provide a good service, the program provides the following three points for the customer: *guaranteed availability*, *avoid downtime*, and *predictable costs*.

The customer has great benefits from the CMA-program since all parts come from a single source. The customer only has one point of contact, and that is the IAC. Besides the advantage that there is one integrated service provider, other advantages are prices and delivery times.

By using long-term contracts and risk-sharing frameworks, such as cost per hour or flight hour contracts, the costs are known to the operator in advance and are therefore predictable. The long-term contracts are beneficial to the customer in delivery since there is an integrated service provider network. Because Schiphol is easily accessible from all over the world, combined with the IAC's years of experience and strong inventory models, the company can serve customers within 24 hours. Besides shipping from Schiphol, where the central pool is located, the IAC has satellite hubs to assist. These are located in LaGrange and Singapore. By using the multiple locations on different continents, the IAC creates a global footprint. This leads to an increase in availability since the company is from all locations nearby the customer.

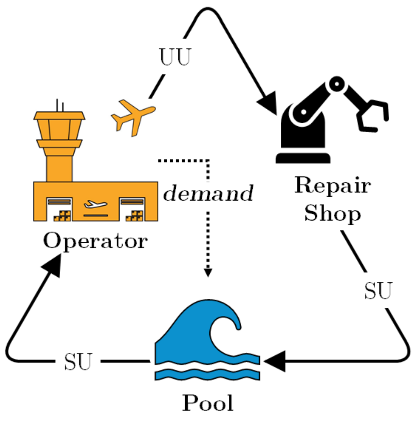
To make the CMA-program, and thereby the spare part pool, successful, the IAC tries to keep the turnaround time (TAT) as low as possible. The TAT is measured as the time elapsed between the moment that the malfunctioning component is removed from the aircraft until the moment that the component is repaired and is ready to be stored as spare unit (Kilpi & Vepsäläinen, 2004). The exchange process can be described as a loop, whereby the company and the operator exchange a serviceable and unserviceable component. The mentioned moment of submission of the TAT is when the unserviceable component is returned from the customer to the repair shop of the IAC. Although the exchange process's completion time is the moment that the component can be used in the spare pool, the moment is not fixed. This is because the completion time is dependent on the service type chosen by the customer. Each service type has a different completion moment. The CMA-program can be divided into three different service elements: exchange, maintenance, and lease services. These three types of service contracts are explained briefly and visually supported in Figure 1.1. The regular product streams are visualized with solid lines, optional product streams with dashed lines and communication streams with dotted lines.

Exchange Services

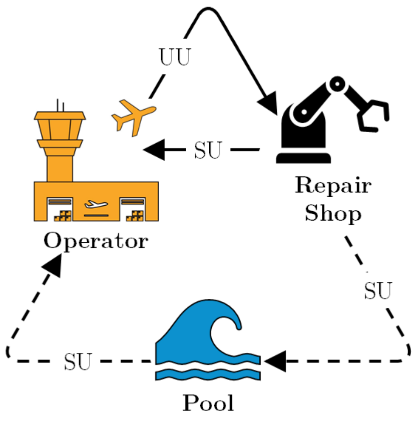
The first element of the program is the Exchange Service. Within this service program, customer demand (dotted line) will trigger the exchange process's start. In case of a malfunctioning component, the customer can order a new, functioning component. The customer request of a Serviceable Unit (SU), is met from the spare part pool. Next, the customer sends the malfunctioning component, an Unserviceable Unit (UU), to the IAC's repair shop. The IAC expects all customers to be sophisticated in sending the UU back in time, such that the company has sufficient supplies to preserve the high service levels for all customers operating in the CMA-program. This applies for all different elements of the CMA-program. The repair shop will restore the UU and transfer the part back to the pool whenever it is serviceable again. All movements of components are visualized with a full line in Figure 1.1a.

Maintenance Services

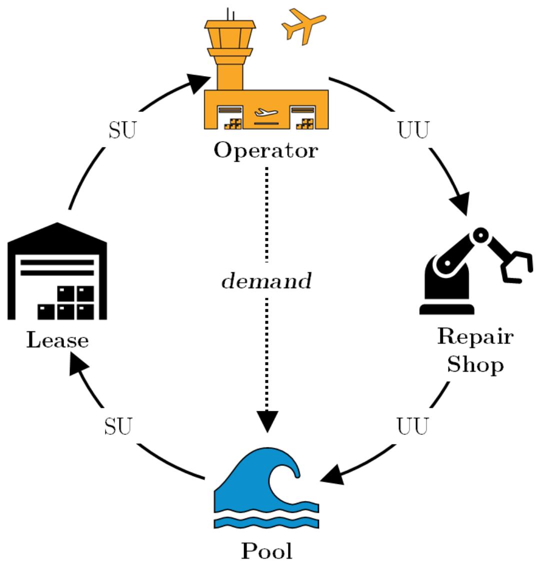
The second CMA-program is the Maintenance Service. With this service contract, the customer decides not to use the spare part pool initially. With this type of contract, the customer sends a UU to the repair shop whenever it is malfunctioning. There, it will be repaired and shipped back to the customer as SU. As mentioned, this process should be fulfilled within the contractual agreed TAT. However, in some cases, the repair time exceeds the agreed turnaround time or, in the worst case, cannot be repaired. When this happens, the customer receives a serviceable unit from the general spare part pool. Subsequently, the IAC uses the repaired component for the replenishment of the pool. This flow is visualized in Figure 1.1b. Note that when the operator gets a SU from the pool (dashed line), the regular shipment from the repair shop to the customer (solid line) will expire.



(a) Exchange Services



(b) Maintenance Services



(c) Lease Services

Figure 1.1: CMA-programs

Lease Services

The last commonly used service contract is the Lease Service. Specific components are essential to have quick access to for a customer. If such component malfunctions, this ensures that the aircraft can no longer fly. The Aircraft on Ground (AOG) status is an expensive situation for the operator; therefore, they want to prevent this from happening. For this reason, customers can decide to use a lease service. The customer can place a spare part from the pool in their own on-site stock. With this lease stock, the customer can take a SU directly from the on-site stock and install it in their aircraft in case a part is down. The customer does not have to wait until the new SU is delivered since it is already on-hand. The customer sends a request for a replenishment (dotted line) of their on-site lease stock to the IAC and sends the UU back to the repair shop. From the repair shop, the repaired part will go back into the pool. This process is visualized in Figure 1.1c.

1.1.2 Summarizing the CMA-program

The three types of service programs, as discussed above, ensure that the downtime will be minimized for the customer. Either when the customer receives the parts from the central pool, an on-site stock, or from the repair shop, the IAC endeavors to provide unlimited access to serviceable and high-quality components. To assure that the repair time does not exceed the contractual agreed TAT, both the IAC and customer must deliver the components in time. Furthermore, since there is one integrated service provider, the costs are predictable at competitive rates. The controlled costs are complied with by risk-sharing frameworks. In addition, the process requires enhanced logistics and innovative maintenance to succeed. In this way, the IAC ensures to cater to the customer and maintain the main pillars: *guaranteed availability*, *avoid Downtime*, and *predictable costs*.

Currently, the IAC works with two operational planners. In Chapter 2, we will further explain the current situation. These operational planners ensure that the parts are available and that the service levels can remain high, as agreed. This is done by using a Service Control Tower (SCT). The SCT provides real-time insight into the performance of the CMA-programs. A further explanation of the operation and application of SCTs will be given in Section 3.1.

1.2 Research Introduction

The IAC states that the current way of working is not creating difficulties or obstacles. However, the company asks for knowledge regarding the connection between operational and tactical planning in service control towers. The need concerns management-, process- (such as alert generation), and decision levels. This request for knowledge originates from the research of Topan, Eruguz, Ma, van der Heijden, and Dekker (2020). This study reviews the possibilities of the operational spare parts in a service control tower setting. Since a strong conjunction between operational and tactical planning can lead to higher service levels, it is interesting to review further possibilities of decision-making in service control towers (Topan et al., 2020).

The IAC would like to explore the potential possibilities of Artificial Intelligence (AI) for operational planning. Therefore, my company supervisor started a Ph.D. program to close the knowledge gap. For this program, the possibilities of incorporating artificial intelligence in a service control tower setting is investigated. The research involves AI since it can add human reasoning to algorithms. Through the interaction with the environment, the AI algorithms can incorporate the exceptions better than other decision support systems. Furthermore, combining planning and learning models shows strong potential in decision processes. With relatively limited data, AI is capable of adapting through progressive learning.

This Master's thesis will carry out contributory research to Ph.D. research. The analysis related to alert management, the preliminary step of this thesis, is carried out by another University of Twente student. He investigates a proactive monitoring tool on the alert generation from the service control tower, supported by Machine Learning. The research concerning the alert generation will, therefore, be out of scope of this thesis.

Tactical planners predict the inventory models in advance for the parts of the CMA-programs. However, an unforeseen deviation may arise in day-to-day planning. In this case, the control tower generates an alert. When such an alert is generated, an intervention must take place on the operational level. These interventions are, for example, buying a component or doing nothing at all. The operational planners are responsible for determining which intervention is most suitable for the given states. The intervention is partially supported by data but mostly based on intuition or gut feeling. Furthermore, the consequences of the decisions are unknown in the current situation.

In short, due to a knowledge gap concerning the relationship and interaction of the operational and tactical planning, this thesis reviews the possibilities of decision-support. By involving artificial intelligence algorithms in the decision-making process, operational interventions can be supported while incorporating and considering the decision-making process's stochasticity.

1.3 Assignment Description

The previous section briefly explained the current way of working for the operational planning professionals at the company. In this section, we will define the corresponding core problem. Because the entire problem has a broad scope, we limit this thesis's scope to create a valuable contribution for the IAC. A research question and five sub-questions support the scope of our research.

1.3.1 Aim of our Research

As described, in the current state, the decision-making concerning the interventions is mostly based on intuition. Therefore, there is no insight into the consequences or the impact of the interventions for the long-term. The aim of this thesis, and hence the core problem that we address, is:

'The Independent Aerospace Company does not use an operational decision support system, causing that the impact of the decision-making on costs, such as backorder and holding costs, cannot be determined adequately.'

1.3.2 Scope of our Research

To solve the mentioned problem, we will look at the input-output relationship of the planning processes. Tactical planning presents the expected long-term necessary stock levels. Due to fluctuations, deviations, and uncertainty, the predicted stocks change over time. To absorb these changes, the IAC carries out interventions at the operational level. Here, tactical planning is used as input for decision support. The output of operational decisions could subsequently improve tactical planning if a permanent change needs to occur.

Nonetheless, the relationship between operational and tactical planning is unknown if there exists any. This means that the feedback relationship exists in a *black box*. The *black box* element is defined in the literature as a system where a visual element enters an imaginary box, from where different observable outputs can emerge. However, the black box approaches lack transparency, i.e., the reasoning and support of decision-making are not clear.

The *black box* contains all uncertainty related to the decision-making, making it challenging to fit a subsequent decision support system to provide reliable advice. Visualizing the *black box* creates insight into the interaction between tactical and operational planning, which can likewise improve tactical planning. Nevertheless, because the CMA-program works with a closed-loop environment, where components do not leave the spare part loop easily, having a positive impact on the tactical level by operational decisions is difficult. Therefore, we will use the tactical level as a guideline in operational decision-making.

Most of the time, in operational decision-making, tactical levels are taken into account while making decisions. However, this is slightly different for inventory management because, in a stable environment, the inventory decreases over time by demand. Therefore, operational decisions do not have an enormous direct impact on tactical planning levels. Within the closed loop, all decisions have a direct impact on the inventory position. Therefore, we will only research the impact on operational decision-making, supported by rewards in the long-term for this research. The investigated tools should advise in decision-making for spare parts of the CMA-program of the IAC.

The company indicates that they want to determine the potential of planning and learning algorithms in a service control tower setting. We focus mainly on planning and learning algorithms, or Reinforcement Learning (RL), because these algorithms can learn optimal decision-making for sequential decision problems. The algorithm learns optimal rewards for different situations within these problems, and it can learn the corresponding policy. Therefore, this research will investigate the opportunities for these models for the given problem. Nonetheless, to validate the reinforcement learning algorithm's potential and strength, other solving approaches will be reviewed as well.

Additionally, to the research of the planning and learning algorithm, this thesis will determine the potential of implementing the algorithm in a Digital Twin (DT). A Digital Twin is a realistic model of the real system, which can support simulation, optimization, and control (Cronrath, Aderiani, & Lennartson, 2019). The system can also perform decision-making for different actions by accessing their own behavior in interaction with the environment. Furthermore, it allows a faster adaption, implementation, and improvement of operations (Rosen, Von Wichert, Lo, & Bettenhausen, 2015).

1.3.3 Research Questions

The research problem states that there is a desire for more knowledge and insight into the impact of the operational decisions on the long-term yield. The IAC wants to make the decision-making measurable and visual, supported by data. Additionally, because of the stochasticity in the decision-making process (due to, e.g., exceptions or variation in demand lead times), the IAC wants to determine the computational power of reinforcement learning. Therefore, the main question that we will answer is:

'In what way and to what extent can a reinforcement learning algorithm improve operational decision-making while incorporating long-term yields?'

In order to answer the main research question, we defined five sub-questions that will provide partial answers. The substantiation of all questions is provided to give a clear overview of the intention and approach.

The first topic that we will address is the description of the current state of inventory planning and decision-making. In Chapter 2 we provide more information about the current working methods, performance, and decision priorities. This will give a better insight into the current deficiencies and required capabilities for our model. We translate this into the first sub-question.

1. Which tools, instruments, or methods does the Independent Aerospace Company use for operational decision-making, and what are the corresponding performances of these techniques?

Next, we will further discuss the interventions that emerge from Chapter 2. We will investigate the overlapping interventions that both the IAC and the literature use. To improve the IAC's performance, we look into alternative decision-making approaches found for comparable interventions in the literature. The second sub-question will provide an answer to this matter. We will address this question in Chapter 3.

2. What are overlapping interventions of current operational decision-making within the IAC and literature, and what are possible additions for the IAC?

Additionally, in Chapter 3, we look for the different algorithms that the literature uses in reinforcement learning. As far as we know, there is not much literature available concerning reinforcement learning in inventory control or service control towers. Therefore, we compare decision-making in different environments to determine the added value of reinforcement learning compared with other algorithms. Based on the comparison, we choose an algorithm that can help us create a planning and learning model for our purpose. The algorithm should be suitable for the inventory control problem, with all stochasticity included. Furthermore, the algorithm should be applicable as a decision support system. Therefore, the third sub-question is:

3. Which planning and learning algorithms are used in reinforcement learning according to the literature and which are applicable for the IAC's purposes?

To examine the decision support system's performance and quality in different scenarios, we model the decision support system in three ways: an exact solution approach, a simple heuristic, and a reinforcement learning model. The exact and heuristical approaches function as benchmarks for the reinforcement learning algorithm. The reinforcement learning model, used in the *Proof of Concept*, is chosen based on the literature review. From the desire for this information, the fourth sub-question arose. In Chapter 4, we discuss the model design for our solution, followed by the experiments, performance, and statistical findings in Chapter 5.

4. How can we use reinforcement learning for the IAC's purpose, and what performance can we expect from this algorithm compared to an exact- and heuristic solution approach in terms of cost-benefit and inventory policy?

Finally, since the decision support system is only designed (Chapter 4) and tested (Chapter 5) with a Proof of Concept, we will also describe the possibilities of implementing the system in a realistic data-set environment by encountering the possible scalability issues. Implementing the decision-support system in a Digital Twin can be useful in this situation. A Digital Twin can help create insight and improve decision-making in spare part management in the service control tower. Therefore, we review the possibilities of implementing the found reinforcement learning models into a Digital Twin. Chapter 6 explains the possibilities by answering the last sub-question.

5. How can we implement the decision support system in a realistic and larger decision-making process, using a Digital Twin?

1.4 Research Methodology

Earlier in this chapter, we described that we want to investigate a shortcoming in knowledge. We refer to this type of problem as a knowledge problem. This type of problem can be solved by conducting research. In case of an action problem, changing or improving the situation can solve the problem. We use the research cycle to structurally solve a knowledge problem (Heerkens & van Winden, 2012; Cooper & Schindler, 2011). This cycle consists of eight steps and is shown in Figure 1.2. Since the first three phases of the research cycle have already been clarified in this chapter, they do not need further explanation. However, to give a more in-depth view of the research cycle, the last five phases of the cycle will be explained next.

First, we will look at the research design. This research aims to gain more information about using an operational decision support system in a service control tower for spare part logistics. We will obtain this information by conducting an extensive literature study. We will define the basic concept, different learning dimensions, and different reinforcement learning algorithms to measure the application and the influence of each of these algorithms. Furthermore, we will translate the most suitable model for our problem definition and operationalize the required variables. We include the abstract variables that are, for example, responsible for the degree of preference. In this way, we can create insight into the impact in both the short and long-term.

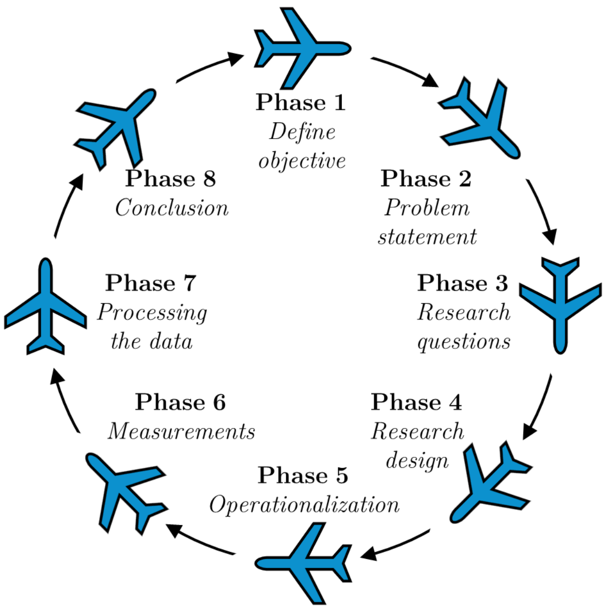


Figure 1.2: The eight phases of the Research Cycle

For the reinforcement learning model, we have to define a framework with all different states, rewards, and the corresponding environment. Since these are not all given as standard variables, we have to make sure that, e.g., all rewards are measurable in the environment and suitable for the algorithm.

The measurements of the study will then have to be tested for reliability and validity. Reliability will be examined by comparing the operational decision support system with outcomes of the exact and heuristic solution approach. We also test validity. We do this in two areas: internal and external validity. Internal validity must be guaranteed by, for example, preventing self-selection. Since this research is relevant for the Ph.D. research, application for the IAC, and also for other follow-up studies with this point of interest, the external validity is covered.

Finally, from the processed experiments with the three different approaches, we can conclude the main question. The conclusion will be based on the experiments with the decision support system, where the comparison of the different algorithms is vital. Within this comparison, we look at which of the selected algorithms can provide the best advice for the IAC's purposes. The conclusion we draw in this thesis applies to the IAC's situation and should provide a complete answer to our research question.

In summary, the research cycle is a solving approach for situations where knowledge is necessary. As described in the literature, the knowledge problem is a description of the research population, the variables, and (if necessary) the relations that need to be reviewed (Heerkens & van Winden, 2012). The research cycle is not about changing situations but understanding. The knowledge problems often come from an action problem, which lacks information about, e.g., solving the problem. By going through all eight steps of the cycle, we create a structured research approach to obtain the desired information.

2 | Current System Analysis

This chapter provides a clear view of the current situation at the Independent Aerospace Company (IAC). First, in Section 2.1, we briefly introduce the IAC's supply chain network. Next, in Section 2.2, we discuss the alert generation. Since another Master's thesis already studies alert generation, we will not review this topic in depth. We use this section for introduction purposes only. Next, we review the current interventions that the operational planning professionals use in Section 2.3. Subsequently, we will review the current decision-making process in Section 2.5 to describe the considerations in the decision-making process. Section 2.6 discusses the operational intervention control's current and potential performance. Finally, we discuss the data set provided by the IAC in Section 2.7. We obtained this chapter's information through an interview with the operational planning professionals. The interview questions can be found in Appendix A. With this chapter's knowledge, we can answer the first sub-question.

2.1 The Supply Chain Network

The IAC currently works with a two-echelon network, schematically visualized in Figure 2.1. In this multi-echelon network, the central warehouse is located in Hoofddorp. Additionally, the regular local warehouses are located in Singapore and LaGrange. The operators of the CMA-program receive their components from both the central warehouse as the local warehouses. The delivery from these warehouses is a proactive procedure. Between LaGrange and Singapore, lateral transshipments can be used (blue dashed arrow). Besides the active delivery, the IAC uses other operators' lease stock as an emergency option (red arrow). This action is reactive.

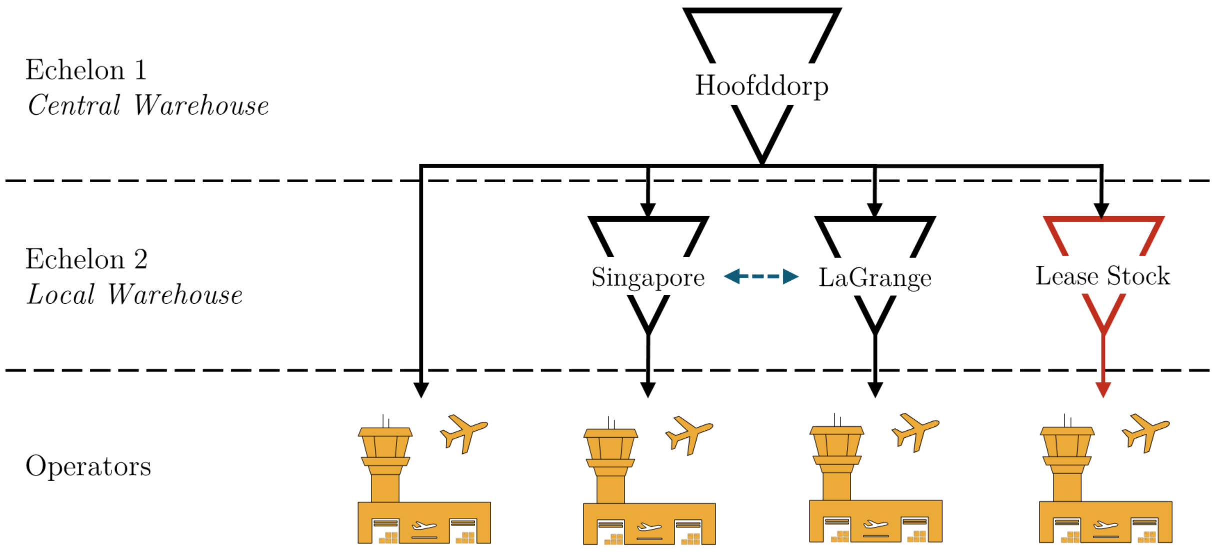


Figure 2.1: Multi-echelon network of the Independent Aerospace Company

Additionally to the local warehouses, the IAC uses a *quarantine warehouse* and a *commercial warehouse*. The quarantine warehouse stores components that are not ready for direct use. I.e., the component can have an expired license or can be malfunctioning. The IAC proactively stores components in the quarantine warehouse for future use. The quarantine warehouse is, therefore, part of the CMA-program. The IAC uses the components from the quarantine warehouse partially or while they are still intact. We will explain the IAC's partial use of components in Section 2.3.6. The commercial warehouse is an independent warehouse targeting the commercial market. Since the IAC has no Service Level obligations to customers on the commercial market, the IAC can use the commercial warehouse components to fulfill the demand of the CMA-program. We will further explain the use of the quarantine and commercial warehouse in Section 2.3.2 and 2.3.3.

2.2 Alert Generation

As described earlier, this thesis leaves the alert generation out of scope. Nonetheless, to create a clear view of the chronological processes, we briefly explain how the alerts are generated, in which situations these occur, and what the nature of the alert situations are.

The operational planning professionals, from now on referred to as OPP, receive an alert whenever a situation occurs that requires attention. Alerts occur in situations that the system expects interference is necessary to successfully continue the process. The alert generation can occur everywhere in the CMA-program cycles, as described in Section 1.1.1. The required intervention for the alert is dependent on the alert's nature. For example, in the *Exchange Service* contract loop, an alert can arise when a customer order arrives and there is no stock available. Other, with the *Maintenance Service*, the alert-process starts when the malfunctioning part arrives at the repair shop. Note that the described alert points are for explanatory use only. In the real situation, all steps in the processes can generate an alert with the corresponding interventions. The ERP system generates alerts. Besides the different origins and starting points of the alerts, there are different circumstances for which the alerts occur. From interviews with the OPPs, we conclude that there are three main situations in which the ERP system or OPP notices an alert.

The first alert situation is *direct backorders*. This alert has the most significant impact on the performance of the CMA-program. A backorder can be defined as an order for a component that cannot be fulfilled due to lack of availability (Sherbrooke, 2004). The customer still requires the component, so the IAC needs to meet the customers' demand. Due to the significant impact, there is little time to respond and the OPPs should undertake immediate action. A consequence of the limited time is that the related costs of interventions in this alert situation are usually high. Often the only options in this situation are procuring or leasing the part from other suppliers.

Secondly the *high Turnaround Time*. As the first chapter describes, both the IAC and customer should be sophisticated in delivering parts within the agreed time intervals. Nonetheless, the TAT can deviate at two different places in the CMA-loop: the repair shop and the customer return. In case the TAT at the repair shop is too high, the OPPs will not receive a direct alert. While reviewing other problems, the high TAT becomes visible for the OPPs. With, e.g., prioritizing or expediting repairs, this alert can be corrected. In case the customer returns the component later than agreed, the pool runs with fewer components. Therefore, the probability of a backorder to occur grows.

Finally, an alert occurs when there is *no stock on-hand*. The alert for this situation is not generated by the ERP system but is reported by a tactical planning tool. This alert does not directly impact the performance of the CMA-program, as direct backorders. However, there is a high probability that a direct backorder occurs soon. Therefore, a quick response is necessary.

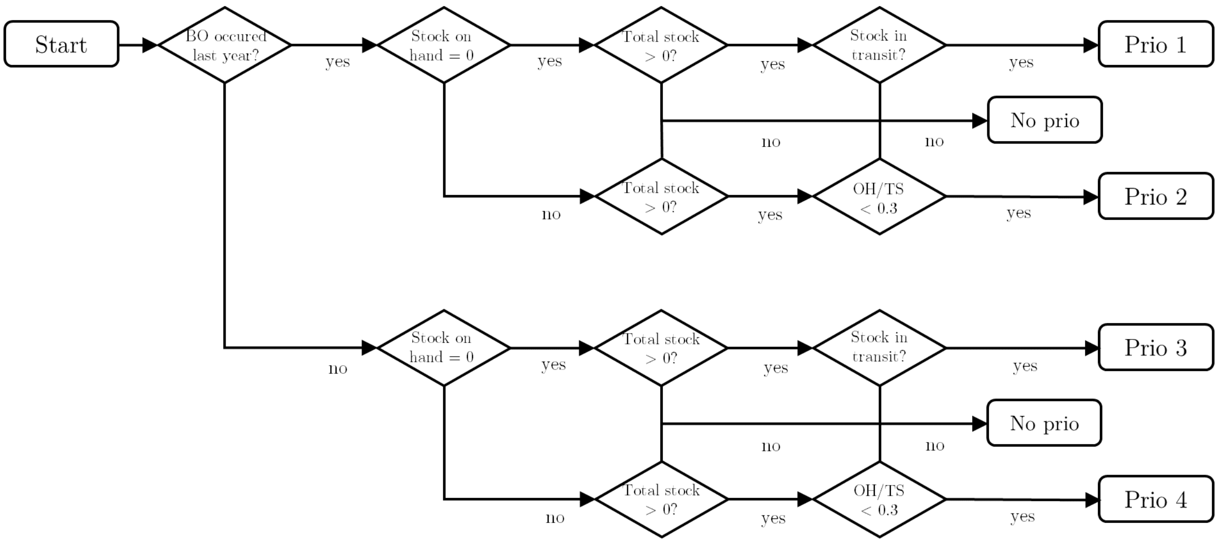


Figure 2.2: Flowchart of alert generation and prioritizing

The OPPs indicate that commonly two direct backorder alerts occur in a week. However, due to COVID, there are fewer alerts since the demand is lower for components. In addition to the mentioned alerts, a priority list is generated. The priority list reflects the current status of the supply chain. The list helps the OPPs in taking proactive interventions to reduce the likelihood of backorders soon. The list is generated by a business ruling that the OPPs run themselves. Since both OPPs can run the priority tool whenever they want, there is no fixed refresh rate for the priority lists.

The generated alert lists are prioritized based on a fixed business ruling. With the given priority list, the OPPs will work on the alerts from *prio 1* to *prio 4*. The business ruling is visualized in the flowchart in Figure 2.2. The first step of the business ruling is to determine whether there has been a backorder for the component in the last year. When a backorder has occurred, the ruling is more conservative such that this can be prevented. I.e., if the IAC is performing well during the year on a component that generates an alert, there is no need for a high priority. However, if a customer's request is missed within the last year, the alert receives a higher alert to prevent this from happening again. In case a backorder has occurred, the alert will be a *prio 1* or *prio 2*. Since the business ruling is both the same after the backorder step, we will only explain the steps for the first two priorities.

Secondly, the system determines if there is stock on-hand. In case there is no stock on-hand, and the inventory position is nonzero, the ruling determines if the stock is in transit. In transit stock means that there is stock in the pipeline which is not available for use. The component can be in repair or can be a core unit at the operator. Since this stock is not free for use, the alert receives a *prio 1*. However, it is also possible that alerts receive *no prio*. This happens when the planned inventory position is equal to zero or if the planned inventory position is larger than zero and the stock is not in transit.

If the on-hand stock is not equal to zero in the other flow, the ruling determines again if the inventory position is nonzero. If the planned inventory position is equal to zero, the alert will receive *no prio*. Otherwise, the ruling determines if the on-hand stock (OH) divided by the inventory position (IP) is less than 0.30. Meaning that if less than 30% of the total inventory position is on-hand stock, the alert receives a *prio 2*. As well as if 30% or more is on-hand stock, the alert receives *no prio*.

The situations, as described above in the flowchart, apply to all inventory problems. The alerts arise to prevent backorders from occurring. Nonetheless, some alerts occur in case, e.g., a repair takes longer than expected, or the operator did not return the component yet. The mentioned components are not available for use and are, therefore, in transit. This leads to a larger TAT than expected. The final scenario we discuss is the backorder from the outside. The scenario can occur when the commercial warehouse has demand for a component and has no stock available. In case this happens, the OPPs can decide to analyze it as a loss of sales case. I.e., using a component from the CMA-program to fulfill the demand or lose the sales and, therefore, profit.

Altogether, alerts are generated in several different situations. This happens when the planning system expects that a component cannot move through the CMA-process without interference. We described three main situations for which an alert can occur; direct backorders, high Turnaround Times, and no stock on-hand. These three situations can occur in different scenarios. Via a given business ruling is for each alert, the priority is determined. Additionally, based on the business ruling priorities, can the OPP work the priority list top-down.

2.3 Interventions

The possible interventions that the operational planning professionals (OPP) can take depend on the alert situation. This section review the different interventions that the IAC uses. Besides the individual interventions, there is also a possibility for combining multiple interventions. We will discuss the interventions, the execution, and in which situations the interventions are used. Below, we explained the interventions in a similar order as the checking preference by the OPPs.

2.3.1 Do Nothing

When an alert arises, the planners will first consider whether an action is needed or if regular transhipments are possible. The OPPs check the accuracy of the alerts. It is possible that, for example, the customer desk oversees a component in one of the IAC's warehouses, leading to a false alert. Therefore, by doing nothing, the IAC can still fulfill the customer's demand. Nevertheless, in certain situations, when the alert is correct, OPPs still decide to do nothing. An example of do nothing is when the component is not crucial for the operator, and the planner expects that the component will return soon.

2.3.2 Discard or Store

This intervention occurs when an alert arises at the end of the loop, the repair shop. In case an unserviceable component comes to the repair shop, the OPPs have to decide whether to repair the component or not. Additionally, the OPP can discard the component or store the component in another warehouse. Three different options are discussed next.

Discard

The first option is to discard the product. In case there is, for example, an overshoot in components, or the repair costs are too high, the OPPs can decide to discard the component. Often this is an expensive action since the components still have a book value. Therefore, this is not the most desirable option in most situations.

Quarantine Warehouse

The quarantine warehouse is a storage location where items are stored that are currently not operational due to, for example, expired licenses or malfunctioning parts. There might emerge a surplus of components in the regular CMA-warehouse due to, e.g., changes in demand. Because all components in the CMA-warehouse need an examination occasionally to prevent the shelf life from terminating, regularly checking the components from the surplus causes unnecessary costs. Therefore, placing components in the quarantine warehouse is a cost-effective measure.

Commercial Warehouse

Finally, relocating the component to the commercial warehouse. In case the repair of a component is not too expensive, the OPP can decide to repair. However, if an overshoot of a component in the CMA-warehouse occurs, a relocation to the commercial warehouse is possible. From there, the component is still useful to the commercial department of the IAC.

2.3.3 Lateral Transshipment

The next option that is discussed is the lateral transshipment. The OPPs can relocate a functioning component from one to another location. From the interview we conclude that there are three different options to relocate the components from. As discussed in the first chapter, comes the regular CMA stock from the pool. Besides the pool, components from the commercial warehouse, quarantine warehouse, and the lease stock from other operators are used.

Commercial Warehouse

First of all, we will discuss the commercial warehouse. The IAC has two separate warehouses for the commercial exchanges and the CMA exchanges. Since the IAC must deliver the components in time to the CMA operators, the IAC can choose to use a commercial warehouse component to fulfill the demand. When the CMA pool is recovered from the backorder, the component can move back to the commercial warehouse.

Lease Stock from other Operators

The second transshipment option that the IAC uses is temporarily relocating the lease stock from other operators. In case the IAC has no available components, and the repair combined with returning to the customer takes too long, the planners can choose to ask another operator that has a lease stock to make their components available. This intervention is only useful when an operator has a lease stock near the operator with a malfunctioning component.

Quarantine Warehouse

Finally, the OPPs use the components in the quarantine warehouse for relocation. The quarantine warehouse stores components with an invalid license or that are not repaired but still useful. The process of relocating components towards the quarantine warehouse is discussed in Section 2.3.2. However, the process can also be the other way around. In case a backorder occurs, the OPPs can decide to restore a component from the quarantine warehouse. With this, the quarantine warehouse's components are pulled into the CMA-pool again.

2.3.4 Interchangeable Parts

Some components are interchangeable with each other. Interchangeable components are components that are, for practical purposes, identical (Curley, 2016). I.e., the components are not one-on-one the same; the components are an older version of the part number, or the configuration-settings are slightly different. The interchangeability can be a *one-way* or a *two-way* interchangeability. The operator obtains an Illustrated Part Catalog (IPC) from the OEM. Since part-numbers can change due to modifications, the IPC describes which part-numbers can be installed into an airplane.

Furthermore, the IPC indicates the relationships between different part-numbers. When two components have two-way interchangeability, it means that the operator can accept both components for repair. For the IAC, it does not matter which component is on stock, making it easy in inventory planning. However, in one-way interchangeability, there is a limitation in which part-number can be installed. Because there is, e.g., a significant technical difference in the product, whereby one-product is better than the other, the operator is restricted to use one product. The one-way interchangeability gives the IAC complexity in inventory management as well. The IAC can decide in this situation only to keep the high standard product on stock, such that the availability remains high.

2.3.5 Prioritizing & Expediting

The repair loop can also benefit from performing an intervention. As soon as an alert arises for a component currently in the repair shop, the component can be prioritized such it is ready the same day. Besides prioritizing a repair, OPPs can ask the repair shop to expedite the repair. Both alternatives occur regularly for internal repairs. External shops are usually not willing to perform these interventions.

The location of the internal repair is of great importance here. Due to the difference between, e.g., Hoofddorp and LaGrange, alerts that arrive in Hoofddorp in the morning cannot be processed immediately. Since OPPs have to wait until the shop opens in the morning in the US, lots of time is lost.

2.3.6 Cannibalization

Cannibalization is a frequently used intervention with internal repairs. The IAC uses partial components from the quarantine warehouse to repair an UU. This is a quick, circular, and cheap possibility of restoring the malfunctioning components since the repair reuses parts without procuring new parts. External shops are often not willing to perform this intervention for economic reasons. External shops want to make money on the materials, and by performing the cannibalization, this is restricted. Furthermore, this disturbs the repair process of the external shop.

2.3.7 Expedited Shipment

In case an operator uses the Exchange or Lease Services, as described in Section 1.1.1, and no stock is available, the OPPs can use an expedited shipment. With this intervention, a component currently in the repair shop will be sent directly towards the customer instead of first going back to the spare part pool. With this action, transportation and handling time can be reduced.

2.3.8 External Sourcing

As explained in Section 2.2, there are situations where corrective interventions are necessary. With corrective interventions, a quick response is required. There are two emergency responses that the OPPs use whereby a component from an external source is used.

Vendor Exchange

The first option is to perform a vendor exchange. With a vendor exchange, the CMA-program leases a component from another vendor until the backorder is solved. This is an expensive option since the IAC pays exchange fees for the lease and pays for a quick repair.

Procure New Components

Another expensive emergency solution is buying a new component. When a vendor exchange has to be entered into several times a year for a particular component, it is cheaper to buy a new component. However, when the new component is bought in a backorder situation, there is not much time for negotiating for a fair price. The OPPs want to try to prevent this situation from happening by updating the tactical planning regularly.

2.3.9 Combining Interventions

The OPPs indicate that several interventions are also combined during an alert's processing. This is done for numerous reasons. First, more interventions are used simultaneously to increase the chances of success. Exploring the different options gives the highest chance of success of the intervention. Furthermore, interventions can also be combined so that costs can be kept as low as possible. For example, when a vendor exchange is used, the IAC has to pay high exchange fees as long as the vendor exchange takes place. Therefore, the OPP will try to speed up the repair (expediting). This combination aims to keep the costs of external sourcing as low as possible and continue working with the IAC's components. Another situation could be that external sourcing of unserviceable units occurs such that cannibalization can occur.

2.4 Key Performance Indicators

During the decision-making regarding the interventions, the OPP take two important Key Performance Indicators (KPI) into account. The KPIs concern the costs and Service Level Agreements (SLA). Currently, the costs performance indicator does not include a direct threshold. For the IAC's purpose, we express the SLA in four different applications; customer, product, transaction, or work-scope level. The SLAs are customized for each customer. For example, in Forward Exchange contracts, the IAC serves the customer within 24 hours, and for transaction-level, the due date is on a fixed TAT.

For the Service Levels for availability, the OPPs use a tactical planning tool. Within this tool, optimal stock levels are determined following a Poisson distribution, such that the Item Service Levels are higher than 93%. Since the stock levels are discrete values, the calculated Service Levels are often close to 100%. The aim for the System Service Level is to keep it higher than 96%, but since the Item Service Levels are close to 100%, System Service Level is also close to 98%. To decrease the System Service Level, the IAC uses a greedy approach. The IAC lowers the most expensive components' inventory levels until the Item Service Levels of the most expensive items are minimal 50%. This greedy approach is performed one item at a time until the System Service Level is 96%.

The final SLA, concerning the work-scope, is related to the amount of work that needs to be done. We can translate the SLAs to the transaction level, so to the due date, to fulfill customer demands. The SLAs can be carried out from the contract or outside contract. Within the contract, the SLA gives a requested due date from the customer. Outside contract, the IAC uses the SLA as a promise date.

Since the IAC is a service provider, the main motive should be to achieve the agreed service levels. In case of a maintenance program, this refers to throughput times, for availability programs to the availability of components. However, motives for both OPPs are different. The first planning professional indicated that he looks at the lowest possible costs that keep the service levels above the agreed thresholds in the decision-making process. In contrast, the other OPP indicates that his decisions are motivated by availability-thought. His decision-making motive is to deliver to the customer in time to prevent downtime, no matter at what cost.

To summarize, the aim of the CMA-program is to guarantee availability, reduce downtime and predictable costs. To fulfill these aims, the OPPs take the KPIs costs and SLAs into account. Needless to say, the costs of interventions should be as low as possible to make the predictable costs more profitable for the IAC. The SLAs, in four different branches, are responsible for reducing the downtime and guaranteeing the customer's availability. The motive in the decision-making process for both OPPs is, however, contradicting.

2.5 Decision-making

As described in the previous sections, there are several options in interventions for the given alert situations. Considerations and motivations for decision-making, derived from the interview, are explained in this section. Furthermore, the connection between the alerts and interventions is reviewed to determine the challenges in this process.

2.5.1 Tacit Knowledge in Decision-Making

Although the interventions are related to the alert's nature, there is no fixed decision pattern for the OPPs. The motives of the OPPs in decision making is based on tacit knowledge. Many decisions are made from a gut feeling to pursue the mentioned KPIs. However, there are reasons and considerations for choosing a suitable intervention during an alert. The OPPs indicate that the two most important moments when an alert can occur are; as soon as a product comes in for repair and in the situation that there is no stock on-hand.

In both of the mentioned cases, the planners evaluate if the stock computed by the tactical planning is still correct after an operational intervention is performed. The OPPs evaluate this to determine whether tactical planning is incorrect or just a one-off case occurred. In case tactical planning is incorrect, an action should follow. By proactively solving problems, the OPPs try to prevent extreme measures.

No stock available

The first step that the OPPs take is to determine whether there is a component available in another warehouse. Since the part is in possession, it is often the simplest solution to move it from one location to another. Furthermore, because the OPPs indicate that the tactical planning tool is leading, stocks calculated by the tool should be sufficient. Therefore, provide the exchange rates a good insight per component whether the shortage occurs more often or if it is a one-off situation. In the most unfortunate situation, a vendor exchange is often used as a sourcing option for a particular component. If so, they will consider buying the part from other suppliers.

Repair cost evaluation

In case of a repair, other considerations are taken into account for finding the best solution. As soon as a part arrives at the repair shop, a repair quote could be provided first. This repair quote is necessary because not all operators are explicit in composing the reason for removal. With the new repair quote, the OPPs can see the expected repair costs for the specific component. Moreover, the reason for removal is an essential indicator for the OPPs. Based on the repair quote, the OPPs make a *Make-or-Buy* decision. Since there is also a third option, *discard*, an explanation will be given of all three decision-options. An example supports each option.

Make As mentioned, the repair quote is considered before deciding whether to repair it or not. From the ERP system, the average repair costs can be derived. The average costs from the ERP system are compared with the expected costs from the repair quote. When the repair quote is lower than the average costs, the component is always repaired. In case the expected costs are higher than the average costs, the OPPs will sift through the repair quote to decide to repair or not. Now, the buy and discard option are compared with restoring the component.

An example of the repair decision is for operators that fly in volcanic areas. Thence, these operators usually have more often malfunctioning components. There are components that, for example, operate on a scheduled maintenance interval of 10.000 flying hours. This means that the component must be checked after the scheduled interval. However, due to the operator's use, it can also happen that the component no longer works properly after 4.000 flight hours. For such components, OPPs should consider restoration of the component as described by the component's OEM.

Buy First, the repair option is determined. A new part can (almost) never be purchased at repair costs. However, due to forces from the market, prices change. With the trading department's help, the OPPs can monitor how prices are behaving closely. In case the repair costs are too high, the OPP can consider buying a new component from the market. Nevertheless, the OPP should take inventory and holding costs into account while making the buy decision.

As well as in the stock out situation, the planners evaluate the exchange rates. According to the OPP, 80% of the failures come from the same piece part in a component; the exchange rates provide a good indicator of whether a component should be repaired or replaced. In case the same part number repeatedly malfunctions, buying a new part might be a better decision in the long-run.

Discard Finally, we discuss the discard option. In case there is a substitute component available for the UU, which means there is a buy option or an overshoot in commercial or quarantine warehouse, the OPP can decide to discard the UU. This is interesting to keep costs low. In addition to the repair costs and the procurement price, book values must also be considered while making this decision. Some components still have a very high book value, which makes depreciating a worse option. For example, depreciating a component with a book value of $500.000 is discarding that value.

2.5.2 Relationship Between Alerts and Interventions

Although the OPPs have certain routines to come up with an intervention, the direct consequence from the alert and intervention is unknown. The decision-making is done based on a gut feeling and experience. However, the OPPs indicate that this feeling and experience can be passed on to others. Since the possible interventions are dependent on the alert's nature, there should be a consequence of the intervention that can be operationalized. This can be defined as a *hidden layer* in the decision-making process.

To clarify the *hidden layer* of the decision-making process, a schematic representation is given in Figure 2.3. For this example, we work with two different alert scenarios, denoted as scenario one and scenario two. Furthermore, we work with five different possible interventions for the alerts, denoted as a, b, c, d, e. As visualized in the figure, for each alert scenario, their effective interventions are given. However, interventions can also be effective for multiple purposes. Given an alert and an intervention, there is a motivation or consequence, which should support the decision-making. Currently, this motivation is located in the gray area, whereby the motivation should be driven by the financial and service levels KPIs. This motivation is carried by the experience and the implicit knowledge or expertise of the OPPs.

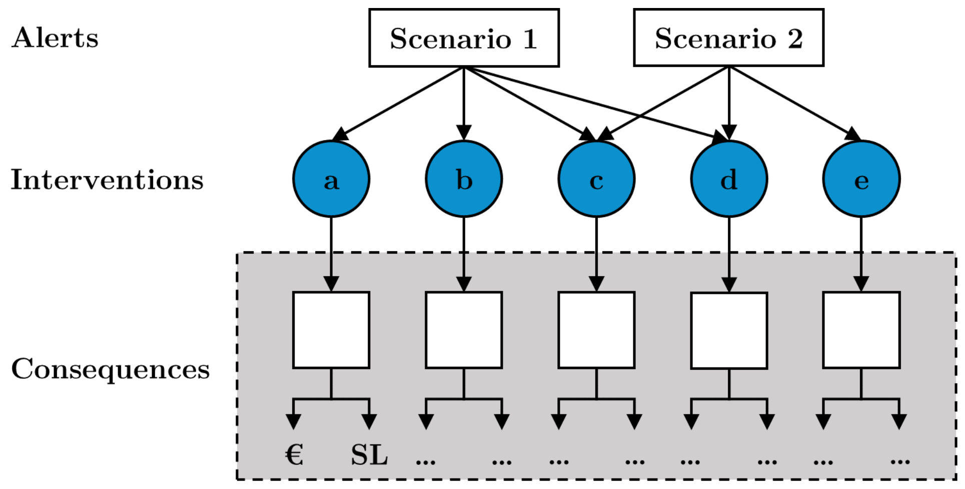


Figure 2.3: Motivation for decision-making

2.6 Aggregate System Performance

In this section, we will discuss the current performance of the decision-making of the IAC. This includes the quality of decisions, the current impact of the decisions on the long-term, and finally, the decision-making's potential performance based on the earlier mentioned KPIs.

2.6.1 KPI Output Measures

As mentioned in the previous section, the intervention's output depends on the implicit knowledge of the OPP, the hidden layer. Given the KPIs that the IAC uses, availability and costs, the decision should be based on these two points. Nevertheless, the quality is difficult to determine for the Operational Planning Professional (OPP). Although the quality is difficult to make measurable, both OPPs state that they make the best, and therefore right, decisions in their conscience.

Since the IAC is a service provider, the main focus should be on servicing the customer on the same day. This variable is relatively easy to measure for the OPPs. The OPPs have access to the current service levels, making it possible to predict the expected duration of interventions decently. Furthermore, if a company aims for a service level of 100%, the stock levels are enormous.

These high stock levels cannot be justified financially. Therefore, interventions should be taken with care to ensure sound financial decisions. As mentioned earlier, multiple different financial aspects (e.g., book value, market price, and repair costs) are considered during the decision-making process. Nevertheless, a decision's financial quality is currently difficult to determine for the OPPs. For example, the quality of a make-or-buy decision, as discussed in Section 2.5.1, is estimated with the expected repair quote. The actual quality of the decision can be determined from the end quote. The planners state that these end quotes are not used very often since these are available a long time after. Nevertheless, the end quote indicates the performance and can be taken into account during the next decisions as feedback.

Because decision-making is challenging to determine currently, it is also challenging to compare the different interventions. This benefits in finding the optimal solution in intervening. In this case, the optimal solution providing high service levels towards the customer, subject to the financial impact caused by the intervention, is minimal.

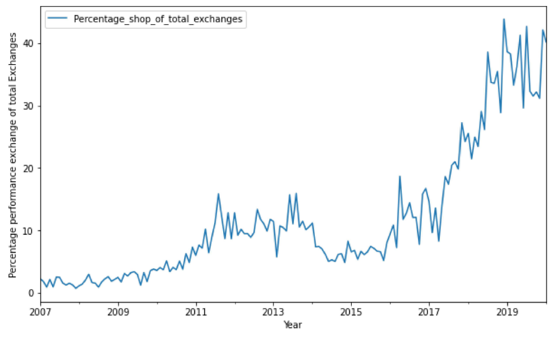
2.6.2 Impact of Decision-making and KPIs on System Level

The decision concerning the intervention has consequences in both the short- and long-term, applying to both availability and costs. The OPPs indicate in the interview that all moving parts in the CMA-loop are taken into account while making a decision. The thought behind this is to ensure availability for all components regardless of the decision that is made now. While determining the availability of components, it is already difficult to predict the long-term impact. However, for the costs, it is even a more significant challenge. Because what seems to be, financially, the best decision now can be a wrong decision in the long-term. For example, in a scenario where the OPPs have to decide to repair a landing gear or a heat exchanger, given a limited budget. Assuming the heat exchanger's repair in this situation is four times as expensive as the landing gear, repairing the landing gear is the cheaper option in the short-term. However, if the landing gear's demand rate is only once a year, and the heat exchanger has a demand rate of once a month, repairing the landing gear is not the best option. Given the consideration of the long-term availability, the financial considerations should support the decision. Combining these two factors leads to the optimal solution.

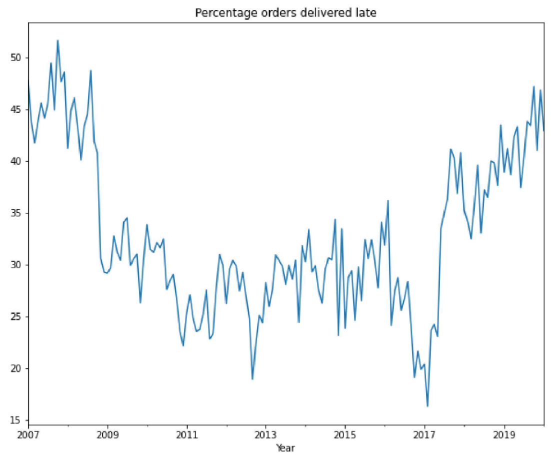
2.6.3 Performance Visualized

The CMA-program started in 1995 and has grown a lot since. In Figure 2.4 we visualized the Turnaround Time performance and the direct backorders over the past 13 years. From Figure 2.4a we conclude that the number of performance exchanges in the past 13 years compared to the total number of exchanges increased a lot. The percentage of performance exchanges is currently around 40%, which indicates that in almost 40% of the time, the repair shops cannot meet the agreed TAT. The IAC must react to this by returning components from the spare part pool to the customer.

Besides the performance exchange, Figure 2.4b shows us the percentage of orders that is arrived late at the customer. The OPPs indicate that the IAC considers an order delivered when the component leaves the spare part pool or the repair shop. Therefore, each late delivery situation is considered as a backorder. In the figure, it can be seen that the performance increased a lot. Nevertheless, the performance of orders delivered late over the past 13 years lies around 35%.



(a)



(b)

Figure 2.4: The percentage of exchanges (a) and orders arriving late (b) over the past 13 years

2.6.4 Potential Performance

The potential performance of the decision-making was discussed as well during the interview. This part of the questionnaire includes the use of computer models or algorithms. The OPPs indicate in the interview that, despite decisions being made well in their conscience, the use of a decision support system can be helpful. Some decisions are more complicated to make. Therefore, the OPP puts the problem temporarily aside to think about it for a while. A computer model can help in making a faster and substantiated decision. However, both OPPs indicate that they cannot accurately imagine how artificial intelligence can include all learning process exceptions. Therefore learning from external feedback is essential to include in an algorithm for this purpose. As the system shows that it can make logical decisions and shows potential in the learning process, confidence grows likewise.

2.7 Content of Data

For the *Proof of Concept*, the IAC provided an MS Excel overview of the data, which contains 560 different part numbers. We know the annual removal rates, transaction data, turnover rates, and market value data for each of these components. This section briefly explains the data's content and how the data set is useful for our problem instance.

2.7.1 Key Findings in Data

The datasheets contain all information from 2006 until 2020. Nevertheless, we will only use the data of all components until 2019. We do this for two reasons; we cut off at the end of 2019 such we can plan within a "known" model, and due to COVID-19, the year 2020 is not entirely representative for learning and testing purposes. Within the *Proof of Concept* we start with a single item approach. For the sake of confidentiality and simplicity, we will refer to this component from now on as *PN1*. Next, costs, demand and turnover rates, and repair probabilities are explained.

Acquisition Costs

The first data that we will discuss is the market values. Here, prices of the components are given, based on the current state. I.e., for each component, prices are given for a, e.g., serviceable, overhaul, and new condition. Additional to the state differentiation in price, there is also a price differentiation. For each given state of the component, the fair market value (*Fmv*), highest price in the last quarter *(HighFmv)*, lowest price last quarter *(LowFmv)*, maximum price all-time *(Max)*, minimum price all-time *(Min)*, and the average payed price *(Mean)* are given. These prices are based on different price points in the past.

For *PN1* we find two conditions on the market, namely serviceable and overhaul components. The costs for the serviceable components are slightly cheaper than the costs of the overhaul components. The difference is approximately 10% on the fair market value. The algorithm in Appendix E.1 gives the code for filling the cost table. From this cost table, the program can randomly pick an available price. Prices for *PN1* are denoted in Table 2.1.

Table 2.1: Price information for component PN1 ($)

Condition	Fmv	HighFmv	LowFmv	Max	Mean	Min	Recent Date
Serviceable	*This information is removed from the public*						
Overhaul	*version of this Master's thesis*						

Intervention Costs

Earlier in this chapter, we discussed multiple different interventions. In our *Proof of Concept* we will only use three of these interventions: *do nothing*, *expedite*, and *external sourcing*. Recall that all interventions can be combined. For the first intervention, *do nothing*, costs are zero since we follow the regular replenishment cycle and nothing changes in the regular transhipments.

The costs of expediting repairs bring some uncertainty. The uncertainty of costs is because not every time an OPP expedites a repair, there are costs. It is easy for the repair shop to prioritize a repair in some cases without making extra costs. However, on the other side, it is also possible that overtime is necessary for finishing a repair (man-hour costs). To realistically simulate the uncertainty in expedite costs, we will randomly select an expedite fee in the interval $[0, XXX]$. The upper bound for this interval is given by the IAC. Maximum costs \$XXX is assumable because it is three times the fixed man-hour cost.

Third, we will discuss the costs of external sourcing or buying components from other vendors. Based on the prices, given in Table 2.1, we determine the external sourcing costs. Because of market fluctuation, prices can be anywhere in the given range. Therefore, we will randomly sample from the given prices of the conditions serviceable and overhaul. Although other conditions are possible, the price of a new component usually is so high that it is not interesting. Furthermore, the market for used components is getting saturated. Therefore, we assume there always is a secondhand component available.

Repair Probability

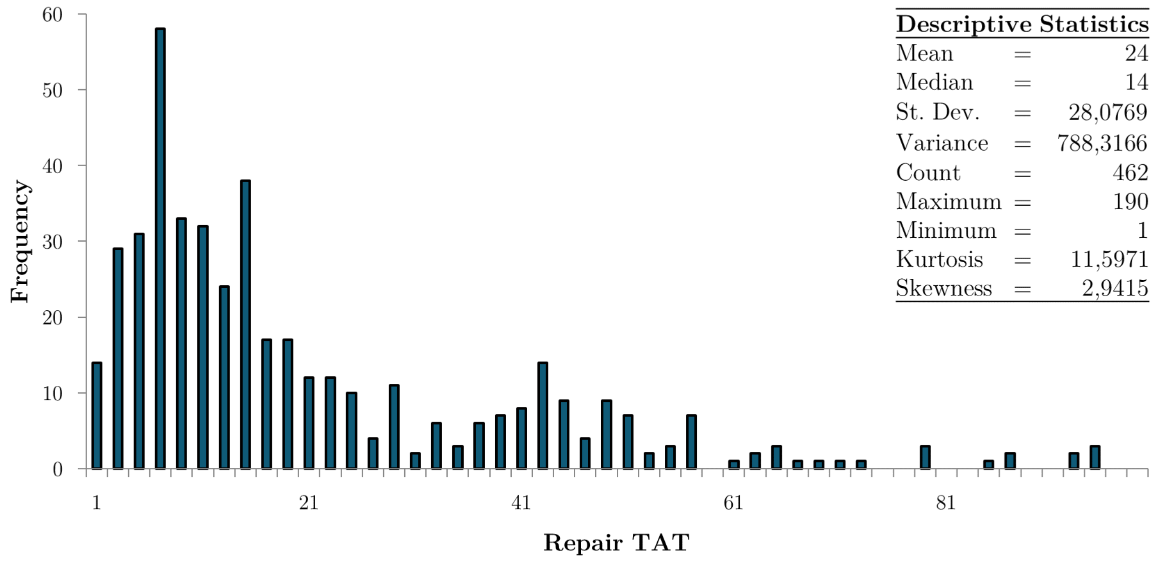
In the data set, we find the repair shop turnaround time in days. The data is presented as discrete points in time because the IAC is only interested in the returning day. For the model to determine how many components will return during a day, we will first have to define the probability function of the repairs' lead time. During the lead time of the repairs, the probability of finishing a repair on a particular day increases each day. We can determine the probability of a repair returning within the interval until that day from the calculated distribution.

To predict the repair TAT's behavior, we fit a probability distribution over the repair TATs. In this way, it is possible to predict the return date of the repair shop's component. Statistical analysis is performed on the available repair shop turnaround times to determine the correct probability distribution. The summary statistics are given in the table included in the histogram. Figure 2.5 represents the distribution of the return turnaround times. The corrupt data, caused by administrative errors, are given in the data set with repair TAT of -1 or 0, although a repair TAT of 0 is possible. However, because the likelihood of a repair TAT of 0 days is minimal, we excluded this from the observed data. Note that the figure only visualizes the frequency until 100 days. In Appendix B the full histogram is given.

Based on a probability plot analysis and a *goodness-of-fit test* for a Normal- and Lognormal distribution, we conclude that the repair turnaround times follow a Lognormal distribution. All steps of the statistical analysis are presented in Appendix B. Additionally, the Python pseudo-code, corresponding to the preprocessing of the repair data, is presented in Appendix E.3.

Now that the repairs' probability function is established, we can calculate the probabilities of repairs returning. As found so far, the cumulative Lognormal distribution gives us the probability that the component returns on the current day, or earlier: $\mathbb{P}(X \leq t)$. To define the probability that a repair will return on a day given that the component did not return yet, we use the conditional probability, where:

- R_1, R_2, R_3: The component returns before (1), during (2) or after (3) day t,
- Q_1, Q_2, Q_3: The period before (1), during (2) or after (3) day t is known.



Descriptive Statistics		
Mean	=	24
Median	=	14
St. Dev.	=	28,0769
Variance	=	788,3166
Count	=	462
Maximum	=	190
Minimum	=	1
Kurtosis	=	11,5971
Skewness	=	2,9415

Figure 2.5: Histogram of return turnaround times in days, cut off at 100 days

Based on *Bayes' theorem* we can determine the probability that the component returns during day t (R_2), given the fact we know the component did not return in Q_1. The mathematical expression is given in Equation 2.1.

$$\mathbb{P}(R_2|Q_1) = \frac{\mathbb{P}(Q_1|R_2)\mathbb{P}(R_2)}{\mathbb{P}(Q_1|R_1)\mathbb{P}(R_1) + \mathbb{P}(Q_1|R_2)\mathbb{P}(R_2) + \mathbb{P}(Q_1|R_3)\mathbb{P}(R_3)} \tag{2.1}$$

Besides the already explained regular return of repairs, we can also express the expedited return of repairs. Because there is no direct data available on the probabilities of expediting a repair's success rate, we approximate the probability of success as the union of events A (regular return) and B (expedited return) with equal probabilities:

$$\mathbb{P}(A \cup B) = \mathbb{P}(A) + \mathbb{P}(B) - \mathbb{P}(A \cap B)$$

Determining the Inventory Position

With the data set, we define the base stock level, from now on referred to as tactical level. As mentioned in Section 2.4, we determine the tactical levels following a Poisson distribution such that the service level is approximately 95%. Next, we use the inverse Poisson function to calculate the corresponding level, with the average demand during the lead time $\lambda = DrD{*}TAT$. The elaboration of the Python code is presented in Appendix E.3. Since the Poisson distribution is a discrete distribution, the returned level is rounded down to give the system a more realistic and more challenging scenario. With parameters $p = 0.95$ and $\lambda = 0.054 * 22$, we find a tactical level of two components.

Demand and Turnover Rates

We need a discrete distribution for the demand data, with non-negative IID random variables. Therefore, we use the Poisson distribution. The Poisson distribution is characterized as a distribution that is applicable for determining the number of events that occur in an interval of time when the events occur, or several items demanded from an inventory at a constant rate, λ (Law, 2015). From the available data set, we calculate the average annual demand $\lambda_{year} = 20.429$. From this, we conclude that the daily demand rate $\lambda_{day} = 0.056$. We can predict the demand probabilities upfront by determining the demand during the lead time L. The probability equation applied gives for the first day ($L = 1$) a probability of 0.9456 that no demand ($x = 0$) occurs, and a probability of 0.212 that there have been precisely a demand of one ($x = 1$) within five days ($L = 5$).

Comparable with the demand rate, the turnover data is given for all years from 2006 until 2019. Over this period, we calculate the average turnovers per year. We can calculate an average turnover of 19.357 per year. In other words, the part number moves on average 19.357 times per year. To define the turnover rate per year for one component, we divide the total turnovers by the total inventory position. Finally, to calculate the expected period before a movement occurs, we divide one by the turnover for one component. In this way, the rate is expressed in years.

In our case, the average turnovers that occur during a year is 19.357. In the previous section, we calculated the inventory position to be three components. Therefore, we can say that each component moves on average 6.452 times per year. The expected turnover rate for component *PN1* is approximately 0.155 year. Contrary to the demand rate, the turnover rate is a constant that can be used directly without input for a probability distribution. The pseudo-code of the turnover rate and demand rate are given in Appendix E.2 and E.3, respectively.

2.7.2 In Summary

The data provided by the company consists of acquisition, demand, transaction, and turnover data. This data is used to implement the model, code provided in Appendix E. First, the acquisition data consists of a table with different costs in different conditions. The Fair Market Value (Fmv) gives a good representation of the current market value. Therefore, we calculate the average Fmv to define the acquisition costs for penalties.

Second, the intervention costs can be expressed for all three interventions. The do nothing intervention gives us no extra costs, the expediting intervention gives us a cost within the interval $[0, 450]$, and external sourcing gives us a price randomly selected from Table 2.1.

Next, to determine the probabilities of a repair returning to the inventory pool, we fitted a distribution on the repair turnaround times first. Based on a Kolmogorov-Smirnov Test, we conclude not to reject the null hypothesis, assuming that the repair probabilities follow a Lognormal distribution. With the given distribution and Bayes' theorem, we calculate the probability of a repair returning at a given day.

Fourth, we calculate the tactical level. The tactical inventory level is determined by calculating the inverse Poisson distribution with parameters $p = 0.95$ and $\lambda = 0.054 * 22$. This returns a value of 2 components.

Finally, we discussed the demand and turnover rates. We conclude that the demand follows a Poisson distribution with $\lambda_{day} = 0.054$. By integrating the lead time in the probability function, we can predict the demand during the planning period. The turnover data gives the average number of movements during a year for all components combined. By dividing all components' movements by the number of components, we calculate the average movements per component per year. From this, we derive a turnover rate of 0.155 year.

2.8 Conclusion

This chapter describes the supply chain network, the current working method, and the planning methods, based on the interviews with the Operational Planning Professionals (OPP). Finally, the content of the IAC's data is presented.

Current Situation The CMA-programs work in a multi-echelon supply chain network. In this network, the actions of the OPPs originate by the alert generation. An alert is generated when the system expects that interference of the OPPs is necessary to prevent backorders soon. This can be be both proactive or reactive. There are multiple different scenarios for which an alert occurs. Besides the alerts, the OPPs receive a priority list given by a business ruling. Based on the alerts, the OPP decides which intervention suits best for the given alert. To determine the best option, both OPPs indicate that they use gut feeling or experience in making decisions. There are no tools, instruments, or methods included in making the best possible decision. Therefore, we can conclude that the motivation for decision-making thrives on implicit knowledge.

Tacit Knowledge The reason why the OPPs are making decisions without any tools, instruments, or methods is because currently there are no supporting models that support operational decisions while taking the long-run (tactical planning) into account for both financial and service level agreement field. These two indicators are representative of the performance of the CMA-program. Contradicting is that the OPPs both prioritize another KPI as motivation. Besides the difficulty in making decisions for the OPPs, it is also complicated to measure the programs' current performance without a supporting model. Therefore, the IAC benefits a lot from a decision support system, which connects the short-term decisions on the system level. Furthermore, the OPPs indicate that a planning and learning algorithm, such as reinforcement learning, can contribute to this process as long as it includes feedback from the outside and shows learning potential.

Content of Data From the data set provided by the IAC, we obtain the different values for the acquisition, intervention, and penalty costs, the repair probabilities, the stock levels, and the demand and turnover rates. The Python-code, corresponding to importing the MS Excel file data, is presented in Appendix E.

3 | Literature Review

In this chapter, we provide an answer to the second and third sub-question. First, we will briefly describe the structure and different interventions within a service control tower (Section 3.1). Subsequently, we compare the IAC's current interventions with the literature to determine possible additions (Section 3.2). Next, we review different decision support systems from literature to support our decision for reinforcement learning. Additionally, we will review the challenges and possibilities of planning and learning within our scope (Section 3.3). To better understand the positioning, concept, and elaboration of reinforcement learning, we give a clear review within the next section (Section 3.4). Finally, we will present different solving methods applicable to our problem statement supported by literature (Section 3.5).

3.1 Service Control Tower

The current economy pushes companies towards excellence, where strong supply chains are vital. This excellence implies aiming for balance, increasing demand visibility, and isolated high costs. To create real-time visibility, companies use the Supply Chain Control Tower (Blanchard, 2011). The supply chain control tower is a central hub that monitors, manages, and controls supply chain data to advise for taking action or making an intervention. It uses real-time data, which supports short- and long-term decision-making. Besides, they provide real-time visibility and collaborative information sharing. These control towers' capabilities provide the opportunity to closely monitor all service developments of the supply chain (Accenture, 2015; Deloitte, 2019). It is a system that executes corrective and preventive action in real-time, taking the resource, contention, and deviation constraints into account (Trzuskawska-Grzesińska, 2017). Because of the service purposes, it is also called a Service Control Tower (SCT).

Creating visibility into the supply chain can create a competitive advantage. However, there are also multiple challenges in the design, development, and implementation phase of the SCTs (Yan, Tan, Koh, Tan, & Zhang, 2012). Companies use the SCT as a potential source of revenue. Integrating the entire supply chain into the SCT helps live up to the standards (Trzuskawska-Grzesińska, 2017).

3.1.1 Layers of Service Control Tower

The SCT consists of five different layers (Shou-Wen, Ying, & Yang-Hua, 2013). These layers build up the control tower, bottom-up:

- **Supply chain business layer:** This layer is located at the bottom of the SCT. It includes, e.g., the manufacturers and service providers and is mainly focusing on transportation, warehousing, and information services.
- **Information perception layer:** The second layer uses real-time data sensing to collect the information in the control tower. This sensing comes from three groups; Application, Internet, and Perception.

- **Information operation control layer:** This layer can be divided into two layers; the *Information storage*, and the *Information control.* This layer contains real-time information about the supply chain. "*It is an information pool to make all information collected together.*". These two parts together have the core control function of the SCT.
- **Information service platform layer:** This is the layer that dynamically stores and updates all information about the visualization and quality. Furthermore, it can be used for real-time monitoring and support. Hence, this layer can be divided into an application-, service-, and environment layer.
- **Information manpower layer:** Finally, the top layer of the SCTs. This is also called the decision-making center. This part is monitoring the quality of supply-chain products. This can provide management control to three different levels; strategic, tactical, and operational.

The scope of this research will focus mainly on operational decision-making. From Shou-Wen et al. (2013) and Blanchard (2011) can be concluded that operational decision-management focuses on transportation management, inventory tracking and exception management on a day-to-day basis. In other literature can be found that the operational level decisions occur on item-level. Operational decisions influence short-term performance, are based on real-time information from the SCT, and cannot influence the strategic and tactical decisions in short-term (Topan et al., 2020). Following Zijm (2000) the operational decisions in material coordination are focused on the purchase and procurement management in a company, while the tactical decisions focus on the inventory management and materials planning.

3.2 Interventions on Operational Level

As mentioned, the SCT provides alerts for intervening in the supply chain. These alerts are targeting both tactical- and operational-decisions. The operational interventions influence all three levels of management control. On the operational level, this includes transportation, inventory tracking, and exception management (van Doesburg, 2011). In this section, we will define the operational interventions given by the literature. We compare the reviewed interventions with the interventions in use by the IAC to find possible additions.

3.2.1 Comparison of Interventions

Topan et al. (2020) did an extensive research on the interventions, also called event management, within spare parts. Since no other interventions were found in the literature, we assume these interventions to be the most important set. The paper mentions that we can categorize the interventions into five groups. These five different interventions can be categorized into two classes; *reactive* and *proactive* interventions. The reactive interventions are there to speed up the recovery of a failed system. The proactive interventions take place to reduce downtime risk. The different groups following Topan et al. (2020) are Stock Reallocation, Expediting, Cannibalization, Dynamic capacity allocation and lot sizing, and Joint ordering. The mentioned five groups of interventions consist of specific aspects or situations for the group. We treat each of these groups individually to compare the literature with the interventions of the IAC. Section 2.3 describes the motives for carrying out an intervention.

Stock Reallocation

First, we review the intervention stock reallocation. This intervention can be used for balancing stock over different echelons or for backorder clearing. The paper distinguishes four different aspects of stock reallocation; *After-repair*, *Reallocation of returned components*, *Dynamic inventory rationing*, and *Reallocation of parts reserved for preventive maintenance.*

The after-repair reallocation is the main activity of the IAC. When a component is repaired, it will subsequently go towards the spare part pool such that the IAC can fulfill demand. The dynamic inventory rationing is described as the Make-or-Buy decision, described in Section 2.3.2. The OPP can choose to repair the component directly or store the component in the commercial- or quarantine warehouse. This intervention happens at the end of the CMA-cycle. Finally, there is the reallocation of preventive maintenance components. With this intervention, commercial- and quarantine warehouse are used again for components. However, in this case, the intervention at the start of the CMA-cycle.

Within the group stock reallocation is the intervention reallocation of returned components not discussed yet, and therefore not used by the IAC. Additionally, this intervention is not relevant since the parts in the CMA-program will only return if the component is malfunctioning. Therefore, there is no use in further investigating the possibilities for these interventions for the company's use. However, reallocating of returned serviceable components is a critical intervention for the IAC.

Expediting

The second intervention group is expediting. We define this group as a time-shortening intervention, which can be used in two different scenarios; in case the order *has not been started yet*, or when the order *has been started*. In case the order has not been started yet, the planning professional has several options. The planning professional can use emergency shipments (faster transportation mode to resupply local warehouses), lateral transshipments (using warehouses from the same echelon), increased capacity (e.g., outsourcing or using overtime), or non-preemptive prioritizing in repairs.

For the case that the order has already been started, the planning professional can undertake the following interventions; rerouting, expediting a delivery in the pipeline, preemptive priority prioritizing repairs (e.g., changing sequence of repair jobs), or expediting a new buy.

For the group expediting, many different interventions are possible for the IAC. First of all, emergency shipment is an option. The shipments from the lease stock from other operations are used as emergency shipments by the IAC. Another option for emergency shipments is the vendor exchange. This exchange represents a form of temporary external sourcing. For this intervention, the IAC uses other vendors temporarily for solving a backorder on short notice. The IAC uses the leasing concept, such as the CMA-program, for its own purposes.

Following the literature, an inventory pooling can be seen as a method of utilizing economics of scale (Kilpi & Vepsäläinen, 2004). Furthermore, the paper recommends that even big airlines should include an inventory pooling (make-or-buy) decision. Although it brings extra costs for leasing components, lower inventory levels are needed. Nevertheless, a vendor-exchange is a combination of economic and relational interactions, where trust is the core source for a fair price and treatment (Zahedi, Bansal, & Ische, 2010).

Besides the emergency shipments, lateral transshipments are used as well. The lateral transshipments are also discussed in Figure 2.1. Further, the IAC uses prioritizing and expediting for preemptive and non-preemptive prioritizing and increased capacity events. Finally, expediting deliveries and expediting new buys are used, mentioned as expedited shipping and external sourcing, respectively.

The IAC's option rerouting is not used since the company has a delivery guarantee of 24 hours most of the time. Within 24 hours, there is almost always a possibility to ship a component from Schiphol towards the operator. Furthermore, there are no separate routes for the operators. This intervention is, therefore, not interesting for the IAC's purposes.

Cannibalization

In case immediate replenishments are not available, cannibalization can be a helpful intervention. The IAC uses this intervention in case a component is in repair. To make the component serviceable again, the repair shop uses a piece-part from another component. The other component is not non-operating and may be unserviceable on other piece-parts. The number of possible policies is large, and therefore challenging to make a decision (Khalifa, Hottenstein, & Aggarwal, 1977). This intervention is a very often used action. As described in Section 2.3.6, cannibalization is a quick, circular, and cheap option for restoring the component.

Dynamic Capacity Allocation and Lot Sizing

This intervention determines the number of components reactively being repaired when there is finite repair capacity or proactively in a flexible capacity. The OPP dynamically allocates short-term constrained capacity to multiple product classes in multiple periods (Barut & Sridharan, 2005). The IAC uses the repair capacity allocation in Make-or-Buy decisions. However, the IAC does not use lot sizing as an intervention.

Joint Ordering

With the joint ordering intervention, the planning professional can decide to save transportation costs and order multiple components simultaneously. This intervention is performed, sometimes, by the IAC. Since the tactical planning model determines the desired stock levels in advance, the planning professionals can use a joint ordering for lowering the costs of procurement. In the case of a direct backorder, there is not enough time to negotiate about a fair price or a combination of components. For this situation, we cannot use this intervention.

Do Nothing

The final option, also supported by literature, is by not taking an intervention. In case a part is less critical, or a regular replenishment is expected to arrive soon, it might be interesting to do nothing. In the case of the IAC, it is also dependent on the customer, whether this is a viable option.

Lacking Intervention

Within the current section, we reviewed both literature's and the IAC's interventions. Based on our knowledge from Section 2.3, we conclude that the reviewed literature does not cover intervention *Interchangeable Components*. However, since interchangeability is strictly a preventive action, we can formulate this action as an inventory policy. In Appendix C.1, we discuss this intervention and the use in more depth. We recommend further research on this intervention.

3.3 Decision Processes

In this section is the gap in the literature discussed concerning the relationship between operational and tactical decision-making. Furthermore, we describe the potential of artificial intelligence, additionally with the lack of available research knowledge of artificial intelligence in operational decisions.

3.3.1 Relation Between Operational and Tactical Decisions

The literature provides multiple studies concerning operational planning, given the tactical planning or vice versa. However, the relationship or communication between operational and tactical planning has not been reviewed earlier for decision-making in spare part management. Several articles (such as Stadtler (2009) and Hu, Boylan, Chen, and Labib (2018)) provide a framework of the relationship of operational and tactical decision-making. Nevertheless, these articles describe the planning in practice and not the integration, as mentioned in (Topan et al., 2020). Additionally, the article of Topan et al. (2020) mentions that the literature on spare parts planning is scarce in integrating operational and tactical planning. Therefore, we can conclude that there is a gap in the literature concerning this topic. The unknown relationship between operational and tactical planning causes a boundary for the planning professionals since decisions in one level can have implications in another level (Tuner, 2003). We referred to that boundary earlier as the *black box*. Integrating operational and tactical planning can help clarify the relationship and communication.

3.3.2 Defining the Potential of Artificial Intelligence

Each of the earlier mentioned interventions functions as a decision for the incoming alerts since a decision is a commitment to action (Boddy & Paton, 2011). The decision process is a way to decide given a set of possible actions (Murtaugh & Gladwin, 1980; Boddy & Paton, 2011). These decisions are often made under uncertainty. Here, the level of uncertainty correlates with the difficulty of decision-making (Ra, 1991). The decision process on the operational level is highly dynamic and must quickly adapt to changes (Mätäsniemi, 2008).

At the different decision stages, each decision should be made to satisfy the given constraints (Rossi, 2013). The constraints can be random (stochastic), or properties are well known (deterministic). Since the researched process is random, we will dive into the stochastic decision processes. The stochastic decision process can also be seen as a two-goal optimization problem. This optimization problem is related to both the expected performance, as risk management (Jokinen, Konkarikoski, Pulkkinen, & Ritala, 2009). Furthermore, Jokinen et al. (2009) conclude that the Operational Decision Support System (DSS) must be developed to match the structure of a stochastic decision problem. A developed DSS creates competitive advantage and understanding in the connection between information and decisions (Cooper & Schindler, 2011).

Boddy and Paton (2011) add that there are four types of decision making-models: rational models; administrative, incremental, and intuitional models; political models; and garbage-can models. Our research will include rational models. We base these models on economic motivations, aim for preset goals, and is rational and logical in assigning values and setting preferences. An automated decision-making tool supports rational decision-making using artificial intelligence or decision support systems. These models are applicable when aiming to optimize yield or for operational control (Davenport & Harris, 2005).

In the past couple of years, multiple new decision support systems have been developed. More often, systems with an interactive function are available, making it easier for the operational planner to make decisions (Nadj, Maedche, & Schieder, 2020). We compare different literature studies to substantiate our decision for the use of artificial intelligence. In Table 3.1, the methods and key findings are listed.

Table 3.1: Comparison of different decision-making methods following the literature

#	Used Method	Type of Problem	Key Findings
1	Mixed-Integer Linear Programming (MILP) Amaro and Barbosa-Póvoa (2009)	The research includes a multi-period planning model for supply chain operational decisions on supply, production, transportation, and distribution at the actual period. Furthermore, the model considers the uncertainty on products' demand and prices.	The case study's conclusion was promising, but as stated in the article: improvements should be further explored. Due to the complex nature of the models, the performance level needs a revision. This includes a comprehensive study on, e.g., price uncertainty and longer horizons.
2	Optimal Control and Stochastic Programming Cheng et al. (2004)	A capacity planning and production-inventory control approach, to compare two different approaches of computation perspective. The authors compare scenario-based stochastic programming and simulation-based optimization. Both models handle a class of problem as a multi-objective decision process under uncertainty.	A simulation-based approach is the most efficient solving method, although the stochastic programming solutions were slightly better. The simulation can be used especially well for multiple scenario problems. Large solution spaces may take much computational power and takes much time.
3	Simulation-based optimization H. Ding et al. (2008)	The authors use a simulation-based multi-objective optimization approach. They take a random demand for the solution model, different customer behavior, different product prices, and different transportation links into account. The model processes the mentioned scenarios as discrete-events.	The simulation model used (MOGA) seems to be an extension of the current optimization approaches. However, with this method, optimality cannot be guaranteed. The current research only highlighted a simplified model. For a large scale model, no further information is available.
4	Discrete-time Markov Decision Process (MDP) García-Alvarado et al. (2015)	As far as known, the study was one of the first inventory system taking recovery and environment into consideration with a finite-horizon. The problem is formulated as a stochastic dynamic problem, with the carbon management decisions that are necessary at each time interval. Furthermore, to solve the MDP, dynamic programming is used.	The numerical example shows that the model leads to a significant reduction in carbon footprints, based on the models' decisions. Besides, the model can meet demand, minimize costs, and comply with environmental policies. However, with dynamic programming, only small instances can be solved; therefore, a genetic algorithm is provided.

Continued on next page

Table 3.1 – continued from previous page

#	Used Method	Type of Problem	Key Findings
5	Random Access Refinement Horsch and Poole (2013)	The policy is focused on decision-making under uncertainty because of all the different types of multi-stage influence diagrams. This method uses a random access ordering, while dynamic programming uses a sequential method.	The model works well for multi-stage decision problems. The decision making copes well with uncertainty. Moreover, the authors conclude that the refinement approach is closely related to Machine Learning (ML) besides a greedy approach or Q-learning.
6	Mixed-Integer Linear Programming (MILP) Do Prado and Qiao (2019)	Here, we review short-term decision-making for short-term demand response. The mathematical model tends to maximize profit over time, over different scenarios. In total 576 different scenarios are computed, and with the objective function, the weighted sum over the profit is calculated.	The model show results for short-term decision-making. Results increased compared with the initial situation. Although only applicable for the short-term.
7	Mixed-Integer Linear Programming (MILP) embedded in a dynamic procedure Galasso et al. (2008)	The dynamic planning process supports short-term planning processes. Therefore, the decision space is defined as a MILP, allowing simulating particular uncertainty and risk assessment.	The model can give managers opportunities to visualize customers' behavior and supporting decision-making. Nevertheless, this model needs to be extended for implementing multiple risk assessments, such as demand probabilities and other statistical data.
8	Mixed-Integer Linear Programming (MILP) Elmaraghy and Majety (2008)	The authors use an integration of the model for purchasing, manufacturing, and logistics. Here, a mixed-integer multi-criteria model can support operational decisions.	The model does not incorporate stochasticity in the parameter settings. Although all scenarios can be made company-specific, forecasting, and fast-moving trends are hard to incorporate in MILP models.
9	Reinforcement Learning (RL) Nanduri and Kazemzadeh (2012)	The article discusses a model that measures the economic impact and makes operational decisions supported by the impact.	The RL as proposed shows great potential following the authors; however, the computing time for the given experiments is still relatively high. Hence, further research on model optimization should be done. Besides, continuous-time modeling is proposed for further investigation.

Continued on next page

Table 3.1 – continued from previous page

#	Used Method	Type of Problem	Key Findings
10	Deep Reinforcement Learning (DRL) Vanvuchelen et al. (2020)	Deep Reinforcement Learning is used to show the applicability on an SCT, while compared to two other heuristics. It is applied to an inventory problem with a trucker. Proximal Policy Optimization is used to solve the problem.	Standard heuristics as periodic review minimum order quantity and dynamic order-up-to heuristics are outperformed by the RL algorithm. Furthermore, the article concludes that smart learning algorithms are highly promising for Control Tower settings.
11	Machine Learning with statistical implementation[1] Dargam et al. (2020)	The problem includes a water demand problem. The operational decisions concern the desalination of plants.	The used model is very applicable in realistic situations. The accuracy of the operational decisions increased a lot since working with the model. Besides, risk mitigation and system performance increased.
12	Reinforcement Learning (RL) Mes and van Heeswijk (2020)	Addresses a case study where a logistics service provider is simulated in a serious game where human- and automated planning decisions are compared. The decision-making concerns day-to-day decisions of containers that need to be assigned to barges, trucks, and trains.	The RL method outperforms students and heuristics. However, logistic professionals come very close to the RL performance. Both human expertise as algorithmic developments is necessary for continuously improving the decision-making following.

The table shows that over the years, several support systems have been reviewed that relate to decision-making. However, studies related to SCTs are limited. Algorithms, such as reinforcement learning, can provide insight for logistic networks, thereby creating a foundation for more collaborative shipping in a Physical Internet with relatively little computational power (Vanvuchelen et al., 2020; Cheng et al., 2004). Due to limited available sources, this research has further oriented itself in addition to DSSs in service control towers. From Table 3.1, we conclude that the literature uses several methods in solving such a problem, whereby various authors, such as Nanduri and Kazemzadeh (2012), refer to the application and the great potential of reinforcement learning in decision-making. Moreover, in a MILP model, as described by, e.g., Elmaraghy and Majety (2008) and Amaro and Barbosa-Póvoa (2009), it is hard to incorporate a longer planning horizon and uncertainty in the models. Although Galasso et al. (2008) created a model that can incorporate uncertainty, the model still requires data such as scenario probabilities, which are hard to determine.

In summary, an automated decision-making approach, such as an AI algorithm, is a powerful tool for optimizing yield and creating operational control. Since a decision is a commitment to action, we want to determine the best way to commit to an intervention. The optimized decision-making model helps us in committing to actions. Table 3.1 compares different decision-making solving methods for mainly supply chain and inventory problems. We conclude from the table that reinforcement learning outperforms the majority of other algorithms. Furthermore, from the comparison, we conclude that incorporating stochasticity is difficult for models such as MILP. In contrast, the abilities of reinforcement learning suit our problem well.

[1]The paper does not mention which ML algorithm is used

3.3.3 Artificial Intelligence in Operational Decision-Making for Spare Parts

Currently, the service control tower and reinforcement learning are not used together in the literature studies. Research points out that especially possible configurations and future in SCTs is in a knowledge gap (Trzuskawska-Grzesińska, 2017). As mentioned earlier, in Table 3.1, implementation of smart learning algorithms in digital control towers can support the transition towards more collaborative shipping in Physical Internet (Vanvuchelen et al., 2020). Following the literature, planning and learning algorithms show significant improvement, and potential for multi-echelon supply chain networks (Kara & Dogan, 2018; L. Wang, Zeng, Zhang, Huang, & Bao, 2006). However, L. Wang et al. (2006) states that the downside of using planning and learning algorithms is that the quantity of data used in the models is limited. Therefore, the model should not entirely replace professional judgment. Nevertheless, algorithms can make decent trade-offs between high-quality, uncensored data concerning the improvement of long-term decision-making and reducing short-term maintenance costs (Abdul-Malak, Kharoufeh, & Maillart, 2019).

3.3.4 Challenges in Planning and Learning Algorithms

In modeling and solving planning and learning algorithms, we encounter multiple challenges. First of all, a commonly discussed challenge is an intensive numerical calculation of the solution set. The literature states that when the action space is a vector, an enumeration is the only option for solving the problem (Powell, 2009; Nozhati, Sarkale, Ellingwood, K.P. Chong, & Mahmoud, 2019). Richard Bellman first mentions this problem as the *Curse of Dimensionality* (Bellman, 1957). A solution might be creating an approximate value function set (Nozhati et al., 2019) or value and policy networks (Silver et al., 2016). However, the chosen policy effects the performance of the algorithm (Powell, 2009). Often, it is not easy choosing the right policy because we can only observe the environment partially in many real-world scenarios. Therefore, the model should incorporate all historical data as well in order to determine the best action (Silver, 2015).

Additionally, for every process, the algorithm needs to learn the actions and rewards from scratch. For an Atari game or an algorithm such as AlphaGo, exploring is harmless (Silver et al., 2016). Nonetheless, in a real-world situation, this is not always possible. Although lots of reinforcement learning techniques work effectively on small problems, only a few techniques apply to large problems since it is challenging to solve arbitrary problems in the general case (Kaelbling, Littman, & Moore, 1996). For small problems, Kaelbling et al. (1996) advises creating leverage to the learning process by bias; we can solve more complex problems. *Shaping*, *imitation*, and *local reinforcement signals* are examples for including bias. Contrary, Schmidhuber (2015) states that most problems are large problems, and only a few are small. Solving these large problems might be difficult since current neural networks try to work with all nodes possible. The method offered by Schmidhuber (2015) is part of the blueprints for a universal reinforcement learning method that can solve unlimited problem depth that is time-optimal in various theoretical senses, even beyond Deep Learning methods.

Finally, besides the mentioned challenges, perhaps the best-known challenge is the exploration-exploitation dilemma. This dilemma occurs in every problem setting of reinforcement learning (Kaelbling et al., 1996; Berger-Tal, Nathan, Meron, & Saltz, 2014). The goal of the reinforcement learning algorithm is to determine the optimal solution. When the algorithm uses a pure exploration, the agents learn a lot about the whole solution space without converging towards an optimal solution (Yogeswaran & Ponnambalam, 2012; Mes, 2019). While on the contrary, with a strict exploiting policy, the algorithm will most likely get stuck in a local optimum (Mes, 2019; Berger-Tal et al., 2014). With, e.g., ϵ-greedy, Boltzmann Distribution, or OR techniques such as Simulated Annealing can be used to deal with this kind of problem (Yogeswaran &

Ponnambalam, 2012). The Boltzmann Distribution also called *softmax*, outperforms all other strategies, although it is slightly more challenging to implement than the ϵ-greedy approach (Tijsma, Drugan, & Wiering, 2016).

3.3.5 Decision Processes In Short

Currently, the service control tower and reinforcement learning are not used together in the literature studies. Research points out that especially possible configurations and future in SCTs is in a knowledge gap (Trzuskawska-Grzesińska, 2017). As mentioned earlier, in Table 3.1, implementation of smart learning algorithms in digital control towers can support the transition towards more collaborative shipping in Physical Internet (Vanvuchelen et al., 2020). Following the literature, planning and learning algorithms show significant improvement, and potential for multi-echelon supply chain networks (Kara & Dogan, 2018).

From other literature, we conclude that reinforcement learning algorithms show potential in supply chain networks, despite the limited quantity of data used in the models. Therefore, the model should not entirely replace professional judgment. For a correct implementation of planning and learning algorithms, we must consider the three significant challenges: the *Curse of Dimensionality*, solving effectiveness, and exploration-exploitation dilemma.

3.4 Reinforcement Learning

Within this section, we give a brief introduction to artificial intelligence and, in particular, to the positioning, Markov Decision Processes, Bellman equations, and learning dimensions of reinforcement learning.

3.4.1 General Introduction

Reinforcement learning is a subset of Machine Learning without requiring a human touch for learning (Hosokawa, Kato, & Nakano, 2014). Since ML can be seen as a subset of AI, we briefly explain both concepts. Additionally, we briefly explain the lowest level of AI: Deep Learning.

Artificial Intelligence (AI)

As mentioned, reinforcement learning is a subset of Machine Learning, which is a subset of artificial intelligence (Figure 3.1). Before we explain RL further, we will clarify the overarching concept of artificial intelligence first. AI is the concept of computers or machines that mimic cognitive functions associated with human behavior. Examples are learning, problem-solving, and image recognition (Russell & Norvig, 2010). Our scope of AI lies within intelligence by, e.g., making decisions and having the capacity for logic. However, giving a proper definition of AI is challenging. Because the designer programs the agent's decision-making, there are discussions

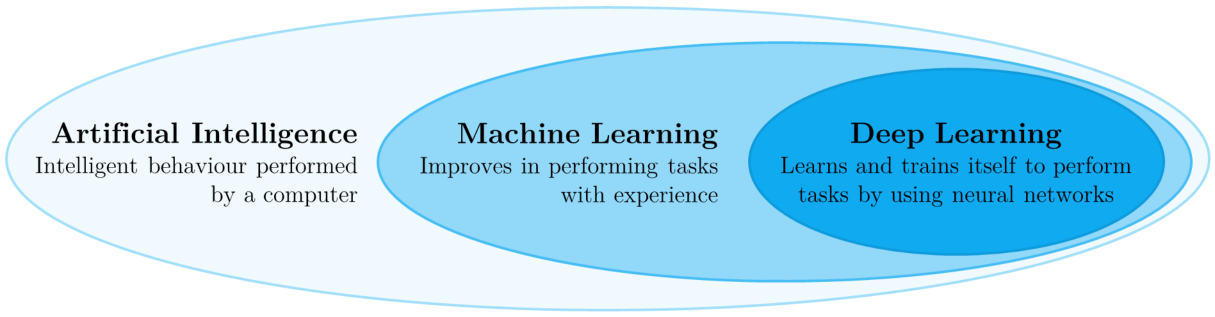

Artificial Intelligence
Intelligent behaviour performed by a computer
Machine Learning
Improves in performing tasks with experience
Deep Learning
Learns and trains itself to perform tasks by using neural networks

Figure 3.1: Positioning of AI, ML, and DL (Dhande, 2020)

about the concept of AI (Pomerol, 1997). Therefore, AI can be in everything (Bertsekas, 2019). For the sake of simplicity, the definition of AI is assumed to be intelligent behavior performed by a computer using logic, if-then rules, decision trees, and Machine Learning (Mes, 2019).

Machine Learning (ML)

Machine Learning can be seen as a subset of artificial intelligence. It is an application of AI which can learn automatically and improve from experience. ML includes statistical techniques that enable machines to improve without being explicitly programmed. The program can learn, with minimal human interaction (Faggella, 2020). There are three main branches within ML; Reinforcement Learning (RL), Supervised Learning, and Unsupervised Learning; these are visualized in Figure 3.2. Alternative branches are left out of scope.

Reinforcement Learning RL is the next level of ML. It can be defined as part of prescriptive analytics. The algorithm, or agent, is predicting what will happen and why it will happen to provide recommendations regarding actions that will take advantage of the predictions. The goal of RL is to maximize the notion of cumulative rewards by learning in taking actions. Examples are real-time decision processes, Robot Navigation, and Game AI (e.g., AlphaGo by DeepMind (2019)).

Supervised Learning This is a form of predictive analytics. The agent is task-driven by, e.g., regression classification or ranking. The agent is taught and trained using correctly labeled data, such as a problem- and solution set. Examples are weather forecasting and population growth prediction when using regression or image classification and identity fraud detection for classification problems.

Unsupervised Learning Like supervised learning, unsupervised learning is a form of predictive analytics. Since the agent does not use a solution-set or outcome, this type of agent is called data-driven. The algorithm itself will find patterns by clustering or dimensionality reduction. Examples are customer segmentation and big data visualization.

Deep Learning

The deepest layer of AI is Deep Learning. This layer can train itself to perform tasks, like speech and image recognition, by exposing multilayered neural networks to vast amounts of data.

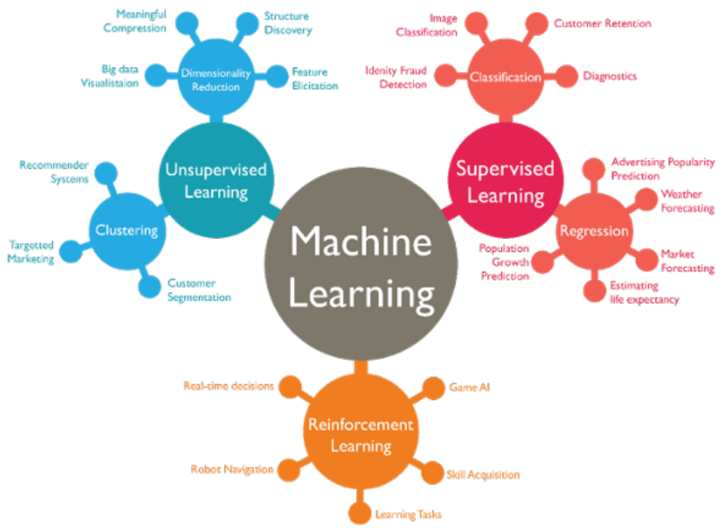


Figure 3.2: Main branches of ML (Ray, 2020)

3.4.2 Finite Markov Decision Processes in Reinforcement Learning

In reinforcement learning, a Markov Decision Process (MDP) describes the problem to resolve. An MDP is a classical formalization of sequential decision-making which consist of an agent, and an environment (Sutton & Barto, 2018; Szepesvári, 2010). We can define the agent as the decision-maker and the environment as its interaction space. The agent and environment interact continuously by selecting actions and responding to the actions with new situations. In case of a discrete-time setting, at each time step t, where $t = 0, 1, 2, \cdots, T-2, T-1, T$, the agent sees a representation of the environments state, denoted as $S_t \in \mathcal{S}$ (Heidrich-Meisner, Lauer, Igel, & Riedmiller, 2007). Based on the state that the agent is in, it selects an action, $A_t \in \mathcal{A}(s)$. Due to this action, the agent receives a reward corresponding to the undertaken action going from one state to the next in the next time step, $R_{t+1} \in \mathcal{R}$. The environment will go to the next state, S_{t+1}.

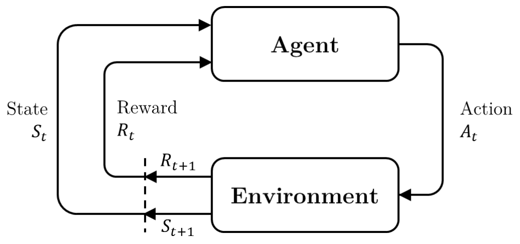


Figure 3.3: Reinforcement learning overview (Sutton & Barto, 2018)

Essential in providing theoretical guarantees about reinforcement learning algorithms is the Markov Property (Buşoniu, De Schutter, & Babuška, 2010). A process is Markov if the process describes a memoryless property, i.e., the future is independent of the past given the present (Z. Ding, Huang, Yuan, & Dong, 2020; Silver, 2015). We can mathematically express this as:

$$\mathbb{P}(S_{t+1}|S_t) = \mathbb{P}(S_{t+1}|S_1 \cdots S_t)$$

The equation tells us that the next state, given the current state, is equal to the next state given the complete history. Therefore, once a state is known, all history can be thrown away. The current state is, in this case, sufficient statistically for the future. This property holds for every time step t and for the complete state-space (Z. Ding et al., 2020).

The environment determines the next state with a certain probability. This probability is dependent on the preceding state and action. These probabilities are only applicable for a random Markov process, where the next state is determined by probability (Silver, 2015; Sutton & Barto, 2018). The state transition probabilities can be defined in a matrix $\mathcal{P}$.

$$\mathcal{P}^a_{ss'} \doteq \mathbb{P}(S_t = s'|S_{t-1} = s, A_{t-1} = a) \tag{3.1}$$

Since the MDP holds for a finite decision process, we can say that:

$$\sum_{S_{t+1} \in \mathcal{S}} p(S_{t+1}|S_t, A_t) = 1, \forall S_t \in \mathcal{S}, A_t \in \mathcal{A}(s) \tag{3.2}$$

Additional to the Markov Process, discussed so far, a reinforcement learning algorithm includes rewards. The Markov Reward Process is a Markov Chain with values. The return can be defined as the return, G_t, is the total discounted reward for a time-step t, where T represents the terminating state (including the possibility that $T = \infty$) (Silver, 2015).

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots = \sum_{k=t+1}^{T} \gamma^{k-t-1} R_k \tag{3.3}$$

In this formula, γ functions as a discount factor. For γ holds $\gamma \in [0, 1]$. The discount factor is a parameter that helps to determine the present value for future rewards, also known as cost-to-go function. Therefore, a reward received now is worth more than a reward k time steps away (Sutton & Barto, 2018). Furthermore, discounting the reward function prevents the MDP from infinite returns in cyclic Markov processes in case $T = \infty$. It is mathematically convenient to discount, and uncertainties about the future are mitigated (Silver, 2015).

The objective function for an MDP is to choose a function, $v(s)$, that typically maximizes the expected discounted sum over an infinite horizon given a specific state. We can decompose the value function into two parts; the immediate reward and discounted value of successor states.

3.4.3 Bellman's equation

To determine which policy is optimal, we use value functions. Bellman's optimality equation breaks the optimization problem into a sequence of simpler sub-problems, following Bellman's *principle of optimality* (Kirk, 2004; Bertsekas, 2017). We can determine the optimal expected sum by finding a suiting policy. Deterministic policies, $\pi : \mathcal{S} \to \mathcal{A}$, represent an action that at any time $t \geq 0$ the actions $\pi(S_t)$ is selected in state S_t (Heidrich-Meisner et al., 2007; Szepesvári, 2010). There are state value functions, $v(S_t)$ and action value functions, $q(S_t, A_t)$, in reinforcement learning. The state value function can be defined as the state's value while following a policy (Sutton & Barto, 2018). It represents the expected return when starting in state S_t and acting following policy π. Subsequently, we can decompose the state-value function, $v_\pi(S_t)$ into immediate reward plus a discounted value of the successor state as given in Equation 3.4.

$$v(s) = \mathbb{E}\left[R_{t+1} + \gamma v(S_{t+1}) | S_t = s\right] \tag{3.4}$$

The second value function is the action value function. This value function gives the value of an action in some state when following a policy. I.e., the expected reward, given an action in the given state of the agent while following a policy. The action-value function can be decomposed equally to the state-value function in two parts; immediate reward and the discounted value of successor states:

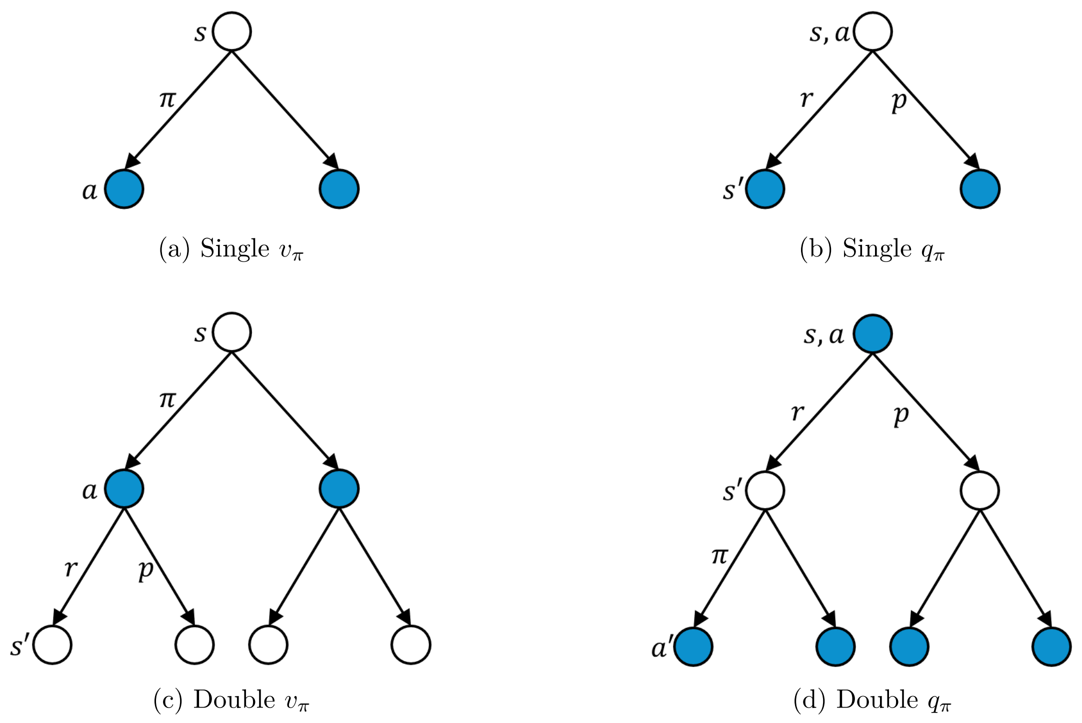
$$q_\pi(S_t, A_t) = \mathbb{E}_\pi\left[R_{t+1} + \gamma q_\pi(S_{t+1}, A_{t+1}) | S_t = s, A_t = a\right] \tag{3.5}$$

Now, we can determine determine the quality of a specific state (Figure 3.4a) or the quality of an action, bringing us to a next state (Figure 3.4b). When both diagrams are combined, we end up in a recursion where v and q are expressed in terms of itself, Figure 3.4c and 3.4d. In this situation, we can solve the Markov Decision Process by performing a two-step look-ahead.

Problem owners often do not care about a random policy, which follows actions by change. The goal of the MDP is to find the best path through the system, i.e., to find the optimal way to solve the problem. We use the optimal value function to solve the problem. From, e.g., Sutton and Barto (2018), we can derive the optimal value functions. The optimal value functions are recursively related by the Bellman optimality equations. As visualized in Figure 3.4 the optimality equations can be derived identically (Sutton & Barto, 2018; Szepesvári, 2010; Bellman, 1957). Leading to the Bellman's Optimality Equations for v_* and q_* in Equation 3.6.

$$v_*(S_t) = \max_a \mathcal{R}_s^a| + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a v_*(S_{t+1}) \tag{3.6a}$$

$$q_*(S_t, A_t) = \mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a \max_{a'} q_*(S_{t+1}, A_{t+1}) \tag{3.6b}$$



(a) Single v_π (b) Single q_π

(c) Double v_π (d) Double q_π

Figure 3.4: Back-up diagrams for Bellman Expectation Equations (Sutton & Barto, 2018)

3.4.4 Extensions to the Markov Decision Processes

To create a good overview of all possibilities, branches, and extensions of the Markov Decision, we explore the average reward value function and the partially observable Markov Decision Problem. These extensions are very commonly used for reinforcement learning purposes. However, since these extensions do not apply to our content, both algorithms are briefly explained in Appendix C.2.

3.4.5 Learning Dimensions of RL

To determine which algorithm we use to solve the problem, we need to identify the learning dimensions of RL. The dimensions reviewed below help us choosing an applicable algorithm for our problem.

Model-Free or Model-Based

The first option that we will discuss is the model-based and model-free approach. In reinforcement learning, a model is defined as the ensemble of acquired, environmental knowledge (Zhang & Yu, 2020). For example, in an MDP, the transition model and reward function together define the model (Sutton & Barto, 2018). Model-free RL is applicable for situations where the agent does not know the model or where the decision trees are too complex to evaluate (Huys, Cruickshank, & Seriès, 2014). With model-free RL, the algorithm learns by updating every iteration to predict or estimate the best value. By sampling from the world, an approximated expectation can be found for this application. In the case of model-based RL, the transitions and rewards are expressed in terms of the Bellman equations (3.6). The models can work with a given model or learn a model. If the model is already known, the agent can use the model without interaction with the environment.

On-policy or Off-policy

Next, we look at on- or off-policy models. As described earlier, an algorithm can follow a particular policy to determine what actions the agent takes to get from state s to s'. If the agent can learn the policy values followed so far, including the exploration steps, we call this on-policy (Mes, 2019). Also, on-policy algorithms try to improve the policy values that make decisions (Sutton & Barto, 2018). To get here, the agent must interact with the environment (Zhang & Yu, 2020).

The other option is an off-policy algorithm. For off-policy algorithms, the agent does not use the policy's actions to determine the action values. Furthermore, Sutton and Barto (2018) say that an off-policy evaluates or improves a different policy from what is known to generate data. Finally, the agent does not have to interact with the environment himself. Experiencing other agents who interact can already significantly improve the policy (Zhang & Yu, 2020).

Real-world or Simulator

Real-world (online) models are addressing future state problems by presenting the data sequentially instead of knowing all data on forehand. Often, online learning algorithms tend to minimize regret (Z. Ding et al., 2020). Dynamics of the real world are taken into account with the online algorithms (Croonenborghs, Ramon, Blockeel, & Bruynooghe, 2007).

In contrast, offline learning acts like a simulator (Mes, 2019). The offline learning methods are more about static data sets, and the aim is to learn the modified dynamics through collected training data. If a neural network acts the same as modified dynamics through offline training, online learning methods are not necessary (Jin, Pipe, & Winfield, 1997).

Passive or Active Learning

In a passive learning environment, the agent's policy is already given. The policy is fixed, so the agent is told what to do (Russell & Norvig, 2010). With passive learning algorithms, the agents learn the expected utility $U_{\pi}(s)$. The utility can be calculated by *direct utility estimation*, *adaptive dynamic programming*, and *temporal difference learning* (Russell & Norvig, 2010; Sutton & Barto, 2018).

Contrary, with active learning, the agent needs to decide what to do. There are no fixed policies, and therefore the goal is to determine the optimal policy (Mes, 2019). For active learning, lots of algorithms are known as well: *Approximate Dynamic Programming with exploration*, and *Q-learning* is most common in active learning (Russell & Norvig, 2010).

3.5 Solving Methods for a Reinforcement Learning Model

In this section, we will determine the power of different reinforcement learning models for our purpose. To define the applicable algorithms, we used, e.g., Sutton and Barto (2018), Szepesvári (2010), and Z. Ding et al. (2020). This section explains for six algorithms why these are functional, what the differences are between the algorithms, and what possible challenges are for the algorithm in our purpose. Because we are only using one of the algorithms for our experimental design, we will briefly explain the algorithms. In Appendix C.3, a more detailed explanation of the different algorithms is given.

In Section 3.4.5, we discussed the difference between model-free and model-based algorithms. For the IAC's application, a model-free algorithm is preferred because it generates a real experience and is not dependent on a given model, which might change over time. Nevertheless, because the model-free approach does not include the rich history of data and is very time consuming, we will also review a model-based algorithm (Approximate Dynamic Programming) and a *hybrid* (Dyna-Q).

3.5.1 Approximate Dynamic Programming

An often used algorithm is Approximate Dynamic Programming (ADP). This algorithm is suitable for problems with a high dimensionality (Mes & Perez Rivera, 2017). ADP is sequential decision-making under uncertainty, where the updating function ensures that not a full enumeration is necessary, as is with the regular Dynamic Programming. Since problem spaces can grow far beyond the computational power, the ADP algorithm combined with Operations Research, can help overcome this problem (Powell, 2010). For the smaller problem instances, which are in our interest, an ADP can already be beneficial in the learning process of making better decisions over time (Powell, 2009).

Besides routing problems, storage and inventory pooling problems are solved with ADP (Mes & Perez Rivera, 2017; Nascimento & Powell, 2013; Simao & Powell, 2009). The paper of Simao and Powell (2009) works on an aircraft manufacturer who responds to random demand. The manufacturer repairs malfunctioning components, which can re-enter the system after completion of the repair. The paper presents the problem of determining the inventory levels at each warehouse. Since this research shows similarities with our problem statement, we will take the ADP into account as a possible algorithm. Appendix C.3.1 explains the exact algorithm steps for this solution approach.

Implementing the Approximate Dynamic Programming algorithm brings some challenges. The first challenge that we will address is the exploration vs. exploitation dilemma. Since we discussed this challenge before (Section 3.3.4), we will not go much in-depth again. To avoid getting stuck in a local optimum, the agent must make a trade-off between exploring and exploiting (Powell, 2009). Additionally, to this challenge, the decision-maker has to consider to make the algorithm on-policy or off-policy, i.e., does the agent needs to update the value functions using the results of the exploration (on) or do we perform an off-policy control (Mes, 2019). For example, (Ryzhov & Powell, 2011) tends to resolve the dilemma using a Bayesian Active Learning approach.

Another challenge is the step size. While updating the VFA, an α must be determined. The α, or step size, determines how important each new update is for the VFA. We can do this by, e.g., constant, variable, harmonic, polynomial, or McClain's formula for step size (Mes, 2019). It is difficult to determine which step size is best for each problem (Powell, 2009).

3.5.2 TD-Control Methods

Q-Learning is an off-policy Temporal Difference (TD) control method. TD control algorithms learn through every step that the agent takes. Since the TD algorithms learn on experience instead of through available known steps, TD algorithms are model-free. The TD methods update the estimates by *bootstrapping*, learning a guess from a guess (Sutton & Barto, 2018). Because Q-Learning is an off-policy method, it does not follow a policy to find the next action. The method chooses the next action in an ϵ-greedy manner. Q-Learning is further explained in Appendix C.3.2. Another form of TD-Learning is SARSA, which is an on-policy TD control method. SARSA aims to find the Q-values given a policy π with all state-action pairs. In general, TD control methods are applicable for finite and relatively small states, and action spaces (Szepesvári, 2010). By interacting with the environment, the agent can earn rewards and improve.

Due to the interdependence of decisions and scenarios, it may be challenging to determine which states have or had the most effect on the outcome. A weight parameter contributes partially to the prediction (Isbell, 1992). Furthermore, the TD-algorithm is designed to learn and predict an outcome. The algorithms method of providing feedback can be inaccurate (Isbell, 1992). Finally, for non-linear networks, the algorithm has shown that it may not converge towards the optimal solution (Tesauro, 1992).

3.5.3 Dyna-Q

In online planning, during the decision-making process, the interaction may change the environment due to uncertainty or randomness of the environment (Sutton & Barto, 2018; Zhang, Huang, & Zhang, 2020). Due to the change in the environment, the interaction in planning can change. The Dyna-Q algorithm is useful to learn within the interaction (Sutton & Barto, 2018). As mentioned, this algorithm is a hybrid between model-free and model-based algorithms. The advantages of both model-free and model-based algorithms are integrated into this single architecture. Within the algorithm, a Q-table is generated to instruct the agent. This table is learned and updated for each step of action in the environment (Zhang et al., 2020). With this process, the Dyna-Q includes planning, acting, model-learning, and direct reinforcement learning continuously (Sutton & Barto, 2018). The algorithm is given in Appendix C.3.3.

Dyna-Q is currently mostly used in, e.g., financial decision-making since the real and historical data can be combined. Furthermore, the algorithm is efficient in robotic manipulation and control. In robotics, Dyna-Q is used to capture the shown movements and interpret it and translate it to the several degrees-of-freedom (Skoglund, Palm, & Duckett, 2005). For the interpretation of inventory control of spare parts, we could not find any literature. However, the robotic manipulation and task-completion dialogue may fit well with the inventory problem since the algorithm communicates well outside the environment and can simultaneously speed up the learning process (Ou, Chang, & Chakraborty, 2020; Peng et al., 2018). Variations on the algorithm, such as Dyna-Q+, where an exploration bonus is included to encourage exploration, might be useful for finding a good fit (Sutton & Barto, 2018).

Nevertheless, due to the structure of the model-based algorithm, there are challenges in this algorithm. The learning performance of the model affects the policy learning results. I.e., in case the environment is non-stationary (dynamic), the algorithm does not work well (Kim, Kwon, & Baek, 2008; Zhang & Yu, 2020). Due to the uncertainty of the system dynamics, performance may be volatile in the control domain. Therefore, the design of efficient learning mechanisms might be interesting (Xu, Zhang, & Liu, 2009). Additionally, the algorithm for planning only a simple lookup table without in-depth knowledge is used. The Q-ADP algorithm of Ou et al. (2020) helps finding a near-optimal policy for improving system production (Ou et al., 2020).

3.5.4 Deep Q-Networks

In a previous subsection, we discussed the Q-Learning algorithm. Since the convergence of Q-Learning can be unstable, Deep Q-Networks DQN can be considered (Tesauro, 1992; Huang, 2020). DQN combines Q-Learning with Deep Neural Network (DNN), to address the instability issue and to create a more efficient and better performing algorithm (Mnih et al., 2013). DQN shows that the agents can achieve high performance without using a different problem-specific feature sets (Sutton & Barto, 2018). In Appendix C.3.4, we will discuss the algorithm and functionalities more in detail.

DQN algorithms are, like many other algorithms, tested on Atari video games. Besides the games, e.g., inventory optimization problems, use the algorithm. In *The Beer Game*, a multi-agent cooperative supply chain problem, the DQN algorithm can obtain near-optimal solutions and does not require knowledge of the demand probability distribution; it uses historical data (Oroojlooyjadid, Nazari, Snyder, & Takác, 2017). However, such an algorithm's computational time is relatively high since there is only one learn-able agent. Additionally to this paper, DQN can be used for predicting the spare part demand of fast-moving parts. The algorithm outperforms the current model in more than 50% of the components (Henkelmann, 2018).

The DQN algorithm has shown some great improvements compared to the Q-Learning algorithm. However, DQN shows that it suffers from substantial overestimations is certain situations (Van Hasselt, Guez, & Silver, 2015). The overestimations can be caused by the fact that the objective function Y_i contains a max operator. Because the Q-value is noisy, the optimal value can be overestimated (Huang, 2020). To overcome this problem, Van Hasselt et al. (2015) and Mnih et al. (2013) provide a Double Deep Q-Network (Double DQN), which decorrelates the noise and evaluates the networks in different stages. In the paper of Huang (2020), we can find more extensions for the DQN algorithms, which might be interesting for further research.

3.5.5 Actor-Critics

Actor-Critic (AC) is a modern approach for reinforcement learning algorithms. As explained earlier, there are two main types of RL methods: value-based and policy-based. The Actor-Critics is located on the edge of both streams (Z. Ding et al., 2020). The Actor takes as input the state and returns the best action as output. Subsequently, the Critic evaluates the action and returns feedback on how good the action was and how it should adjust (Sutton & Barto, 2018; Szepesvári, 2010). In this case, the Actor is learning the policy (policy-based), while the Critic evaluates the value function (value-based). Additionally to AC, we can include the Advantage function (A2C). The advantage function gives the value of how good an action is, compared to other actions, given a state. To make the A2C model more powerful, we can use the Asynchronous Advantage Actor-Critic (A3C). The A3C consists of multiple independent agents, each in an own parallel environment (Mnih et al., 2016). Therefore, the algorithm can operate more robust, faster, and achieve higher scores (Z. Wang et al., 2016). The algorithm for A3C is given in Appendix C.3.5

Literature points out that the use of A2C or A3C in multi-echelon networks for inventory management is able to handle the multi-objective reward (Sultana et al., 2020). The experiments' results come close to reality, where it can optimize performances in the long-run. The A3C shows in literature studies to outperform multiple heuristics and other algorithms. Not only in the Atari games the Actor-Critic performs well, but in multi-echelon inventory management with lost sales or backorders as well (Gijsbrechts, Boute, Van Mieghem, & Zhang, 2019).

Despite the promising results of the A2C or A3C algorithms, there are still some flaws. Currently, there are no specialized policies that lead the algorithm to an optimum. Therefore lots of parameter tuning is necessary to create a realistic environment (Gijsbrechts et al., 2019). Additionally, by a deterministic policy behavior, the Actor cannot learn the non-preferable policies (Szepesvári, 2010). By including stochasticity or randomness in decision-making, this is easier to learn. In a small action-space, the Critic can use, for example, an approximate action-value function. An ϵ-greedy or Boltzmann exploration approach is recommended for solving the problem. Finally, Sultana et al. (2020) point out that Actor-Critic algorithms are not further developed or tested for varying lead times in replenishments. Further research should create insight into the performance of the algorithms with this additional restriction.

3.5.6 Proximal Policy Optimization

Proximal Policy Optimization (PPO) is an algorithm within the policy gradient methods for reinforcement learning. PPO alternates between sampling data and interaction with the environment, and it is suitable for high-dimensional control problems (Heess et al., 2017). Although, the algorithm is derived form Trust Region Policy Optimization (TRPO), PPO is better scalable than TRPO, and still very data efficient, robust, and reliable (Schulman, Filip, Dhariwal, Radford, & Klimov, 2017). We will briefly explain the PPO algorithm in Appendix C.3.6. For further explanation about TRPO see (Z. Ding et al., 2020), (Heess et al., 2017), or (Schulman, Levine, Moritz, Jordan, & Abbeel, 2015).

PPO can be seen as an approximate version of TRPO, which is more convenient to apply for large-scale settings (Heess et al., 2017). For the algorithm steps of PPO, see Appendix C.3.6. The PPO algorithm has proven to be effective in optimal control for inventory problems (Meisheri, Baniwal, Sultana, Khadilkar, & Ravindran, 2020). Also, within the control tower setting, the PPO algorithm tested and proved itself to be useful (Vanvuchelen et al., 2020). Since PPO is relatively easy to implement (compared to TRPO) and it provides robust and reliable results, is PPO a promising algorithm for our purpose.

Currently, in the literature, multiple flaws are found concerning PPO. Although it is data-efficient and more comfortable to implement than TRPO, the performance of PPO can be unstable (Y. Wang, He, & Tan, 2019). The performance is dependent on the effectiveness of the exploratory policy search and can be caused by lousy initialization (Y. Wang, He, Tan, & Gan, 2019). Appendix C.3.6 presents several methods to overcome these flaws.

3.6 Conclusion

Very little information concerning reinforcement learning in a service control tower setting for spare parts is known. Therefore, we tried to create a better overview of the different aspects following the existing literature. This chapter creates insight into interventions and decision support, basic reinforcement learning concepts, and several different reinforcement learning algorithms.

Interventions and Decision Process A service control tower is a central hub which monitors, manages, and controls supply chain data (Accenture, 2015; Deloitte, 2019). The SCT creates insight into the streams and can therefore create a competitive advantage. Furthermore, it generates alerts, to which the company can respond. This process influences transportation, inventory tracking, and exception management (van Doesburg, 2011). We reviewed the most common interventions stated in the literature and compared these with the interventions currently performed by the IAC (Topan et al., 2020). Here, we found out that currently, one intervention that the IAC uses is not stated before in the literature as a backorder prevention cause. The intervention, interchangeable components, is commonly used in manufacturing and pharmaceutical industries. Although the IAC uses it as an intervention for backorder prevention, we can use it as a preventive inventory policy.

Further, in this chapter, we compared 12 papers concerning the potential of artificial intelligence in decision-making. We compared AI models with, e.g., Mixed-Integer Linear Programming (MILP) and stochastic programming methods. In decision-making, using AI, models tend to optimize yield for operational control (Davenport & Harris, 2005). From this comparison, we can conclude that planning and learning algorithms, such as reinforcement learning, have a great potential in operational decision-making for inventory and spare part problems, mentioned by, e.g., L. Wang et al. (2006) and Abdul-Malak et al. (2019).

Although there is immense potential, there are also some challenges in planning and learning algorithms. The first challenge discussed is the *Curse of Dimensionality* (Bellman, 1957). Due to the large state space, solving can be a very intensive calculation. The literature offers different solutions to overcome this problem. For example, using value and policy networks (Silver et al., 2016), or approximate value function sets (Nozhati et al., 2019). The second challenge concerns the effectiveness of the solving algorithms of reinforcement learning. Following the literature, it is difficult to solve arbitrary problems in the general case (Kaelbling et al., 1996). Including bias can help in the learning process. Larger problems are often only solvable with neural networks. With an optimal method, unlimited problem depth can be solved (Schmidhuber, 2015). The final challenge we discussed is the exploration-exploitation dilemma. Different approaches, such as ϵ-greedy, can help us to find a near-optimal solution without getting stuck in a local optimum too early (Yogeswaran & Ponnambalam, 2012).

Reinforcement Learning We can define reinforcement learning as subset of machine learning, which is again a subset of artificial intelligence. The literature describes two basic formulations of reinforcement learning problems; the Multi-Armed Bandit problem and Markov Decision Processes Framework. Because our problem is suitable for MDP, we will leave the MAB out of scope. The MDP is a classical formalization of sequential decision-making problems, which posses the Markov property. This means that the future is independent of the past given the present. An MDP is formulated as follows; at each time step t, the agent sees a representation of the environment S_t. Based on the observation, the agent determines an action A_t. With this action, the agent will receive a reward R_{t+1} and the agent moves with transition probability $\mathbb{P}(S_{t+1}|S_t, A_t)$ to a new representation of the environment S_{t+1}. The decisions over time lead to the discounted reward function denoted in Equation 3.3. The main objective of the MDP is to maximize the expected discounted reward function given a state S_t and find the corresponding actions. These value functions also known as the Bellman equations, consist of a state- and an action-value function. Naturally, finding the optimal state-value function leads directly towards finding the optimal action-value function, and vice versa.

Solving Method Finally, we discussed six different RL methods for solving MDPs; Approximate Dynamic Programming, TD-Control, Deep Q-Networks, Dyna-Q, Actor-Critics, and Proximal Policy Optimization. From the literature, we found the applications and potential of the different algorithms. We conclude that the Dyna-Q algorithm seems most powerful for our purpose from the six algorithms, although no other literature is found in our scope. Dyna-Q works both with model-based as model-free algorithm, whereby easy implementation, strong, and quick performance show added value for our research. Additionally, Dyna-Q starts improving very fast following the literature. Besides the Dyna-Q algorithm, the A3C and PPO algorithms show excellent potential for our purpose. However, because both algorithms are a deep learning approach, and the implementation of the algorithms is far more complicated than the Dyna-Q algorithm, we will not continue this research with these algorithms. Nevertheless, because literature concerning A3C and PPO show potential for our purpose, we recommend exploring the possibilities of A3C and PPO in future research.

4 | Solution Design

This chapter describes the solution design for the Component Maintenance & Availability (CMA) problem. For the solution approach, we first formulate the decision-making framework based on the stochastic dynamic programming framework. Section 4.1 describes the framework in decisions, states, rewards, and transition probabilities. In order to test and validate the planning and learning algorithm, we create an initial situation. By designing an exact solution approach in Section 4.2, we can determine the true optimal value. Next, a we discuss a heuristic solution approach in Section 4.3, which functions as the counteragent and benchmark of the planning and learning solution approach, discussed in Section 4.4. We review the design and properties of all three models in the mentioned sections.

4.1 Stochastic Dynamic Problem Framework

So far, the problem definition has been kept fairly strict and simple, where decisions are independent of time. From this point of view, an infinite MDP would suffice for this problem. However, we know that practice does not behave this way. Therefore, we need a method that solves the optimal policy problem for each day, e.g., used by García-Alvarado et al. (2015) in Table 3.1. In this section, we present the elements of the SDP applied to our problem statement. We discuss the states (Section 4.1.1), decisions (Section 4.1.2), reward functions (Section 4.1.3), transition functions (Section 4.1.4), and transition probabilities (Section 4.1.5).

4.1.1 State

From the data analysis in Section 2.7, we know several component characteristics, such as costs and the base stock level (tactical level). Within the SDP Framework, we make a decision based on the current inventory position while minimizing the costs and backorders. Here, we use the following interpretation. We define $I_{t,n}$ as the inventory position at the end of time period $t \in T$ for component $n \in N$. Here, we assume that a time period t represents one day and the finite planning horizon is denoted as T. Since the inventory position must be an integer number, we denote the state as $I_{t,n} \in \mathbb{Z}$. The inventory position, represents the stock over the complete CMA-loop. We present the inventory position in Equation 4.1b. The inventory position includes the deterministic integer variables *current on-hand stock* ($s_{oh,t,n}$), *current stock in repair shop* ($s_{repair,t,n}$), *current stock at operator sites* ($s_{customer,t,n}$).

$$S_t = [I_{t,n}]_{\forall n \in N} \tag{4.1a}$$

s.t.

$$I_{t,n} = s_{oh,t,n} + s_{repair,t,n} + s_{customer,t,n} \tag{4.1b}$$

All variables $\in \mathbb{Z}$

4.1.2 Decision

The agent makes decisions in a finite planning horizon at each time period t for a component number n. Additionally, without loss of generality, we assume that the decisions are made for a single location model because, in reality, most transactions are performed from the central warehouse. With this assumption, there is hardly any deviation from reality. Therefore, it is not trivial to add multiple echelons or locations in decision-making. This decision represents the intervention which is carried out by the OPP. For now, we start with three different interventions: do nothing, expediting, and external sourcing. As mentioned, the interventions can be combined. Additional restrictions to decision-making are: a limit to buy a maximum of three new components (Equation 4.2b), and only components in the repair shop $s_{repair,t,n}$ can be expedited (Equation 4.2c). Due to the buying restriction, we can limit our total combined interventions to eight. Note that doing nothing means regulating inventory via regular transhipments, as explained in Section 2.3.1. The natural decision variable $x_{t,n,f}$ defines the quantity of components at time t, of component $n \in N$, in intervention $f = \{0, 1, 2\}$.

$$x_t = [x_{t,n,f}]_{\forall n \in N, f \in \mathcal{F}} \tag{4.2a}$$

s.t.

$$x_{t,n,expedite} \leq s_{repair,t,n} \tag{4.2b}$$

$$\sum_{t=1}^{T} x_{t,n,buy} \leq 3 \tag{4.2c}$$

$$x_{t,n,f} \in \mathbb{N},\ \forall t, n, f$$

At each time t, the agent can decide the number of expedited repairs and bought components from external sources. We can denote the decisions as vector x_t. Each of the decisions impacts the change in the transition of states and can be used simultaneously. E.g., the OPP can decide to expedite a repair and buy a new component from external sources to quickly and successfully absorb the incoming demand. Since there are many components available on the market that can be delivered quickly, we assume that the lead time L is equal to zero time periods. For expediting the repairs, we assume no lead time ($L = 0$) because expediting leads to a prioritized and accelerated repair in the current time period t. The agent's objective is to make decisions to minimize the total backorder-, holding- and intervention costs over the given planning horizon.

4.1.3 Reward

For this instance, the rewards are the costs that the agent makes in the decision-making. The direct reward function is given by:

$$C(S_t, x_{t,n,f}) = \begin{cases} BO(s_{oh,t,n}) + \sum_{f=0}^{F} E_{n,f} x_{t,n,f}, & \text{if } t < T, \forall n \\ BO(s_{oh,t,n}) + \sum_{f=0}^{F} E_{n,f} x_{t,n,f} + L(I_{t,n}), & \text{if } t = T, \forall n \end{cases} \tag{4.3a}$$

s.t.

$$BO(s_{oh,t,n}) = \begin{cases} 0.5 Q_n |s_{oh,t,n}| a^{g_n} u_n, & \text{if } s_{oh,t,n} < 0,\ \forall n \\ 0, & \text{otherwise} \end{cases} \tag{4.3b}$$

$$L(I_{T,n}) = \begin{cases} Q_n (I_{T,n} - I_{tactical})^2, & \text{if } (I_{T,n} > I_{tactical} | I_{T,n} \neq I_{1,n}),\ \forall n \\ 0.5 Q_n (I_{tactical} - I_{T,n})^2 u_n & \text{if } I_{T,n} < I_{tactical},\ \forall n \\ 0, & \text{otherwise} \end{cases} \tag{4.3c}$$

The cost function (Equation 4.3a) includes three parts: backorder, intervention and terminal state costs. During the planning horizon, the cost function includes only backorder and intervention costs. To incorporate the long-term, and thereby tactical planning, we include the future holding or shortage costs in the terminal time step.

The first expense we discuss is the backorder costs, as noted in Equation 4.3b. This thesis defines backorders as the situation that the on-hand inventory in period t cannot directly satisfy the demand of period t; it is below zero. The backorder cost function penalizes the agent worth half of the acquisition costs, multiplied by the turnover rate u_n, to incorporate the time between movements for the component. By including this rate, we differentiate between fast and slow movers and therefore penalize them more realistically. Additionally, the cost function includes a multiplication with an exponentially growing penalty rate a^{g_n}. For each time that a backorder occurs, the value of g_n increases by one, with a as a constant. In this way, it is less interesting to ignore backorders for a long time period. In other words, due to this growth, it becomes more and more attractive to close the backorder by buying a component.

Next, the intervention costs $E_{n,f}$ are dependent on each intervention. The intervention costs can be, e.g., costs for repair man-hours or exchange fees that the company needs to pay for external sourcing such as order costs $(O(x_{t,n,f}))$. Equation 4.4 explains the formation of the order costs. In case there are new bought components, $x_{t,n,buy} > 0$, the order costs consists of the fixed costs, K and the acquisition costs $Q(x_{t,n,buy})$. The acquisition costs are dependent on the quantity and the type of component. If $x_{t,n,buy} = 0$, the order costs are zero.

$$O(x_{t,n,buy}) = \begin{cases} K + Q(x_{t,n,buy}), & \text{if } x_{t,n,buy} > 0 \\ 0, & \text{if } x_{t,n,buy} = 0 \end{cases} \tag{4.4}$$

Finally, we explain the terminal costs, as discussed in Equation 4.3c. As the name suggests, these costs only occur in the terminal state. As earlier, this research's scope is to improve decision-making on the operational level with the tactical level as a guideline. Therefore, the terminal costs consist of two different parts: *holding costs* if the terminal inventory exceeds the tactical inventory, and *shortage costs* if the terminal inventory is less than the tactical inventory. Naturally, when no holding or backorders occur, there are no terminal state costs.

The calculation for holding costs, as shown, is based on the regular computation of the holding costs. The IAC's estimate for the holding costs per component per year is 15% of the acquisition costs per item on stock. Because 15% is on a yearly basis, the penalty for keeping stock is negligible for a short period, such as a single day within the planning horizon. Because we perform the current analysis for a closed-loop environment, where components can not quickly leave the loop, we take the holding costs for an extended period. Note that we only pay holding costs when the ending inventory differs from the tactical planning or the starting inventory. Additionally, we penalize holding costs more heavily when diverging further from the tactical planning or the starting inventory by multiplying the inventory's squared difference by the acquisition costs.

The shortage costs are calculated similarly to the backorder costs, excluding the exponential penalty rate g_n. We penalize the shortage because we calculate tactical inventory levels such that the Expected Backorder (EBO) is minimized. Therefore, not meeting the tactical inventory levels in the current planning horizon increases the probability of backorders in the next planning horizon. The difference between tactical and total inventory levels gives the direct shortage, leading to potential backorders. In case the difference increases the shortage costs, and urge in solving the shortage increase. Because the seriousness of the future backorder is dependent on the expected time for a component will move, we include the turnover rate u_n.

In this research, we aim to find the policy that minimizes the costs over the planning horizon. The policy that presents the corresponding optimal value function can be represented in the Bellman optimality equations, as denoted in Equation 4.5 and 4.6. In the first equation, the state value function is expressed in terms of minimizing the action value function as expressed in the second equation.

$$V_t^{\pi^*}(S_t) = \min_{x_t} \left(Q_t^{\pi^*}(S_t, x_t) \right) \tag{4.5}$$

$$Q_t^{\pi^*}(S_t, x_t) = C(S_t, x_t) + \sum_{s' \in \mathcal{S}} \mathbb{P}(S_{t+1} = s' | S_t, x_t) \min_{x_{t+1}} Q_t^{\pi^*}(s', x_{t+1}) \tag{4.6}$$

4.1.4 Transition Functions

Besides the deterministic variables, the inventory position changes each time period t due to the stochastic variables: returning repairs Z_{repair}, returning components from customers $Z_{customer}$, and demand D. Realizations of the stochastic variables are denoted by $z_{repair,t,n}$, $z_{customer,t,n}$ and $d_{t,n}$, respectively. Probability distributions for the stochastic variables are given in Section 2.7. Each time period, the system updates the inventory model due to the decisions made in the corresponding time period. The updating of the stock levels on-hand, at the repair shop, and the customer are explained in Equation 4.7a to 4.7c, respectively. For example in Equation 4.7a, the stock on hand in the next time period ($s_{oh,t+1,n}$) is equal to the current on-hand stock ($s_{oh,t,n}$) plus the repairs ($z_{repair,t,n}$) and new buys ($x_{t,n,buy}$) in this t, minus the demand of the current t ($d_{t,n}$). The updating of the other two equations works similarly.

$$s_{oh,t+1,n} = s_{oh,t,n} + z_{repair,t,n} + x_{t,n,buy} - d_{t,n} \tag{4.7a}$$

$$s_{repair,t+1,n} = s_{repair,t,n} + z_{customer,t,n} - z_{repair,t,n} \tag{4.7b}$$

$$s_{customer,t+1,n} = s_{customer,t,n} + d_{t,n} - z_{customer,t,n} \tag{4.7c}$$

Because we are assuming our model to work with discrete time periods, we must define the sequence of the events happening at each time period t (Topan, Tan, van Houtum, & Dekker, 2018):

1. The current state location is determined on given inventory positions from the previous time period t.
2. Demand ($d_{t,n}$) and the returns from customers ($z_{customer,t,n}$) for the current time period is known.
3. For decision $x_{t,n,f}$, the direct expected costs are calculated. The objective cost function considers the optimal decisions to prevent backorder, holding, and intervention costs.
4. Based on the decision, the number of returns from repair are realized $z_{repair,t,n}$.
5. The current state inventory positions for on-hand ($s_{oh,t,n}$), repair shop ($s_{repair,t,n}$), and customer ($s_{customer,t,n}$) are updated given the previous state with update procedures 4.7a - 4.7c. Corresponding holding-, backorder-, and intervention costs are payed.

To define the interaction between the three states, we give an example. Recall the CMA-program layout of the *exchange services*, visualized in Figure 1.1a. State variable s_{oh} represents the number of components in the pool, $s_{customer}$ the components at the operator, and s_{repair} in the repair shop. When demand occurs, a component will move from s_{oh} to $s_{customer}$. If a component malfunctions, then the UU goes from $s_{customer}$ to s_{repair}. The final movement within this figure and states is the movement of a SU from s_{repair} to s_{oh} after repair.

4.1.5 Transition Probabilities

The transition probabilities are dependent on different elements. By the diagrams' principle, discussed in Section 3.4.3, we move from one state to another with a given probability. In our case, the transition probabilities are dependent on the number of components returning from repair (Z_{repair}) and the demand (D). We leave the return from customers $Z_{customer}$ out of scope for this probability because within the currently used planning horizon, these do not impact the number of components in the repair shop $s_{repair,t,n}$. With the probability distribution, corresponding to the return from repair and demand rate, the expected inventory position S_{t+1} can be calculated. By multiplying the probabilities of $z_{repair,t,n}$ and $d_{t,n}$, we can determine the transition probabilities for the given time period. These probabilities are dependent on the component.

$$\mathbb{P}(S_{t+1} - S_t = K | x_t, S_t) = \sum_{L=0}^{\infty} \mathbb{P}(z_{repair,t,n} = L | x_t, S_t) \mathbb{P}(d_{t,n} = L - K | x_t, S_t) \tag{4.8}$$

The state difference is denoted as K partially expressed by the number of repairs L, with $K \in \mathbb{Z}$ and $L \in \mathbb{N}$. Naturally, this leads to a state transition $S_{t+1} = S_t + K$. This equation holds for both regular and expedited repairs. For the current scenario we assume that we can expedite all or repairs or nothing. Therefore, in case we expedite, we assume a union of events with regular repairs and expedited repairs. The mathematical definition of the returns of repairs is given in Equation 4.9a. To ensure we take the correct probability, we included the binary variable C in the probability function. This variable is equal to zero if we decide not to expedite, and one otherwise caused by Equation 4.9b. Here, $BigM$ is a random large number.

$$\mathbb{P}(z_{repair,t,n} = X) = C\mathbb{P}(Z_{reg} = X) + (1 - C)\mathbb{P}(Z_{reg} = X) \cup \mathbb{P}(Z_{exp} = X) \tag{4.9a}$$

where:

$$(1 - C) \leq x_{t,n,expedite} BigM \tag{4.9b}$$

$$BigM = 10.000 \tag{4.9c}$$

$$C \text{ is binary, and } X \in \mathbb{N} \tag{4.9d}$$

To clarify the theory about transition probabilities, we use Figure 4.1. We assume $s_{oh,t,n} = 2$, and $s_{repair,t,n} = 3$. Now, the agent can decide whether to do nothing, expedite, or buy a component. For this example we assume that $Z_{return,t,n} \sim B(n, p)$, $D_{t,n} \sim Pois(\lambda)$. The parameters included in the example are $n = s_{repair,t,n} = 3$, $p_r = 0.1$, $p_e = 0.15$ and $\lambda = 0.5$ for the Binomial and Poisson distribution, respectively. Note that Binomial probability is defined for a regular p_r and expedited p_e situation. With the given parameters, we can limit the range for inventory change to a reasonable possibilities for the inventory on $t + 1$. Probabilities for demand, combined with both normal and expedited repairs are given in Table 4.1.

The table represents the possible additions or subtractions from the inventory position during a time period t, i.e., these are the probabilities that current on-hand stock $s_{oh,t,n}$ increases or decreases with X components to the stock in the next time period $s_{oh,t+1,n}$. As can be found in the table, the probability that the inventory position decreases with 3 or 4 is very little in both situations. An example to clarify reading the table, the probability that the inventory is equal to 3 in the next time period ($s_{oh,t+1,n} = 3$) is equal to the probability that the on-hand stock grows with 1 is 0.1557 or 0.4755, for respectively regular and expedited repairs. The corresponding backup diagram is visualized in Figure 4.1. Since we assume that a new buy component arrives at $t + 1$, the probabilities for change in inventory are equal to the do nothing scenario for $s_{oh,t,n} + x_{t,n,buy}$. For the example back-up diagram, no combinations are considered.

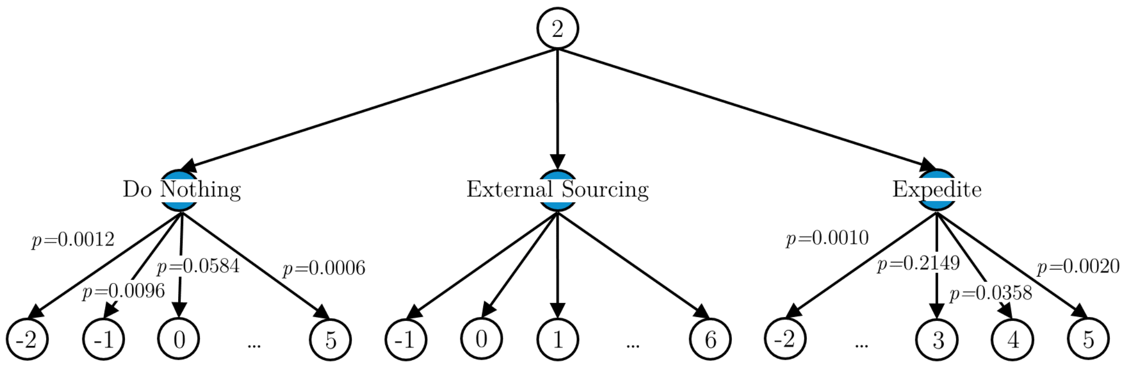


Figure 4.1: Example back-up diagram with $S_t = 2$

4.1.6 Future Addition of Interventions

In the current decision-making model three interventions are included: *Do Nothing*, *Expedite* and *External Sourcing*. For future use, we recommend implementing more interventions in the model. Obviously, besides incorporating the future additions in the state formulation and transition probabilities, the future additions should be implemented in the decision space and reward functions. Interventions which can be added are explained in Section 2.3. We will briefly explain the implementation for two of these interventions, namely *emergency shipments*, and *cannibalization*.

First of all, we review the implementation of emergency shipments. This intervention is a deterministic intervention, comparable with external sourcing. We assume this intervention to be deterministic because the number of components available for emergency shipments is known for each time period. Therefore, if there are components available, we can decide how many components to use for an emergency shipment. Because the emergency shipment does not change the inventory position, only the decision space will change. The transition probabilities will also not change because no stochasticity is involved in the decision-making concerning emergency shipments.

Contrary to emergency shipments, we treat cannibalization as a stochastic intervention. As explained in Section 2.3, with this intervention, the repair shop tries to speed up a repair by using components of unserviceable components to make one serviceable again. When the OPP carries out this intervention, the number of repairs returning increases because the repair can happen faster than in the typical case. However, there is a probability that the repair shop does not finish that day, or there may be no component available for cannibalization. Therefore, transition probabilities are necessary to express the probability of success of returning components correctly.

Table 4.1: Regular and expedited repairs combined with demand transition probabilities

K	$\mathbb{P}(S_{t+1} - S_t = K \mid x_{exp,t} = 0, S_t)$	$\mathbb{P}(S_{t+1} - S_t = K \mid x_{exp,t} > 0, S_t)$
3	0.0006	0.0020
2	0.0167	0.0358
1	0.1557	0.2149
0	0.5179	0.4755
-1	0.2398	0.2116
-2	0.0584	0.0508
-3	0.0096	0.0083
-4	0.0012	0.0010

4.2 Exact Solution Approach

The problem, as approached now, is relatively small. Therefore, it is possible to solve the problem with an exact approach. For this exact approach, we use a Stochastic Dynamic Programming (SDP) method. This section will first explain the characteristics of dynamic programming. Further, we discuss the optimal path and the efficiency of dynamic programming.

4.2.1 Backward Recursion

Richard Bellman initially found Stochastic Dynamic Programming to model and solve decision-making problems under uncertainty (Bellman, 1957). Comparable with Deterministic Dynamic Programming, SDP is a framework which can be solved to optimality by working backward from the end of the problem towards the beginning, also known as backward recursion (Bertsekas, 2017). Besides the backward recursion, a forward recursion can also be used (Bertsekas, 2017). With a forward recursion, only the states that are necessary for the given state are calculated. However, since backward recursion gives a better scope of the value functions, we will continue with the backward recursion.

Typically, for problems with finite planning horizons, SDP can be sufficient. Infinite MDPs, on the other hand, are commonly easily to approach with, e.g., a Linear Programming approach to define the steady state. Consequently, the goal of the MDP is often to provide a long-term solution, whereas the SDP provides a solution for the given finite horizon. Therefore, the SDP represents a particular class of the infinite Markov Decision Process, where underlying stochastic processes are stationary that features the Markov property.

Winston (2004) characterizes dynamic programming by five points. We determine the applicability of dynamic programming for our problem by comparing our problem with the DP problem's characteristics. First of all, the problem can be divided into stages where decisions are required at each stage. I.e., at each time step, a decision is required to move towards the next state.

Second, each state has given properties that are needed to make an optimal decision. These properties, for example, contain information about the inventory position. Note that these properties do not include information about previously visited states. The optimal decision is purely based on the properties known about the current state.

Third, the chosen decision is responsible for transforming the current state towards the next state. Although the consequences of the decisions seem clear, this third characteristic needs a little more explanation. For both the deterministic as the stochastic dynamic programming problem, the consequences differ. In the case of Deterministic Dynamic Programming, we know the next state directly from the given action. With stochastic dynamic programming, an action only determines the probability distributions for transitioning from one to another state.

The fourth point Winston addresses is state independency. In other words, the optimal decision for the current state is independent of the previously chosen decisions and visited states. As mentioned in Section 3.4.2, the future is independent of the past given the present (Z. Ding et al., 2020; Silver, 2015).

Finally, the problem can be divided into sub-problems. Therefore, there must exist a recursion that relates all costs or rewards of all time steps $t, t+1, t+2, \cdots, T-1, T$. The recursion formula formalizes the backward procedure. This characteristic is in line with Bellman's principle of optimality, as discussed in Section 3.4.3. The recursion formula is mathematically expressed in Equation 4.10 (Bertsekas, 2017).

$$V_t(S_t) = \min_{x_t \in \mathcal{X}_t} \left\{ \mathbb{E}(R_t|S_t, x_t) + \alpha \sum_{S_{t+1}} \mathbb{P}(S_{t+1}|S_t, x_t) V_{t+1}(S_{t+1}) \right\} \tag{4.10}$$

In this formula, $V_t(S_t)$ denotes the maximum expected reward that can be obtained during states $t, t+1, \cdots, T$, given the current state S_t. The recursion function consists of two parts: the expected reward now given current state and action (S_t, x_t), and the future expected rewards. In short, we can say that the recursion function determines which action x_t maximizes the sum of the current reward and cost-to-go function for the given state.

The expected reward in state t is calculated by a given reward function for the given action x_t. We calculate the cost-to-go function to determine the potential of particular actions in the given state. We can calculate this by multiplying the transition probability from S_t to S_{t+1} defined by the action with the maximum expected reward for the next time step (V_{t+1}). The transition probabilities should add up to one since each decision leads to a state at $t+1$. Included in the cost-to-go function is the discount factor $\alpha \in [0, 1]$. The discount factor represents the importance of future rewards in comparison with current rewards. When the alpha is lower, future rewards are less important.

All in all, the MDP described in Section 4.1 fulfills all five characteristics as described above; so, we can define the Markov Decision Process as a Stochastic Dynamic Programming problem. Since our goal is to find the solution for the finite plan horizon, we can determine the steady-state of the SDP.

4.2.2 Optimal Policy & Computational Efficiency

So far, the optimal expected reward is calculated for the given state S_t. To determine the optimal decision strategy, we determine which actions will return each separate state's best reward. Note that the use of dynamic programming is way more efficient than enumerating over all possible options since we are not determining all combinations of policies (Winston, 2004). Although dynamic programming can be quite efficient for some problems, it does not apply to large problems. Because the number of states grows exponentially with the number of state variables, it is difficult to generate an exact solution (Sutton & Barto, 2018). We have earlier referred to this problem as the *Curse of Dimensionality*.

At first, we describe the problem as a small state space problem. Therefore, modeling the problem as an SDP and solving it with a backward recursion still generates optimal results. We use the exact solution of the SDP to compare the quality of both simulation approaches' decision-making. As mentioned, an SDP approach works best for relatively small state space instances. When different components are calculated simultaneously in future research, the state space will increase, which is time consuming to enumerate the different options. It is assumable that the exact method is not able to solve the problem instance anymore.

4.2.3 Solving Stochastic Dynamic Programming in Short

In conclusion, backward recursion can be used for the exact approach of decision-making problems under uncertainty. The backward recursion algorithm is effective on problems that can be characterized as described in this section. To generate a better overview of backward recursion, Figure 4.2 visualizes the algorithm's steps. Additionally to the flowchart, Appendix F gives the corresponding Python pseudo-code for the SDP.

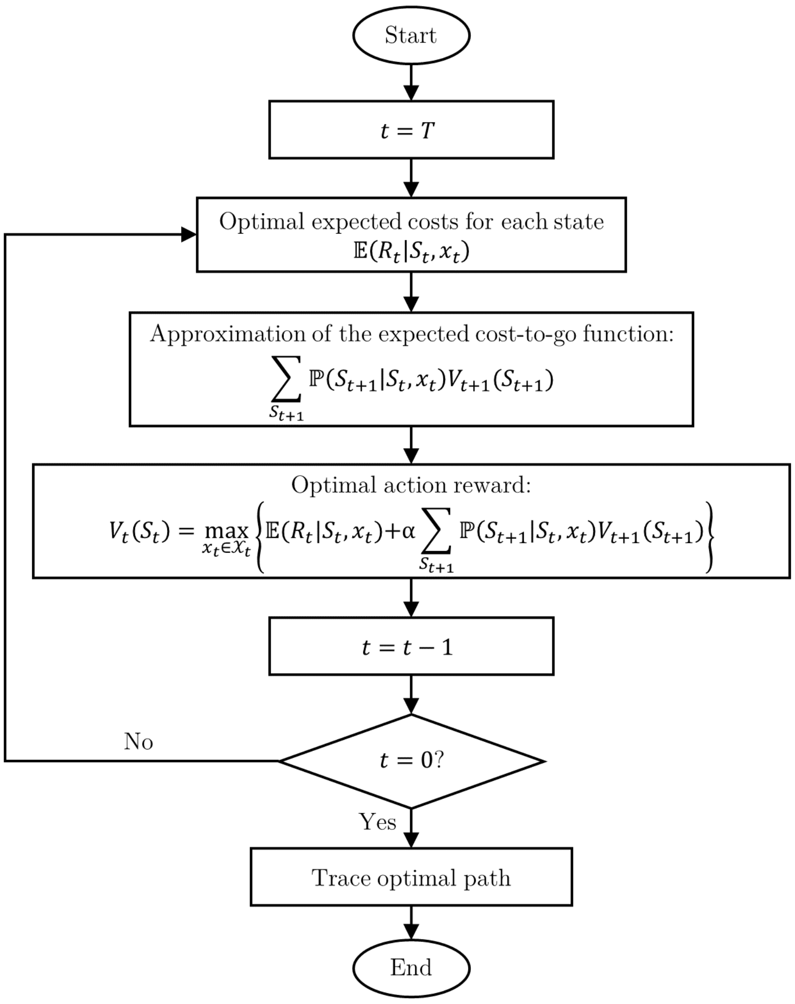

Start
$t = T$
Optimal expected costs for each state $\mathbb{E}(R_t|S_t, x_t)$
Approximation of the expected cost-to-go function: $\sum_{S_{t+1}} \mathbb{P}(S_{t+1}|S_t, x_t)V_{t+1}(S_{t+1})$
Optimal action reward: $V_t(S_t) = \max_{x_t \in \mathcal{X}_t} \left\{ \mathbb{E}(R_t|S_t, x_t) + \alpha \sum_{S_{t+1}} \mathbb{P}(S_{t+1}|S_t, x_t)V_{t+1}(S_{t+1}) \right\}$
$t = t - 1$
$t = 0$?
No
Yes
Trace optimal path
End

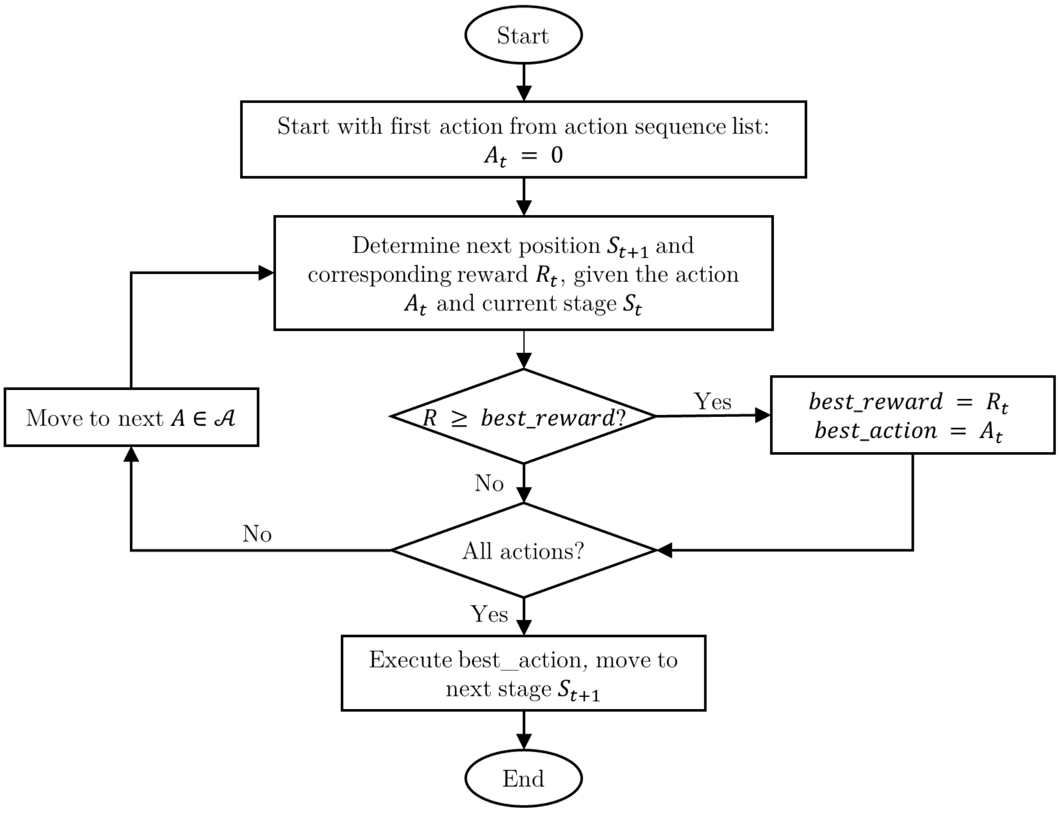
Figure 4.2: Structure of the SDP algorithm

4.3 Heuristic Approach

The second model we dive into is the heuristic solution. The heuristic-based solution aims to determine the planning and learning algorithm's performance quality and provide an alternative solution approach. This alternative can be for the Dyna-Q if the heuristic performs better than the planning and learning algorithm. Additionally, the heuristic is an alternative for the SDP if the state space increases too much for the dynamic programming approach. This section explains the chosen heuristic, the motivation of the heuristic, and the heuristic design.

4.3.1 Heuristic of Choice

For the heuristic approach, we use an approximation in the value space while ignoring the cost-to-go function. In each time period t, the agent chooses the action which gives the best direct reward. We use a myopic policy algorithm simulation for the first step approximation, referred to as a greedy algorithm. Greedy algorithms are commonly used for Operations Research neighboring solutions (Black, 2005). Since the greedy algorithm does not consider all branches of alternatives, the greedy approach is often a quick heuristic and easy to implement. Nonetheless, since the heuristic always chooses the action which returns the best direct reward, the chance of getting stuck in a local optimum is large. The greedy heuristic will always avoid decisions where direct costs are included, unless there is no other option.



Start
Start with first action from action sequence list: $A_t = 0$
Determine next position S_{t+1} and corresponding reward R_t, given the action A_t and current stage S_t
Move to next $A \in \mathcal{A}$
$R \geq best\_reward$?
Yes
$best\_reward = R_t$ $best\_action = A_t$
No
No
All actions?
Yes
Execute best_action, move to next stage S_{t+1}
End

Figure 4.3: Structure of greedy heuristic

Currently, the IAC's operational decisions tend to maximize the expectation of the direct reward. This approach corresponds with our defined greedy algorithm. Although the greedy approach is not necessarily the optimal approach for this problem, the simple heuristic functions as a good benchmark for the planning and learning algorithm's outcome. Therefore, we suggest future research to experiment with different heuristics, such as simulated annealing, to determine the heuristics potential.

4.3.2 Heuristic Design

The greedy decision-making is shown in a flowchart in Figure 4.3. With this heuristic we try every possible action to find which action gives the best direct reward. Then, with this action, we move to the next state. To evaluate the performance of this heuristic, we use a simulation. The simulation functions as an imitation of reality by iterating the planning horizon over multiple episodes. Although the greedy heuristic does not consider future rewards, we simulate 14 days to provide appropriate comparable environment behavior. In this way, we can compare the heuristic and RL algorithm most fair. The Python-code is presented in Appendix F.

In short, the greedy algorithm performs the action which gives the best reward for the current state. In other words, the greedy algorithm will do nothing unless there is no other option. To evaluate the performance of this algorithm, we use a simulator which mimics 14 days of the planning horizon for a fair comparison with the RL algorithm.

4.4 Planning and Learning Solution Approach

For well understanding of the Dyna-Q algorithm, we will review the structure (Section 4.4.1) and the strengths (Section 4.4.2) of the algorithm. From the earlier given algorithm, we present in Section 4.4.3 the general overview. Combined, this will support the decision for working with the Dyna-Q algorithm.

4.4.1 Structure of Dyna-Q

In Section 3.4.5, we reviewed the difference between model-free and model-based algorithms, and additionally, the hybrid algorithm Dyna-Q. To provide a better overview of the Dyna-Q algorithm, we dive further into this algorithm's structure. This will help us understand the programming steps that follow. Note that the algorithm is already discussed in Appendix C.3.3. As the literature describes, the learning model always consists of two parts: a reward and transition component (Silver, 2015). This model describes how the agent understands the environment. When parsing the algorithm, as discussed in Algorithm 3, we find that the algorithm consists of a model-free algorithm, Q-Learning, and the model-based part, the Dyna structure (Sutton, 1990).

Figure 4.4 displays the relationship between the experience, model, and value or policies. Each arrow shows a relationship of influence and presumed improvement. The relationships represent the behavior of the Dyna-Q algorithm. We read this as follows: the agent follows a given value or policy function and acts to it. Next, based on the observation (or experience), the agent learns by direct reinforcement learning. For direct reinforcement learning, we use Q-Learning. Besides direct learning, the agent creates a second model based on the observations so far. Based on this known model, the agent makes decisions for already visited states to get closer to that state's true value (planning). Therefore, the Q-table is updated both with direct RL and planning steps for a faster converging solution.

When we translate this to the algorithm given in Algorithm 3, the Q-Learning and the Dyna component can be explained as follows. The Dyna-Q algorithm first initializes both the Q-table and the $Model\ \mathcal{M}(S, A)$. Next, steps a to d give the normal path of the Q-Learning algorithm. These steps contain (a) the definition of the current state S, (b) determining an action A, (c) the observation of the reward R and the next state S' given the action A, and (d) the updating procedure of the Q-table.

To improve the learning pace of the process, we translate the model-free knowledge to a model-based algorithm. For this model, we know the states and the actions. However, we assume that the model learns rewards and transition probabilities during the process. We can train the model using a Table Lookup Model, Linear Expectation Model, or Linear Gaussian Model. Because the Table Lookup Model provides reasonable solutions following the literature (e.g., Silver (2015)), and it is easy to implement, we use this model for further implementation. We explain this method next.

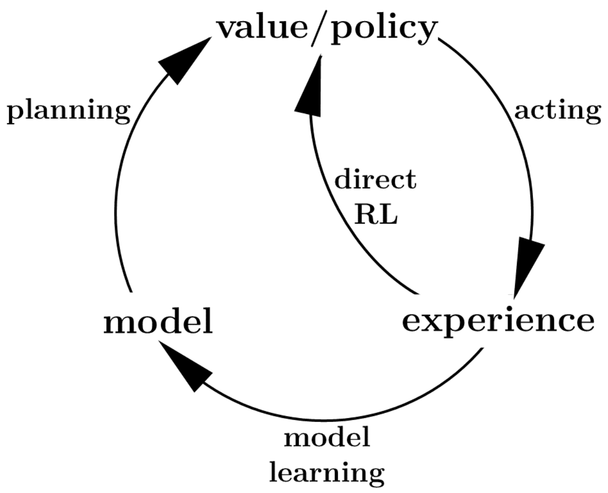


Figure 4.4: Relationships of planning, learning, and acting (Sutton & Barto, 2018)

The transition probabilities for the model are expressed as $\mathcal{P}_\eta[s,a,s']$, which represents the probability that given a state and action, the agent moves to another state. We count how often the transition from state $S_t = s$ to $S_{t+1} = s'$ occurs during the Q-Learning algorithm. Each time the transition happens the count variable $\mathcal{P}^c_\eta[s,a,s']$, is incremented by one. The transition probability observed so far is then calculated as noted in Equation 4.11. During each episode of the learning algorithm, the model update happens. Consequently, the algorithm chooses a state and an action based on the determined probabilities resulting from the model update. Obviously, in the random state and action determination, states, and actions that are visited or used more often have a higher probability of being chosen.

Besides the probabilities, we calculate the reward function for the model. We can denote $\mathcal{R}_\eta[s,a]$ as the expected reward for a given state $S_t = s$ and action $A_t = a$. The value of the expected reward is calculated in Equation 4.12. The equation consists of the learning rate α and the reward value of the Q-Learning algorithm for state-action pair $\mathcal{R}[s,a]$. We include $(1-\alpha)\mathcal{R}[s,a]$ to smooth the updating process. Furthermore, we add the direct reward $R(s,a)$ to fully update the model reward function.

$$\mathcal{P}_\eta[s,a,s'] = \frac{\mathcal{P}^c_\eta[s,a,s']}{\sum_i \mathcal{P}^c_\eta[s,a,i]} \tag{4.11}$$

$$\mathcal{R}_\eta[s,a] = (1-\alpha)\mathcal{R}[s,a] + \alpha R(s,a) \tag{4.12}$$

In the planning section of the algorithm, we define the planning steps, n. The planning steps define the number of times that we run the Dyna-architecture. Sutton (1990) shows that higher planning steps show faster-converging opportunities. In each planning step, we choose a random $S_t = s$, with a random action $A_t = a$. From these experiments, we determine $S_{t+1} = s'$, R, and the Q-values as discussed. After the n planning steps, we return to the beginning of the algorithm. Important to note is that a Dyna-Q algorithm with $n = 0$ is the basic form of Q-Learning.

4.4.2 Strengths of Dyna-Q

Literature, as found, often discusses the Dyna architecture for maze problems. The agent can choose whether he wants to go up, down, left, or right within the maze. The power of the Dyna architecture here is that with each step in the algorithm, the learning steps will update the state's Q-value. Therefore, the Q-values likely improve and get closer to the true value. As

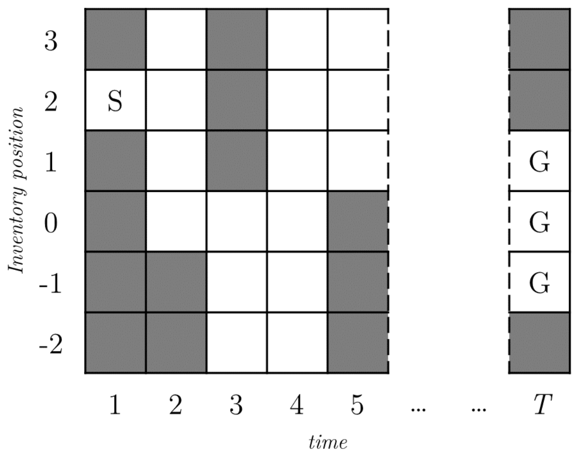


Figure 4.5: The inventory problem expressed as a maze

well as in a stationary situation, Dyna can find a suitable solution for a changing environment. Within the changing environment, Dyna-Q+, as described by Sutton and Barto (2018), can be a powerful method. However, the Dyna-Q+ algorithm is very computational intensive which is not attractive for large state spaces. We explain this method in Appendix C.3.3.

Figure 4.5 represents the inventory problem as a maze problem. In our situation, we will always start at $t = 1$, where the starting state is known. Due to variations in the stochastic variables, such as demand and repairs, different states are unavailable at $t + 1$. These are displayed as gray squares and represent the obstacles of the maze. While choosing an intervention, the agent moves one step in time and moves up or down depending on the action. This maze aims to reach the terminal state, which is each state at time T.

4.4.3 Dyna-Q Applied

The structure of the Dyna-Q algorithm is visualized in Figure 4.6. The algorithm, as denoted in Algorithm 3, is directly translated into the flowchart. Similar to the other modeling approaches, we present the corresponding Python pseudo-code in Appendix F.2.2, with additional code structures as cost-function and the class definition of the inventory environment. Earlier, we discussed that the Dyna-Q algorithm is model-free. To imitate the real-world, we use a simulation model where we can test the Dyna-Q's performance. we test it with different planning steps in the experiment setting, including $n = 0$ to find the basic Q-Learning performance.

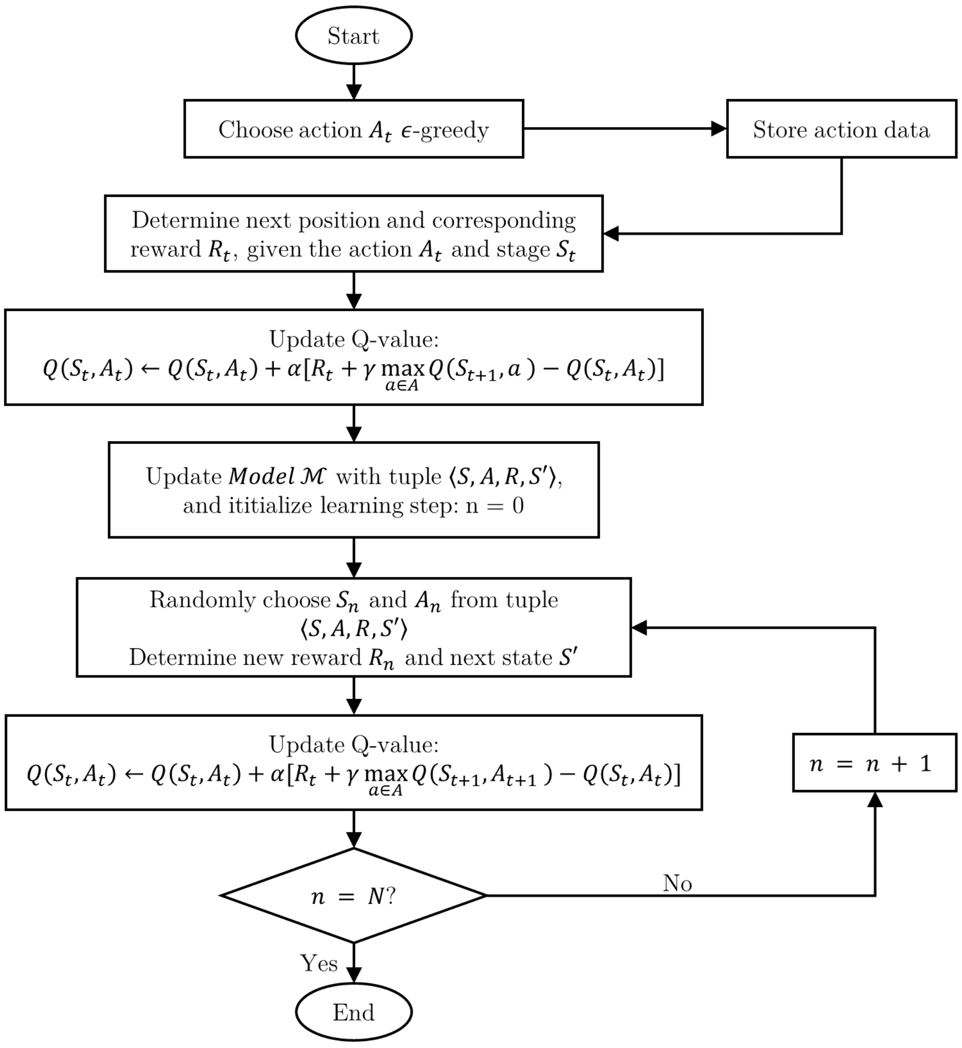

Start
Choose action A_t ϵ-greedy
Store action data
Determine next position and corresponding reward R_t, given the action A_t and stage S_t
Update Q-value: $Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha[R_t + \gamma \max_{a \in A} Q(S_{t+1}, a) - Q(S_t, A_t)]$
Update $Model\ \mathcal{M}$ with tuple $\langle S, A, R, S' \rangle$, and ititialize learning step: n = 0
Randomly choose S_n and A_n from tuple $\langle S, A, R, S' \rangle$ Determine new reward R_n and next state S'
Update Q-value: $Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha[R_t + \gamma \max_{a \in A} Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t)]$
$n = n + 1$
$n = N$?
No
Yes
End

Figure 4.6: Structure of the Dyna-Q algorithm

4.4.4 Dyna-Q in Summary

In summary, Figure 4.4 shows that we can apply direct reinforcement learning and model learning to plan and learn policies and value functions with the agent's experience. Therefore, Dyna-Q is a combination of: direct reinforcement learning from real-world experiences, updating of an internal model by the observed tuples, and simulating the experiences by using a model (Sutton & Barto, 2018; Silver, 2015). While building the model for our problem statement, we use the following structure; *Initialize all variables*, *iterate Q-Learning step for the time horizon*, *Define Model $\mathcal{M}(S, A)$*, *Simulate $\mathcal{M}(S, A)$ n times.* The Dyna-Q algorithm is strong in the learning capacity through the planning step. This helps to converge faster towards an acceptable solution.

4.5 Conclusion

The goal of this chapter is to help to answer the fourth sub-question. To create an overview of our problem statement's solution design, we explained the model approach in this chapter. We discussed the applied stochastic dynamic programming framework and three solution approaches.

Stochastic Dynamic Programming Framework In this chapter we discussed the outline of the SDP related to our problem in Section 4.1. The SDP consists of *decision*, *state*, *reward*, and *transition probabilities*. The dynamic problem can be defined in the following characteristics. The decision variable $x_{t,n,f}$ defines which intervention f is carried out at time step t for component n, displayed in Equation 4.2. Next, we defined the state-space of the MDP. The state can be defined as the inventory position, $I_{t,n}$, at time t, for component n. The inventory position is dependent of the current stock, $s_{oh,t,n}$, components in repair, $s_{repair,t,n}$, components at the customer, $s_{customer,t,n}$, and new buys, $x_{t,n,buy}$, of a component n on time t. The relationship is noted in Equation 4.7.

The reward function is given in Equation 4.3. This equation includes backorder costs $BO(s_{oh,t,n})$, intervention costs $E_{n,f}$, and holding and shortage costs $L(I_{effective,t,n})$. In this way, the agent is forced to make the trade-off between preventing a backorder and keep the holding costs for the long-term as low as possible. Our model connects the short-term decisions with long-term outcomes by validating the stock levels with the tactical level. Since our goal is not to overrule tactical decisions but optimize decision-making on the operational level, we penalize both way deviations. Finally, we determined the transition probabilities in Equation 4.8. The probabilities are dependent on the stochastic variables repair $Z_{return,t,n}$ and demand $D_{t,n}$.

Solution Approach For the solution approach, we discussed three different methods: backward recursion, Greedy Algorithm, and Dyna-Q algorithm. The state-, decision-, and action-space are all three relatively small. Therefore, we are still able to solve the problem exactly. The SDP works following the recursion formula (Equation 4.10) and given the structure from the flowchart in Figure 4.2 and pseudo-code of Appendix F.2.1. When state-, decision-, and action-space grow, the complete environment will grow. Due to the *Curse of Dimensionality* it will be very time consuming to solve the problem with an exact solving method. Therefore, a heuristic or planning and learning approach is necessary.

For the validity of the algorithm, we compare the quality of the Dyna-Q algorithm with decision-making following a greedy heuristic. With the simulation, we create a big data set where statistical analysis can discover trends or correlations. Finally, the model design of the Dyna-Q algorithm is reviewed. Appendix F.2.2 shows the structure of the Pseudo-code applied in Python. In conclusion, the exact solution of the backward recursion can be compared with the alternative models (Greedy and Dyna-Q) to test the quality of the alternatives.

5 | Experimental Results, Analysis, and Optimization

This chapter analyzes and presents the results of the three different solution approaches, as discussed in Chapter 4. In Section 5.1, we discuss our assumptions and parameter tuning for implementing three solving methods to our problem statement. Next, we will briefly discuss the policy learning for the backward recursion and Dyna-Q algorithm, in Section 5.2. Finally, we present our results of the backward recursion, greedy algorithm, and Dyna-Q algorithm within different experimental settings in Section 5.3.

5.1 Assumptions and Parameter Tuning

Within this first section, we will briefly introduce and substantiate our assumptions for the experimental setting. Additionally, we will also explain the expected impact for each of the assumptions and which model each assumption applies most. Then, we discuss parameter tuning for our experimental setting.

5.1.1 Experimental Assumptions

The assumptions we discuss next, concern the following topics: *base stock level*, *state space*, *pseudorandom numbers*, and the *starting state*. These assumptions apply to all three solution approaches both in learning and evaluating.

Base Stock Level Our first assumption concerns the base stock level (or tactical level) during the experiments. In Section 2.7.1, we calculated this value as the current tactical level of The IAC. Therefore, for simplicity and to stay close to the real setting, we assume tactical level $I_{tactical} = 2$ for all training and evaluation experiments.

State Space To be able to compare the state space properly, we have to frame it accordingly for all solution approaches. Therefore, we will use the following intervals for the state tuple $(s_{oh,t}, s_{repair,t}, s_{customer,t}, t)$: $s_{oh,t} \in [-3, 3]$, $s_{repair,t} \in [0, 1]$, $s_{customer,t} \in [0, 5]$, and $t \in [1, 14]$. With the limitation of maximum five components in the loop, and therefore five start inventory positions, we have a total of $65 \times 5 \times 14 = 5.880$ possible states.

Pseudorandom Numbers Third, we discuss the randomness in the different model approaches. In our objective to find a suitable policy for the spare part intervention problem, we try to evaluate and compare the models as well as possible. Therefore, we use *pseudorandom numbers* to reproduce an exactly corresponding experimental setting for each of the three different modeling approaches. We do this by fixing the random number seed in Python. As a result of this, *demand*, *returns from repairs*, and *returns from customers* behave exactly the same in each experiment.

Starting State Next, we discuss our approach to the starting states of each episode. For each experiment, we fix the stock on-hand $s_{oh,t}$, in repair $s_{repair,t}$ and at the customer $s_{customer,t}$. We will use starting states with some challenges involved for the experiments, i.e., we start in a state with already two backorders. We expect that the decision support system is obliged to use one or more interventions.

Backorder Counter As mentioned in Section 4.1.3, for the SDP we assume the backorders penalty to increase over time. Recall Equation 4.3b, where a^{g_n} represents the backorder counter for each state. Because we cannot easily count the backorders in a backward manner, we use another equation for the exact solution approach. Here, we use the following procedure: $a^{g_n} = 1.1^t$. Since the probability of a backorder occurring is larger at the end of the time horizon, we use a penalty dependent on time. For the greedy heuristic and Dyna-Q we will use $a^{g_n} = 1.15^{g_n}$, where g_n counts the days that backorders are open. Since the SDP wants to reach the lowest cost position, including the cost-to-go function in decisions, it is not likely that the system will ignore backorders for a long time period.

Discount Factor The final assumption we make concerns the discount factor α for the backward recursion and γ in the Dyna-Q algorithm. Recall that the discount factor represents the importance of future rewards in comparison with current rewards. The higher alpha, the more important the future rewards are. Since we are interested in the long-term impact of the decisions, a high discount factor is required. For this reason, we assume $\alpha = \gamma = 1$.

5.1.2 Initialization and Parameter Tuning

The parameter tuning helps us in defining the environment for the experimental setup. Besides the assumptions mentioned previously, the backward recursion does not require parameter tuning for learning the optimal policy. Therefore, we discuss parameter tuning for Dyna-Q within the scope of learning. Next, we discuss the parameter tuning for an evaluation approach by simulation applicable for all three methods.

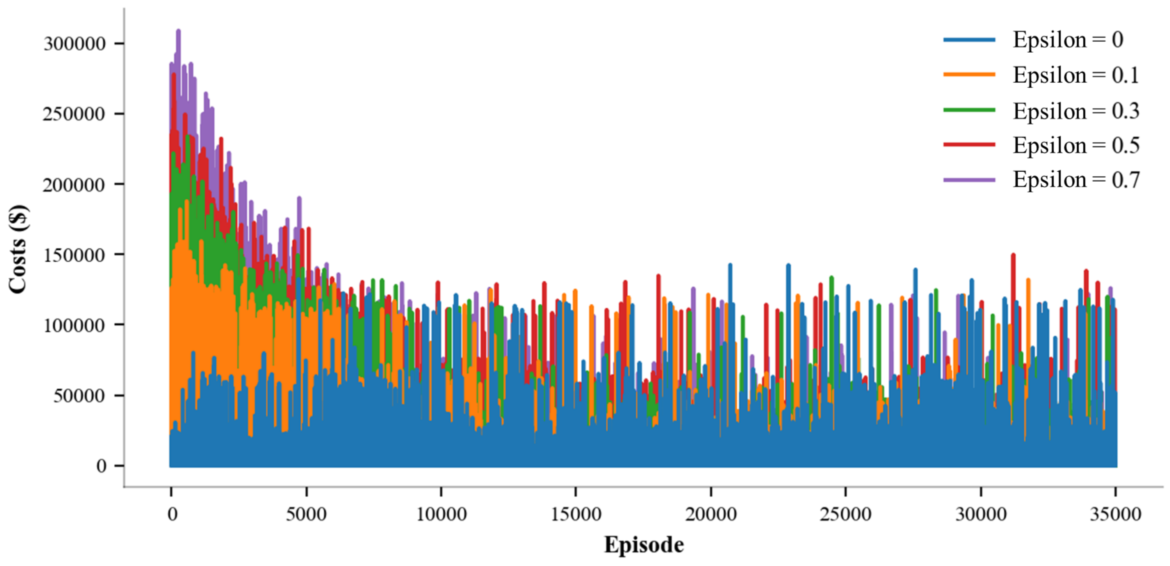
Learning

For the first initialization to update all state-action pairs of the Q-table, we use the greedy algorithm. Recall the Q-update function to be as noted in Equation 5.1. With this initialization, we try to find a well-performing ϵ first. To find the value of ϵ, we run the model 35.000 episodes with $\alpha = 0.3$ and $\gamma = 1$ for five different values of epsilon: $\epsilon = 0$, $\epsilon = 0.1$, $\epsilon = 0.3$ $\epsilon = 0.5$ $\epsilon = 0.7$. The learning rate α is set to 0.3, such that the newfound value does not impact the Q-value too much. The used γ is equal to one because we do not want to include discounted cost-to-go values in decision-making. To ensure a decay in epsilon, and therefore exploit more and more, we use an epsilon decay value $\epsilon_{decay} = 0.9995$. The results of the different values for the first experimental setting of ϵ are visualized in Figure 5.1a. Learning curves of the other experiments are presented in Appendix G. We choose an epsilon that shows decent learning for each experiment while including an acceptable part of exploring. Therefore, we conclude the following epsilon values: $\epsilon = 0.3$ for the high tactical preference, $\epsilon = 0.7$ for the high operational preference, and $\epsilon = 0.5$ for the equilibrium point.

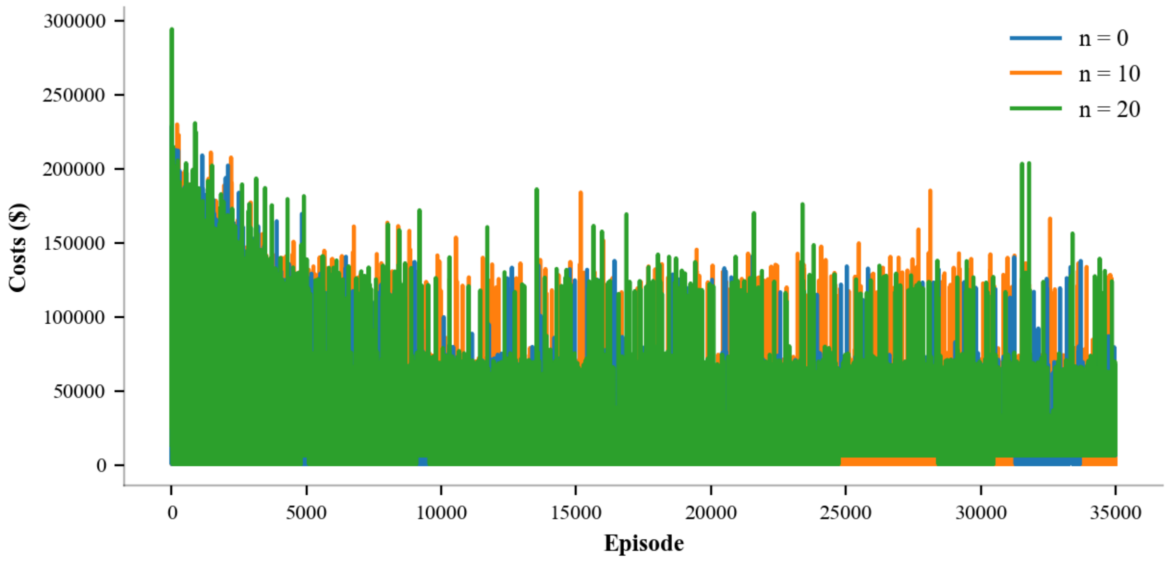
$$Q(S, A) \leftarrow Q(S, A) + \alpha[R + \gamma \min_a Q(S', a) - Q(S, A)] \tag{5.1}$$

Next, to define the planning steps n, we use $\epsilon = 0.3$, $\epsilon_{decay} = 0.9995$, $\alpha = 0.3$ and $\gamma = 1$ for a total of 35.000 episodes for the first experiment. With the greedy values as initialization input, we determine the episode values for $n = 0$, $n = 10$, and $n = 20$. Figure 5.1b presents the course of episode costs. From the figure, we conclude that the performances of all three approaches are comparable for this instance. As discussed in Section 4.4, Dyna-Q's strength is to converge faster with the planning steps incorporated. Therefore, we will proceed with the experimental setting $n = 20$.

In short, for the experiments, we found a value of $\epsilon = 0.3$, $\epsilon = 0.7$, and $\epsilon = 0.5$, respectively. With $\epsilon_{decay} = 0.9995$, the epsilon will decay each episode, such that we will exploit more and more. The learning rate α is set to 0.3 such that the newfound value does not impact the Q-value too much. Although we find that the model converges after approximately 15.000 episodes, we will use 35.000 episodes to learn all possible states properly. To find fast converging learning, we use $n = 20$ planning steps.



(a) Tuning ϵ with $n = 0$



(b) Tuning n with $\epsilon = 0.3$

Figure 5.1: Learning curves in original scenario of (a) epsilon and (b) planning steps

Evaluating

We evaluate all policies derived from the backward recursion, greedy heuristic, and Dyna-Q by a simulation model. As mentioned, we use a pseudorandom number generator with a common seed for demand and repair. The start states are fixed for the experiments. From Law (2015), we can derive the formula of calculating the number of replications for the experiment, denoted in Equation 5.2.

$$n^* = \min \left\{ i \geq n : \frac{t_{i-1,1-\alpha/2}\sqrt{S_n^2/i}}{|\bar{X}_n|} \leq \frac{\gamma}{1+\gamma} \right\} \tag{5.2}$$

With this equation, we can express the number of replications required to obtain a given relative error γ. Further $t_{i-1,\alpha/2}$ represents the t-distribution, and $\bar{X}_n$ and S_n^2 are the sample mean and variance. We run the following settings for the greedy heuristic to determine the number of replications of the evaluation: $\alpha = 0.05$, $\gamma = 0.025$ and $n = 25.000$, we find a value of $n^* \geq 11.535$. Since the calculated value of n^* is a minimum amount of repetitions required, we conclude to use 15.000 repetitions for our evaluation. Recall that we will learn the policies of the backward recursion and Dyna-Q preliminary, so we cannot change the policies during the evaluation.

5.2 Learning

Before evaluating and comparing the quality of the different policies given by the different algorithms, we first have to learn the (optimal) policy. Learning applies to backward recursion and Dyna-Q only, which we will discuss in this section. The greedy heuristic does not learn since it is a short-sighted heuristic. Therefore, the greedy algorithm is excluded from this section.

5.2.1 Backward Recursion

With the backward recursion, we start calculating at $t = T$. We work backward from the end of the planning horizon, calculating at each state in each time period the expected first step costs, including the approximate cost-to-go function. The states, for which the backward recursion is calculated, exist of the tuple $(s_{oh}, s_{repair}, s_{customer})$. Recall the formula used to solve the SDP, is as follows:

$$V_t(S_t) = \min_{x_t \in \mathcal{X}_t} \left\{ \mathbb{E}(R_t|S_t, x_t) + \alpha \sum_{S_{t+1}} \mathbb{P}(S_{t+1}|S_t, x_t) V_{t+1}(S_{t+1}) \right\} \tag{5.3}$$

Solving the SDP with a backward recursion usually is sufficient since the SDP calculates the expectation exactly without sampling it. However, because the backorder counter in the exact solution and the simulation function differs, there might be a mismatch in the optimal policy. For consistency of policy performance, we use the policy from the exact solution within an evaluation step.

Within the evaluation simulation, we run episodes with different start states. Naturally, the start states are given within the range of the state space. Because our cost function penalizes the terminal state dependent on the start state, we solve the SDP for $I_{total} = 0$, $I_{total} = 1$, $I_{total} = 2$, $I_{total} = 3$, and $I_{total} = 4$. Note that we do not take backorders into account for the total stock, and solving $I_{total} = 5$ is not relevant because it is an absorbing state. Therefore, solving these five problem instances is sufficient.

Table 5.1: Count and percentage of decisions in the total state space per start inventory

	$I_{total} = 0$		$I_{total} = 1$		$I_{total} = 2$		$I_{total} = 3$		$I_{total} = 4$	
Action	Count	%	Count	%	Count	%	Count	%	Count	%
Do Nothing	399	85	618	73	618	73	587	69	530	63
Expedite	0	0	130	15	130	15	120	14	96	11
Buy 1	40	9	43	5	43	5	57	7	94	11
Buy 2	25	5	26	3	26	3	32	4	54	6
Buy 3	4	1	4	0	4	0	10	1	22	3
Exp Buy 1	0	0	21	2	21	2	30	4	38	4
Exp Buy 2	0	0	3	0	3	0	9	1	11	1
Exp Buy 3	0	0	0	0	0	0	0	0	0	0

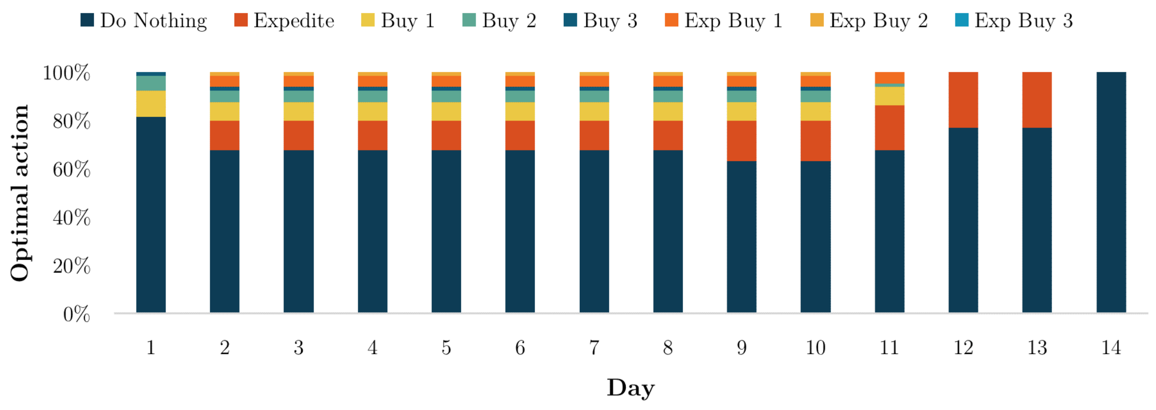


Figure 5.2: Optimal decision ratio per day, for $I_{total} = 3$

We present the complete policy tables per starting inventory in Appendix G.1. Table 5.1 gives the count and percentage of how often an intervention is optimal. Here, we see that at, e.g., $I_{total} = 0$, we do nothing for 85% of the time. We see that the only time we perform another action is when we have backorders, and the stock at the operators is less than the tactical level. For $I_{total} = 1$ and $I_{total} = 2$, we see a slight shift in policy behavior. Still, for the vast majority of the decision points, we do nothing. However, now we include expediting since there might be a component in the repair shop. We still see the extreme measures occurring in most backorder cases with an inventory position lower than the tactical level.

The policy behavior moves even further towards extreme measures for $I_{total} = 3$ and $I_{total} = 4$. Note that this is caused because we cannot move down in the inventory position during this experimental setting. Therefore, states lower than the starting state cannot be reached. Nevertheless, because of the tactical penalty, the SDP advises more extreme measures if we move further from these lower states.

Additionally, in Figure 5.2, we see the expression of different preferred actions per day. As well as the episode results, we see a majority of the optimal decisions to be do nothing, followed by expediting. We see that expediting is not optimal on the first day because the probability of a repair returning the first day is already high. Nonetheless, if the part did not return on day 1, we start expediting the next day.

5.2.2 Dyna-Q

For the learning process of Dyna-Q, we use the preset parameters; $\epsilon = 0.3$, $\epsilon_{decay} = 0.9995$, $\alpha = 0.3$, and $\gamma = 1$. Using a simulator, we can learn the policy for the current settings. We run the simulator 35.000 episodes and $n = 20$, where pseudorandom numbers determine demand and repairs.

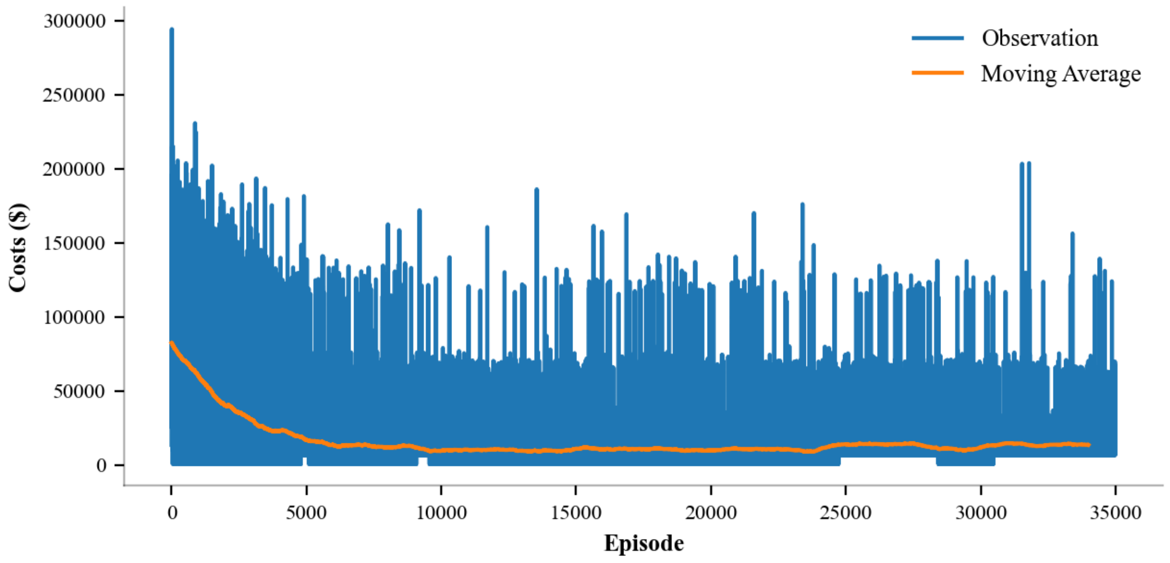


Figure 5.3: Learning of Q-values over episodes for the IAC's initial setting

Figure 5.3 shows us the Dyna-Q learning curve with the IAC's initial cost setting. Here, we present the actual learning curve and its moving average. We see a volatile result because of the changing start states and the low probability of an event occurring. We store the results of the learning simulation in a Q-table, such we can use them in our evaluation.

5.3 Evaluating

Within this section, we discuss the evaluation and comparison of the simulation model. We evaluate and discuss three different scenarios: the IAC's initial situation, low tactical impact, and the equilibrium point. We will explain the different scenarios and their purpose first. Next, we evaluate the numerical results of the simulation. The learning steps, discussed in Section 5.2, are repeated for the other two scenarios.

5.3.1 The IAC's Initial Setting

In Chapter 4 we presented the solution approach for our stochastic dynamic programming framework. Here, we defined our *states*, *decisions*, *costs*, *transition functions* and *transition probabilities*. The values used for this setting are derived from the desire, requirements, and data of the IAC, as discussed in Section 2.7.1.

Within this setting, we can find in the cost function (Equation 4.3) that the terminal state costs currently have an immense impact on the total costs. In, e.g., backorder costs, we included the turnover rate u_n but not in the terminal state costs. From our data analysis, we conclude that this value is approximately $u_n = 0.155$. So, the weight of our terminal state cost is 6.45 times higher than all other costs. Therefore, we conclude that operational interventions have little to no impact and will be ignored for this reason.

5.3.2 Low Tactical Impact

To determine the impact of operational decisions, we lower tactical limitations by using a turnover rate $u_n = 1$. Due to the significantly increased impact of operational interventions, we expect that tactical guidelines will be overruled most of the time.

The higher operational impact is best suitable for open inventory systems, where the inventory disappears when customer demand occurs. Although this is not the IAC's way of working in the CMA-loop, we can determine the differences in decisions due to the shifting focus.

5.3.3 Equilibrium Point

The CMA-program works with a closed inventory loop. Therefore, it is not desirable for the IAC to work with a strategy, as discussed in the second point. In case operational decisions overrule all other limitations, inventory will increase tremendously. However, if tactical limitations have too much weight in decision-making, a decision support system is unnecessary. Therefore, we will define the equilibrium point of the tactical and operational levels. The optimal decision-making should include a balanced approach.

5.3.4 Results and Analysis

We will compare the three solving methods for each scenario individually. We will do this, by comparing and evaluating each scenario on cost behavior, actions, and backorders.

Initial Scenario

The first scenario we look into is the IAC's original setting. This scenario includes a high tactical preference in decision-making. Cost, decision, and backorder results are denoted in Tables 5.2, 5.3, and 5.4, respectively. Additionally, we visualized the cost distribution of all three solution approaches in Figure 5.4. For this scenario, we see that the backward recursion performs better than the other two methods. Further, we see that the Dyna-Q solutions are relatively close to the optimal solution values for the first two start states. Only in the start state (-2, 1, 1), we see greedy and Dyna-Q performing comparable. Because we have one component in the repair shop, the decision-making is more straight-forward, leading to better predictable costs. For the final two experiments, we see that the greedy heuristic performs better than Dyna-Q. At the same time, we can find that Dyna-Q performs more actions and is, therefore, more conservative.

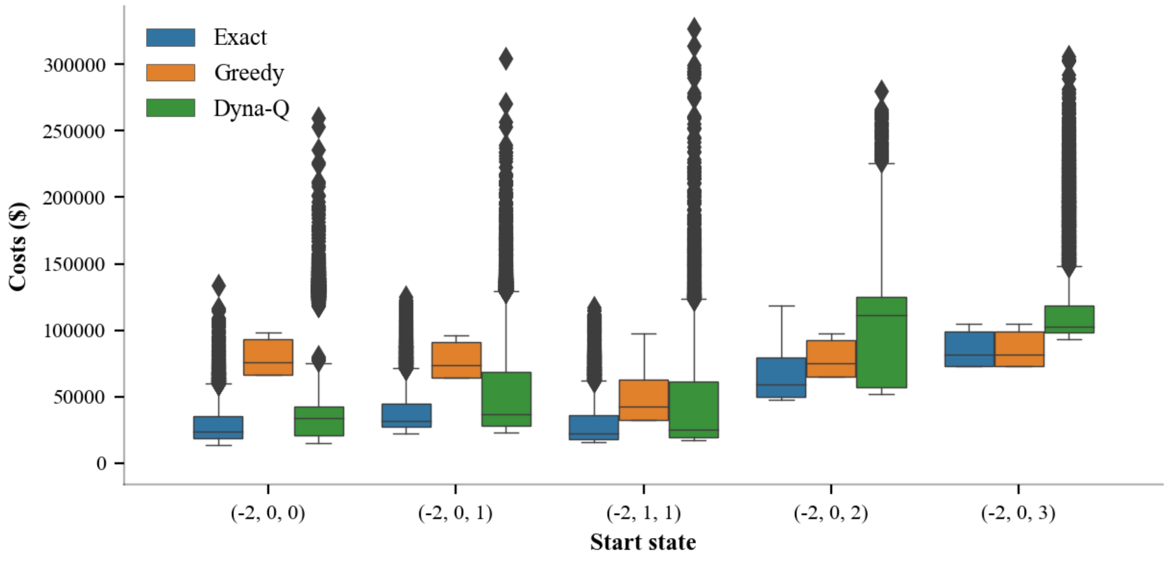


Figure 5.4: Costs per episode in initial scenario

Table 5.2: Cost performance of three solving methods in initial scenario ($)

	Exact		**Greedy**		**Dyna-Q**	
Start state	Mean	Median	Mean	Median	Mean	Median
(-2, 0, 0)	28.142,38	23.626,00	79.401,45	75.678,59	44.204,84	33.876,44
(-2, 0, 1)	38.474,38	31.853,75	77.281,31	73.621,91	58.285,25	36.935,40
(-2, 1, 1)	29.498,98	22.497,00	49.361,40	42.373,78	49.626,46	24.980,44
(-2, 0, 2)	66.559,47	59.281,54	78.336,44	74.993,03	95.759,03	111.157,75
(-2, 0, 3)	85.699,73	81.783,35	85.442,75	81.783,35	115.693,53	103.005,40

Table 5.3: Percentage action behavior of three solving methods in initial scenario

	Exact					**Greedy**					**Dyna-Q**				
Action	0	1	2a	2b	3	0	1	2a	2b	3	0	1	2a	2b	3
DN	91	92	91	92	100	100	100	99	100	100	88	80	87	88	88
Exp	0	0	1	0	0	0	0	1	0	0	0	0	0	0	0
B1	1	0	8	8	0	0	0	0	0	0	5	20	12	4	4
B2	8	8	0	0	0	0	0	0	0	0	7	0	0	7	8
B3	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0
EB1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
EB2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
EB3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 5.4: Expected backorders of three solving methods in initial scenario

	Exact	**Greedy**	**Dyna-Q**
(-2, 0, 0)	0,37793	2,28458	0,34077
(-2, 0, 1)	0,49755	2,28288	0,29193
(-2, 1, 1)	0,51419	1,41953	0,40696
(-2, 0, 2)	1,41684	2,28288	0,54185
(-2, 0, 3)	2,29020	2,28288	0,49789

Low Tactical Impact

For the low tactical impact, we conclude that by lowering the tactical planning costs' impact, the system includes more different interventions in the planning period. The optimal solution recommends now in situations where backorders occur to buy a new component, i.e., backorder costs' impact increases. This new cost distribution is visualized in Figure 5.5. Contrary to the backward reduction, we see that the greedy heuristic applies the same behavior as in the previous experiment; it does nothing as long as possible. Ignoring the cost-to-go function causes this behavior. The Dyna-Q algorithm shows in this experiment a significant improvement compared to the greedy heuristic and performs close to the optimal solution. Again, in the last two experiments, we see a better performance in costs of the greedy algorithm but also more EBOs than Dyna-Q. All results of the different solving methods are numerically supported in Tables 5.5, 5.6, and 5.7.

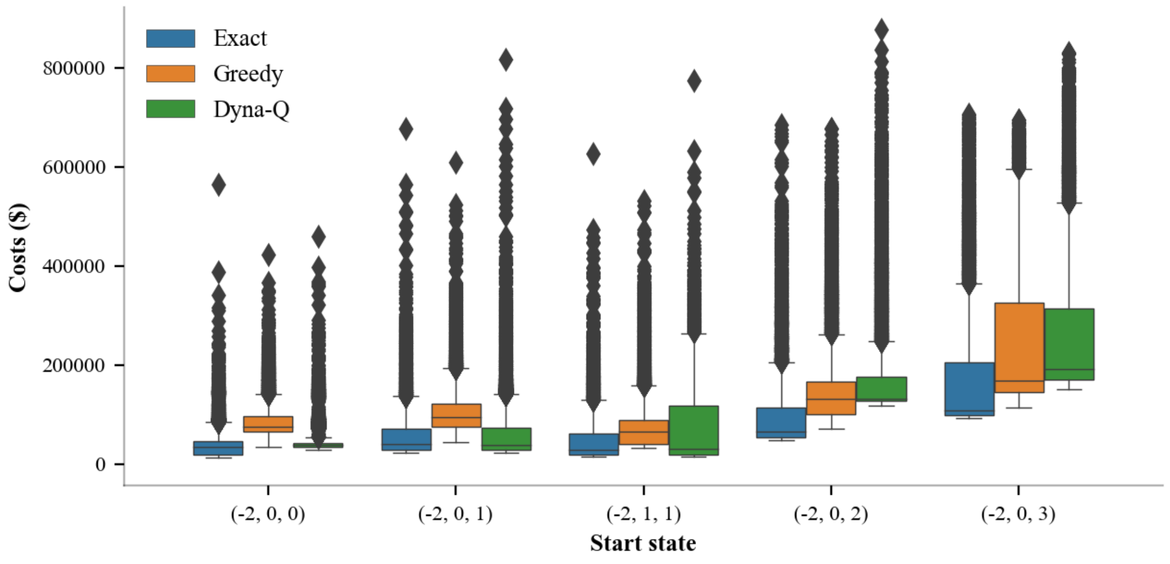


Figure 5.5: Costs per episode in scenario with low tactical impact

Table 5.5: Cost performance of three solving methods in scenario with low tactical impact ($)

	Exact		**Greedy**		**Dyna-Q**	
Start state	Mean	Median	Mean	Median	Mean	Median
(-2, 0, 0)	38.790,20	34.187,02	84.719,44	75.942,11	46.542,19	38.092,00
(-2, 0, 1)	57.731,13	41.084,02	105.578,40	94.252,05	63.662,39	38.223,1
(-2, 1, 1)	48.199,40	28.633,58	75.626,95	66.654,69	66.407,42732	31.396,905
(-2, 0, 2)	95.085,44	65.395,10	147.043,69	132.536,855	163.970,43	131.743,11
(-2, 0, 3)	166.365,27	108.598,10	236.114,84	169.201,27	253.365,74	191.880,42

Table 5.6: Percentage action behavior of three solving methods in scenario with low tactical impact

	Exact					**Greedy**					**Dyna-Q**				
Action	0	1	2a	2b	3	0	1	2a	2b	3	0	1	2a	2b	3
DN	88	88	88	89	92	84	85	87	86	88	91	87	85	72	48
Exp	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0
B1	5	4	4	3	0	6	6	6	6	5	1	5	12	15	43
B2	8	8	0	8	8	2	2	1	2	2	0	8	3	10	2
B3	0	0	0	0	0	0	0	0	0	0	8	0	0	3	8
EB1	0	0	8	0	0	5	5	5	4	3	0	0	0	0	0
EB2	0	0	0	0	0	2	2	0	2	1	0	0	0	0	0
EB3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 5.7: Expected backorders of three solving methods in scenario with low tactical impact

	Exact	**Greedy**	**Dyna-Q**
(-2, 0, 0)	0,19741	0,73581	0,16207
(-2, 0, 1)	0,20979	0,74224	0,21235
(-2, 1, 1)	0,21020	0,47649	0,22048
(-2, 0, 2)	0,26060	0,78731	0,37043
(-2, 0, 3)	0,49755	1,02743	0,58030

Equilibrium

In the previous two experiments, we saw the impact of a tactical and operational preference in decision-making; the tactical preference makes operational decisions unnecessary and vice versa. Therefore, we are interested in the equilibrium point of tactical and operational decision-making to include the best of both. To shift the impact of decisions from tactical to operational, we changed the turnover rate's value. Therefore, we divided this difference into ten steps in order to find the best suiting point. By solving the backward recursion for each of these points, we find the results presented in Table 5.8. We are interested in the point where we find a decent balance between tactical and operational decisions. We think point four is best suitable for this purpose.

Table 5.8: Percentage actions per preference step

	TP	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**OP**
Do Nothing	73	67	66	63	61	58	57	56	55	53	53	52
Expedite	15	16	15	15	14	13	13	12	11	11	11	11
Buy 1	5	7	7	8	9	10	10	11	12	12	13	13
Buy 2	3	4	4	5	6	6	7	7	7	8	8	8
Buy 3	0	1	2	2	3	3	4	4	4	4	4	4
ExpBuy1	2	4	4	5	6	6	7	7	7	7	7	7
ExpBuy2	0	1	2	2	3	3	4	4	4	4	4	4
ExpBuy3	0	0	0	0	0	0	0	0	0	0	0	0

Naturally, for the final experiment within the equilibrium point, we see again the exact solution outperforming the other two solution approaches. Also, for this experiment, the spread of the distribution is relatively small. Contrary to the other experiments, we see that the greedy algorithm performs poorly in this scenario; the greedy algorithm finds it hard to decide between tactical and operational levels. Therefore, both costs and backorders are high in this scenario. Dyna-Q, on the other hand, performs in this scenario very well. The results that Dyna-Q shows are in most situations relatively close to the optimal solution while having similar or even better EBO. Reinforcement learning shows its potential and applicability very well in this scenario.

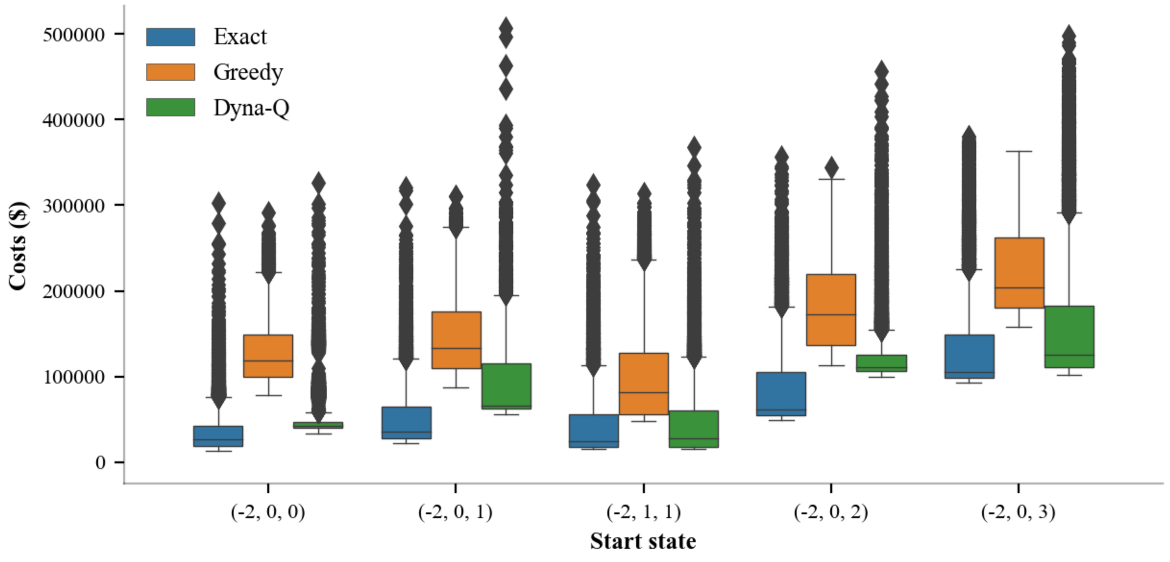


Figure 5.6: Costs per episode in equilibrium point

Table 5.9: Cost performance of three solving methods in equilibrium point ($)

	Exact		**Greedy**		**Dyna-Q**	
Start state	Mean	Median	Mean	Median	Mean	Median
(-2, 0, 0)	49.497,21	34.502,20	125.424,21	118.324,97	50.678,68	42.795,07
(-2, 0, 1)	49.494,70	35.703,08	144.346,13	133.716,01	83.593,17	66.445,00
(-2, 1, 1)	40.375,23	24.102,32	99.926,92	81.365,86	49.603,38	27.692,25
(-2, 0, 2)	84.040,00	61.992,69	178.627,19	172.258,98	121.774,90	110.530,00
(-2, 0, 3)	130.019,43	105.195,69	220.730,98	204.156,47	156.355,81	125.796,69

Table 5.10: Percentage action behavior of three solving methods in equilibrium point

	Exact					**Greedy**					**Dyna-Q**				
Action	0	1	2a	2b	3	0	1	2a	2b	3	0	1	2a	2b	3
DN	88	90	90	92	92	85	85	88	86	88	91	82	87	89	73
Exp	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0
B1	4	2	2	0	0	6	6	5	6	5	2	10	5	1	27
B2	8	8	0	8	8	2	2	1	2	2	1	8	0	2	0
B3	0	0	0	0	0	0	0	0	0	0	7	0	0	8	0
EB1	0	0	8	0	0	4	4	4	4	4	0	0	8	0	0
EB2	0	0	0	0	0	2	2	1	2	1	0	0	0	0	0
EB3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 5.11: Expected backorders of three solving methods in equilibrium point

	Exact	**Greedy**	**Dyna-Q**
(-2, 0, 0)	0.22229	1.64631	0.31488
(-2, 0, 1)	0.28702	1.64680	0.18311
(-2, 1, 1)	0.28954	0.99805	0.21333
(-2, 0, 2)	0.46247	1.65723	0.21782
(-2, 0, 3)	0.49755	1.76978	0.57264

5.4 Conclusion

In this chapter, we discussed the initialization, learning, and evaluation of our problem instance with three different solution approaches. In this way, we find the final answer on the fourth sub-question.

Learning Before evaluating the performance of the solution approaches, we learn the optimal policy of the backward recursion and the policy of the Dyna-Q algorithm. For the optimal policy, we see that, on average, 73% of the time, we do nothing in the scenario where tactical input has high importance. We see that the percentage of doing nothing decreased to 52% of the time for the operational preference. To learn the Dyna-Q policy, we use parameters: $\alpha = 0.3$, $\gamma = 1$, and $n = 20$ for 35.000 episodes. Furthermore, we use an $\epsilon_{decay} = 0.9995$ for $\epsilon = 0.3$ for the first, $\epsilon = 0.7$ for the second, and $\epsilon = 0.5$ for the third experiment.

Evaluating We run the solution approaches for three different scenarios; tactical preference, operational preference, and the equilibrium point. By comparing these three preferences, we create good insight into the relationship between the layers of decision-making. Naturally, the exact solution gives the optimal solution for all three scenarios. Further, we see that Dyna-Q performs better than the greedy algorithm when the starting inventory is lower than the tactical level in the first two scenarios. When the starting inventory is equal or higher than the tactical level, the greedy performs comparably. Nevertheless, in the equilibrium point, we see that the performance of Dyna-Q is close to optimal in all cases. Here, the greedy algorithm cannot find the right decision, suiting both operational and tactical decision-making.

In conclusion, for straightforward cases, heavy computational algorithms as reinforcement learning are not useful. We see a simple (greedy) algorithm performs comparable or better with the Dyna-Q algorithm in the states where buying leads to a high penalty. Both for tactical and operational preference, we see that when the starting state is higher than the tactical level, Dyna-Q starts performing worse. However, when combining the two worlds of operational and tactical decision-making, Dyna-Q outperforms a simple heuristic in every case. Therefore, by creating a decision-making environment where both operational and tactical levels are essential, the IAC can create higher availability with reasonable costs.

6 | Implementation

In the previous chapters, we discussed the solution design and results of a single item decision-making process. Because the Independent Aerospace Company (IAC) aims to use the decision support system for multiple items simultaneously, depending on the alert generation, we discuss the possibilities of implementing the proposed decision-making system. First, we will discuss the possibility of using a more complex problem and how to cope with scalability in Section 6.1. Additionally, this chapter discusses the possibilities and strengths of implementing the DSS in a Digital Twin in Section 6.2. With the information obtained from this chapter, we can answer the fifth and last sub-question of this research.

6.1 Scalability

The current problem instance, as solved in Chapter 5, works with a relatively small problem instance. Although this first step with a single item approach might already be valuable for the IAC, the aim is to include a more complex instance in future use. Therefore, we will discuss the possible pitfalls of scaling the problem instance and how to overcome this problem. We will consecutively discuss the scalability in the number of state and action spaces and the possibility of reinforcement learning with big data. Although there can be many other scalability types, we think these two types are the most relevant for the IAC's purpose.

6.1.1 Increasing State- and Action Space

Currently, we are using a tabular solving method. In many real-world problems, the state- and action spaces are enormous. For these instances, a tabular solving method is a very inefficient way of computing the solution. So, with an increasing state- or action space, the aim is to use an approach based on each state's features. Therefore, we make an estimation, also called a function approximation. The function approximation methods expect to receive examples of the desired input-output behavior of the function they are trying to approximate (Sutton & Barto, 2018; Varshavskaya, Kaelbling, & Rus, 2006). There are many different function approximators; the most common approximation functions are linear, neural networks, and decision trees (Z. Ding et al., 2020).

Nonetheless, from Sutton and Barto (2018), we conclude that not all approximators are suitable for all different reinforcement learning instances. This is because we encourage the RL system to learn online by interacting with the environment. Therefore, methods that support nonstationarity in the decision-making are best suitable for reinforcement learning.

Finally, Z. Ding et al. (2020) conclude that a way to leverage high-dimensional spaces' scalability is by using Deep Reinforcement Learning (DRL). For some problem instances, only a function approximation might not be sufficient to overcome the scalability issue. By including Deep Neural Network (DNN), the learning algorithm can solve a larger problem instance. This network develops features internally that help in learning and generalizing complex problems.

6.1.2 Learning with Big Data

So far, we have discussed multiple large instance cases, e.g., AlphaGo (DeepMind, 2019). Z. Ding et al. (2020) finds that DeepMind solves the DRL methods in a population-based training framework and advanced network structures. Within these approximate value-based methods, there is sample complexity that guarantees the approximate policy will get close to the optimal policy when there is lots of data by combining different sub-optimal policies. However, the corresponding computational time for a given problem is very inefficient. Techniques such as Imitation Learning and Hierarchical Reinforcement Learning strategies are often required for big-data instances (for further reading, see (Z. Ding et al., 2020)).

6.2 Digital Twin

This section reviews functionality and utility of a DSS within a Digital Twin environment for the IAC. We will first provide a general introduction about Digital Twins since there is no unambiguous definition covering every field. In other words, *Digital Twin* can mean different things for different people with different interests, expertises, and capabilities. Next, we will give the functionality of a Digital Twin, supported by the literature. Based on the reviewed literature and definitions from practice, we give our interpretation, which we will use in this thesis when referred to Digital Twin. We combine the knowledge to discover how the concept of Digital Twin can benefit the IAC.

6.2.1 General Introduction

As mentioned in Section 1.3.2, the Digital Twin (DT) is a realistic model of the system, used for simulation, optimization, and control (Cronrath et al., 2019). A Digital Twin is the ability to take a virtual representation of the elements and dynamics of how an Internet of Things (IoT) device operates, works, and lives throughout its life cycle (O'Connor, 2017). Therefore, it has to understand all of its dynamics. The Digital Twin learns from *reality*, *use* and *design* (Grieves & Vickers, 2017).

Currently, Digital Twins are often used for prototyping, factory layouts, and manufacturing simulations. As stated in the literature, the Twin's purpose is to support decision-making by understanding, learning, and reasoning for decision-makers (Popkov, 2019). Furthermore, the Twin should analyze its validity in different cases to improve its effectiveness (Marmolejo-Saucedo, 2020). In the application of supply chains, the DT learns to understand and monitor the supply chain's behavior, predict non-regular situations, and plan actions. Additionally, as in our problem statement, a control tower approach benefits from service and inventory reduction. The reduction is due to the Digital Twin Supply Chain (DTSC) explores the potential beyond the (current) equipment and factory boundaries (Srai, Settanni, Tsolakis, & Aulakh, 2019; Park, Son, & Noh, 2020).

Altogether, a Digital Twin is the virtual representation or simulation of a real-world environment. The Twin helps to improve the effectiveness of processes by understanding the dynamics and elements of the virtual-environment. Furthermore, the DTSC determines what decision-making is required for different scenario inputs (alerts), which is beneficial to service and inventory levels.

6.2.2 Functionality of the Twin

A Digital Twin is based on an accurate simulation model. However, there are some criteria that a Digital Twin requires. First of all, the Digital Twin of the Supply Chain should be able to collect and analyze the supply chain interaction (e.g., changes in demand) and enable predictions of financial flows and scenario testing (Popkov, 2019; Barykin, Bochkarev, Kalinina, & Yadykin, 2020). By synchronizing the information between virtual- and physical worlds, lead times can be reduced significantly (Y. Wang, Wang, & Liu, 2020). Additionally, from this paper, we conclude that the dynamic and comprehensive data collection improves the forecast accuracy. The virtual models allow us to develop action plans and tackle detrimental situations.

Nonetheless, not only providing virtual data is essential for a Digital Twin. The Twin should additionally learn from reality input, and feedback from use (Grieves & Vickers, 2017). Finally, Popkov (2019) states that we can integrate a Supply Chain Digital Twin into a service control tower, such we can directly improve the *Information manpower layer*, as discussed in Section 3.1.

6.2.3 Digital Twin Applied to the Independent Aerospace Company

As mentioned in the section's introduction, Digital Twins' definition differs in each field and per person. Based on the literature, current practice, and functionality, we define our understanding of the concept of Digital Twins. In this thesis, the term Digital Twin will describe a virtual representation or simulation model of a real-world entity or system that uses real-time data to predict the dynamics. Furthermore, the Twin includes understanding, learning, and reasoning within a two-way interaction between the real and virtual worlds to pursue continuous improvement. We explain the two-way interaction in the IAC's interest further in this subsection.

The virtual environment should behave and react comparably with the real world to create a feasible test-environment, i.e., realistic scenarios should be simulated to advise optimal decision-making. As discussed, the Twin should learn directly from the inputs, as well as from the feedback from use. For the IAC's objective, we use the Internet of Things (IoT) to align the real and virtual worlds. Additionally, the virtual model within the DT can change both in real-time as during the operation. Therefore, a DT consists of connected products, using the IoT and digital threads (Madni, Madni, & Lucero, 2019). These digital threads provide connectivity throughout the system's lifecycle within the physical to the Digital Twin's updating process. The connectivity within a DTSC leads to a better synchronization of virtual- and physical worlds, where lead times and forecasts can be improved significantly.

For the IAC's objective, the reinforcement learning environment represents the virtual world of the DT. In this way, the impact of the situations (or alerts) on operational decision-making can be validated due to the virtual world that shows the interventions' impact, allowing continuous improvement. The alignment between virtual- and real-world should be checked with tactical planning decision-making due to the operational decisions. Potential adjustments and feedback can be provided to the Digital Twin for improvement. For clarification, we use Figure 6.1. This figure shows that the alert generation tool processes the data from the real world. Based on the alert priorities, the DSS provides information about the alert's optimal intervention. Subsequently, the real environment provides the DSS with feedback on the impact of the decision on tactical level, such that the real- and virtual worlds will learn and improve.

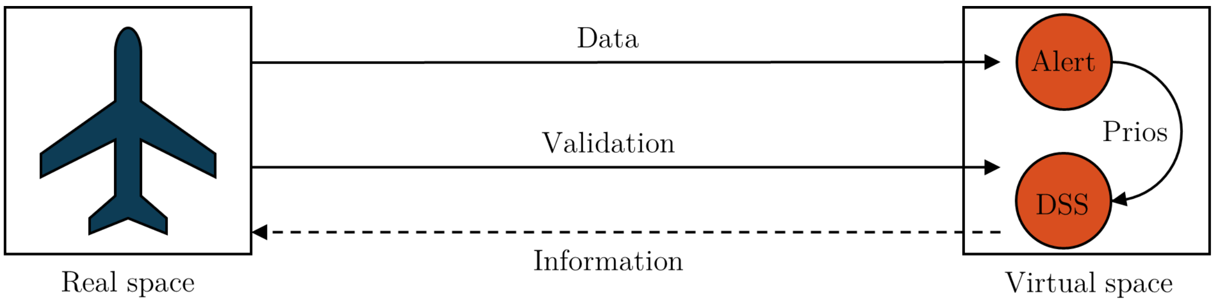


Figure 6.1: Visualization of Digital Twin concept applied to problem statement

A recent case study shows a different potential of using Digital Twins within the aerospace industry (Lutters & Damgrave, 2019). The purpose of the Digital Twin of this research is to identify and track and trace components. Combining the components' status and the inventory information, both the alerts and the decision-making can provide a more preventive approach. Following Lutters and Damgrave (2019) the ERP data and strategic insights on order processing and portfolio selection comes together.

In conclusion, a DT can be suitable for the IAC's use since visibility leads to better insight and control in cost behavior of interventions. This can help understand and monitor the supply chain's behavior by continuous improvement. With the alert generation and Decision Support System (DSS), we can construct a virtual-environment that supports decision-making and therefore learns from reality.

6.3 Conclusion

This chapter aims to review the possibility of implementing the DSS in a more extensive process while using a Digital Twin. Therefore, we reviewed the possible scalability issues and the requirements for a Digital Twin.

Scalability This chapter has discussed two major scalability issues for larger problem instances: an increasing state- and action space, and big data. In case the state- and action space will increase, a tabular solving method will not be efficient enough anymore. By Value Function Approximation (VFA), we can overcome this problem. Be aware that only methods that support nonstationarity are suitable for RL. For problem instances where VFA is not sufficient, Deep Reinforcement Learning can help developing learning and generalization features.

Naturally, by including more data in the problem instance, solving can be more problematic. Approximating the optimal solution by combining sub-optimal policies requires lots of computational power. Therefore, more advanced methods as Imitation Learning are required. We did not review the applicability for the IAC's purpose within this thesis, and therefore we recommend it for future research.

Digital Twins The virtual representation or simulation of the real-world environment helps in improving the effectiveness of procedures. The interaction between large data-sets of the environment and information about the real-world lead to an improvement of planning and management (Söderberg, Wärmefjord, Carlson, & Lindkvist, 2017; Uhlemann, Lehmann, & Steinhilper, 2017). The Digital Twin requires a two-way interaction consisting of three components: the real-world input towards the Twin, output, or processed data from the Twin, and the feedback from the environment on the output. As the Digital Twin aims for continuous improvement, the Twin learns from reality and use.

We conclude that the IAC's alert generation model can function well in processing input from the physical world. The model included in this research's scope can be responsible for the intervention or action management's output. We conclude that the desired model can be very functional within the Digital Twin interaction for the IAC. However, for the Twin's validity, we need feedback from tactical planning to create a properly functioning Digital Twin with the aimed model. Additionally, we see an immense potential of a DT in collaboration with AI. A combination of the two can lead to preventive decision-making and health monitoring of operational planning. Therefore, we advise to research and validate the feedback relationship and AI in further research.

7 | Conclusions, Discussion, and Recommendations

This final chapter includes the conclusion, discussion, and recommendations of this research for Independent Aerospace Company (IAC). In Section 7.1 we provide a conclusion for this research, followed by a discussion in Section 7.2. Finally, in Section 7.3, the recommendations for our findings and future research are given.

7.1 Conclusion

The objective of this research is defined as follows:

'In what way and to what extent can a reinforcement learning algorithm improve operational decision-making while incorporating long-term yields?'

To create insight into the supply chain, current decision-making process, and performance, we identify the current situation of the IAC. Based on an interview with the Operational Planning Professionals (OPP), we conclude that currently decisions are made based on gut-feeling. Since the OPPs do not use any measurement tools, we cannot express the current performance well. However, we find a need for a Decision Support System (DSS) to create better insight into the impact of decision-making.

Subsequently, in the literature, we find no additional interventions that are suitable in practice for the IAC. Also, the use of reinforcement learning algorithms in a Service Control Tower (SCT) of spare part management is not available in the literature. Nonetheless, to define AI's potential, we review different approaches of DSS in the literature. Because of the immense potential of AI in decision support systems, we reviewed six different reinforcement learning algorithms. Based on the simplicity of implementation, promising results in the literature, and the applicability of small instance problems, we conclude that Dyna-Q shows the largest potential to our experimental setting.

To analyze the Dyna-Q performance, we compared this algorithm with an exact solving method of a stochastic dynamic programming framework and a greedy heuristic. We solve the SDP with a backward recursion to find the optimal solution. This exact solving method works for small instances, such as our problem. In case the problem instance grows, an exact approach is no longer possible due to the *Curse of Dimensionality.* We use the exact approach in this setting as the initial scenario value. Next, we compare the SDP with a greedy heuristic. This heuristic always chooses the optimal action for the current state while ignoring the cost-to-go function. After learning the policy by Dyna-Q, we evaluate all three solution approaches by a simulation.

To test the three presented algorithms properly, we create three scenarios where we prefer tactical, operational, or an equilibrium in decisions. As expected, we see that the backward recursion performs best for all three scenarios. Further, we see that the greedy heuristic does not undertake much action because it ignores the cost-to-go function. In other words, the greedy algorithm does nothing unless it has to. This decision-strategy works for straight-forward cases, but not when we start with more backorders. We see Dyna-Q performing better in the more challenging scenarios than the greedy algorithm. However, in the straight-forward cases in operational and tactical preference, we see that Dyna-Q uses a more conservative approach; costs are higher, but backorders are way lower. Finally, we see that Dyna-Q outperforms the greedy heuristic in all situations for the equilibrium point and even performs close to optimal. The greedy algorithm shows in this scenario that it has difficulties deciding correctly. Therefore, we conclude that reinforcement learning can be of great benefit for decision-making in challenging scenarios. Furthermore, by moving the importance of decisions to an equilibrium point, we see a delicate balance between costs and backorders.

The solving approach, as used in this research, is suitable for small problem instances. The IAC's purpose is to use the DSS for multi-item problem instances. Therefore, some scalability issues can occur. In case the state- and action space will increase, a tabular solving method will not be efficient enough. Using a nonstationary Value Function Approximation (VFA) or Deep Neural Network (DNN), we can overcome this problem. The second scalability issue that we reviewed is learning with big data. Approximating the optimal solution by combining sub-optimal policies requires lots of computational power. Therefore, advanced methods, such as imitation learning or hierarchical reinforcement learning, are required for solving these problem instances.

Implementing the DSS combined with an alert generation tool in a Digital Twin brings ERP data and, e.g., strategic insights on order processing together. Within the Twin, two-way interaction between the real and virtual worlds is crucial for aligning the two worlds and therefore enabling continuous improvement. This can help understand and monitor the supply chain's behavior, clarifying the relationship between the operational and tactical level.

In conclusion, a reinforcement learning algorithm, such as Dyna-Q, can benefit operational decision-making by creating insight into the operational planning horizon's expected costs. Furthermore, because of the adaptive learning abilities of planning and learning algorithms, stochasticity in demand, returning repairs, and returning customer components can be processed without performance loss. By including an equilibrium point between tactical and operational limitations in the decision-making process, reinforcement learning can be very beneficial for both costs and availability.

7.2 Discussion

Within this research, we made multiple assumptions for the simplicity of the model. Therefore, we will discuss the assumptions and limitations of the current research approach.

The first point of discussion is our assumption for time-based decision-making. For modeling simplicity, we have chosen to make decisions during a planning period of 14 days. Although the planning horizon is representative for the actual decision-making process, the results are very volatile. In other words, the expected episode costs can be zero, while the costs in other episodes are approximately \$10.000. The small probability of demand occurring causes this effect within the current experimental design. Since the episodes without any event are not very interesting for the DSS, the algorithm's full potential might not be used. By using way longer episodes of, e.g., five years, we can overcome this problem. Another suitable approach in these situations is event-based. However, this method gives a troublesome modeling approach because the state behavior is challenging to model in a changing time interval scenario. Therefore, we assumed the time-based approach to be sufficient.

Second, in order to define the decision space of the SDP, we assumed that lead times for the decisions were zero time periods. This means that the consequences of a decision are directly visible. There might be a longer lead time than zero time periods, or even a stochastic lead time in the real planning scenario. We expect a slightly more conservative approach in decision-making in such scenarios.

Although we modeled our state-space relatively strict and straightforward, we see that Dyna-Q is having difficulties solving the problem instance. Currently, we solved the SDP framework for a single starting state separately with decent results. However, when combining all possible states into one learning model, we saw that Dyna-Q has some converging issues. Even in a simulation with 500.000 episodes of learning, we could not get comparable results with the greedy heuristic. Literature states that the fact that both approaches contribute to the same value function estimate causes this problem. Therefore, the model-free process can slow down the learning process. Additionally, in sizeable stochastic problem instances, the model's observations might be insufficient, leading to an incorrect and computational inefficient policy computation (Sutton & Barto, 2018; Z. Ding et al., 2020).

The next assumption concerns the backorder counter. For the backward recursion and simulation models we used a different backorder counter, because it is difficult to express a backorder counter in a backward recursion. This difference leads to a wrong estimate of the exact value computed by the backward recursion. Therefore, our evaluation simulation with the optimal policy gives a better representation of the direct quality of all methods. Nonetheless, when the initial estimate is wrong, the policy from the backward recursion might not be optimal. Note that the backorder counter is a simple assumption which might not be representative for the IAC's desire or behavior.

The final assumption we discuss concerns the boundaries of the state space. For reasons of simplicity, we made assumptions to keep the state space relatively small. However, we see some odd behavior in the states located at the edges of the state space due to the boundary assumptions. For example, the SDP does nothing when located at the maximum backorder state, because no extra backorders can occur. For the approximation of the true value of these states, it is better to enlarge the state space. Nonetheless, the states included in the sensitivity analysis are unlikely to change due to the small probability of transitioning towards one of the boundary states.

7.3 Recommendations

Within this final section, we will review our recommendations concerning operational decision-making. We recommend the following points for future research:

- **Event-Based Decision-Making:** As mentioned in the discussion, we expect the learning abilities for the reinforcement learning algorithm to improve when using an event-based decision-making approach instead of a time-based. Furthermore, the event-based approach will also cause the results to be less deviating from the mean.

- **Long Run Evaluation:** In our current solving approach, we try to find the optimal decision for each particular day while incorporating future states and decisions. Here, to properly include long term costs, we determine tactical impact by an analytical approach. We can determine the long-term relationship by creating a long-term simulation over, e.g., five years. Because our exact and Dyna-Q approaches learn the best decisions for time period t onward, we can say that the policy learned for $t = 1$ includes the whole planning horizon. Therefore, we can use the policy of $t = 1$ for each independent day in the long-term simulation. As a result, the tactical costs, now analytically included in the cost function, can be approached by a simulation.

- **Closed Loop Inventory Planning:** In the current literature, there are multiple sources available concerning decision support systems for spare part management. However, none of these sources include a (semi-) closed loop, meaning that the current literature approaches all assume that the on-hand stock will never return to the inventory pool. For a service provider with a spare part pooling system, such as the IAC, the decision-making differs from an open system. Further research should provide a better insight into the effect of the returning components from customers to decision-making. Note that in the current experimental setting we assumed the probability of components returning from the customer to be zero. In future research the expected customer returns should be incorporated, dependent of the planning horizon.
- **Multi-Echelon:** Without loss of generality, we assumed that a single-echelon approach would be sufficient for the IAC's purpose. However, by including more echelon levels the model can be generalized. Therefore, the decision support system would be applicable to more different company structures. This means that the DSS would significantly impact the current gap in the literature.
- **Solution Approaches:** We used a greedy algorithm as a benchmark for the RL algorithm for the current solution approach. As expected, the greedy algorithm did not perform well in all scenarios. Therefore, we recommend testing more advanced heuristics. Although the greedy heuristic is not suitable for this purpose, other non-learning heuristics might be sufficient as well. Naturally, testing on larger state and action spaces is preferable. Further, we recommend testing the performance on larger state- and action spaces for the planning and learning algorithms. Therefore, using function approximation or deep neural networks is suggested. Based on the reviewed papers in the literature study, we expect the most potential for the IAC's purpose in Actor-Critic (AC) methods. From Schulman (2016) we find that AC-methods are located in the intersection of policy and value based methods; we learn both policy and value function. Especially A3C shows strong results for large problems while being a computationally efficient solving method. Another alternative for the IAC's purpose is Proximal Policy Optimization (PPO). This algorithm has proven to be effective in optimal control for inventory and service control tower problems (Meisheri et al., 2020; Vanvuchelen et al., 2020). Furthermore, PPO is relatively simple to implement and it provides robust and reliable results
- **Incorporating Tactical Planning:** Within this research, we are investigating operational decision-making while incorporating the tactical level as a guideline. As mentioned, we do not find DSSs that incorporate tactical decisions in the closed-loop spare part management from the current literature. However, within this research, we propose a first interaction between operational and tactical decisions. For further research, we recommend investigating the possibilities of including the reversed interaction of operational and tactical decisions in a decision support system. With this two-way interaction, both operational and tactical planning can improve.
- **Reinforcement Learning in Service Control Tower Settings:** Our final recommendation is to further investigate the potential of reinforcement learning within a Service Control Tower (SCT) environment. Based on the current literature study, we encountered one paper concerning a joint replenishment problem using PPO (Vanvuchelen et al., 2020). The paper concludes and emphasizes the potential of (deep) reinforcement learning algorithms for SCT settings. Including a reinforcement learning algorithm into the SCT leads to a more effective and efficient logistics network.

References

Abdul-Malak, D. T., Kharoufeh, J. P., & Maillart, L. M. (2019). Maintaining systems with heterogeneous spare parts. *Naval Research Logistics*, *66*(6), 485–501.

Accenture. (2015). *Creating a supply chain control tower in the high-tech industry.* Retrieved from `https://www.accenture.com/{_}acnmedia/Accenture/Conversion-Assets/DotCom/Documents/Global/PDF/Industries{_}19/Accenture-Supply-Chain-Control-Tower.pdf`

Amaro, A. C., & Barbosa-Póvoa, A. P. (2009). The effect of uncertainty on the optimal closed-loop supply chain planning under different partnerships structure. *Computers and Chemical Engineering*, *33*(12), 2144–2158. doi: 10.1016/j.compchemeng.2009.06.003

Åström, K. J. (1965). Optimal control of markov processes with incomplete state information i. *Journal of Mathematical Analysis and Applications*, *10*, 174–205. Retrieved from `https://lup.lub.lu.se/search/ws/files/5323668/8867085.pdf` doi: 10.1016/0022-247X(65)90154-X

Barut, M., & Sridharan, V. (2005). Revenue Management in Order-Driven Production Systems. *Decision Sciences*, *36*(2), 6.

Barykin, S. Y., Bochkarev, A. A., Kalinina, O. V., & Yadykin, V. K. (2020). Concept for a supply chain digital twin. *International Journal of Mathematical, Engineering and Management Sciences*, *5*, 1498–1515. Retrieved from `https://doi.org/10.33889/IJMEMS.2020.5.6.111`

Bellman, R. (1957). *Dynamic programming.* Princeton: Princeton University Press.

Berger-Tal, O., Nathan, J., Meron, E., & Saltz, D. (2014). The exploration-exploitation dilemma: a multidisciplinary framework. *PloS one*, *9*(4), e95693.

Bertsekas, D. P. (2017). *Dynamic programming and optimal control* (Fourth ed.). Athena Scientific.

Bertsekas, D. P. (2019). *Reinforcement learning and optimal control* (First ed.). Athena Scientific 2019.

Black, P. E. (2005, February 2). *Greedy algorithm.* Retrieved from `https://www.nist.gov/dads/HTML/greedyalgo.html`

Blanchard, D. (2011). *Supply Chain Management: Best Practices* (Second ed.). New Jersey: John Wiley & Sons, Inc. doi: 10.1093/acprof:oso/9780195138108.003.0010

Boddy, D., & Paton, S. (2011). *Management: An Introduction* (Fifth ed.). Pearson Education Limited.

Buşoniu, L., De Schutter, B., & Babuška, R. (2010). Approximate dynamic programming and reinforcement learning. In R. Babuška & F. C. A. Groen (Eds.), *Interactive collaborative information systems* (pp. 3–44). Berlin, Heidelberg: Springer Berlin Heidelberg. Retrieved from `https://doi.org/10.1007/978-3-642-11688-9_1` doi: 10.1007/978-3-642-11688-9_1

Cheng, L., Subrahmanian, E., & Westerberg, A. W. (2004). A comparison of optimal control and stochastic programming from a formulation and computation perspective. *Computers and Chemical Engineering*, *29*(1), 149–164. doi: 10.1016/j.compchemeng.2004.07.030

Cooper, D. R., & Schindler, P. S. (2011). *Business Research Methods* (Eleventh ed.). McGraw-Hill Education.

Cronrath, C., Aderiani, A. R., & Lennartson, B. (2019). Enhancing digital twins through reinforcement learning. In *2019 ieee 15th international conference on automation science and engineering (case)* (p. 293-298).

Croonenborghs, T., Ramon, J., Blockeel, H., & Bruynooghe, M. (2007). Online learning and exploiting relational models in reinforcement learning. In *Proceedings of the 20th international joint conference on artifical intelligence* (p. 726–731). San Francisco, CA, USA: Morgan Kaufmann Publishers Inc.

Curley, R. (2016, June 17). *Interchangeable parts.* https://www.britannica.com/technology/interchangeable-parts. Encyclopaedia Britannica inc.

Dargam, F., Perz, E., Bergmann, S., Rodionova, E., Sousa, P., Souza, F. A. A., ... Bonachela, P. Z. (2020). Supporting operational decisions on desalination plants from process modelling and simulation to monitoring and automated control with machine learning. In J. M. Moreno-Jiménez, I. Linden, F. Dargam, & U. Jayawickrama (Eds.), *Decision support systems x: Cognitive decision support systems and technologies* (Sixth ed., pp. 150 – 164). Zaragoza, Spain: Springer International Publishing. doi: https://doi.org/10.1007/978-3-030-46224-6

Davenport, T. H., & Harris, J. G. (2005). Automated decision making comes of age. *MIT Sloan Management Review*, *46*(4), 83–89.

DeepMind. (2019, June 18). *Alphago.* Retrieved from `https://www.deepmind.com/research/case-studies/alphago-the-story-so-far`

Deloitte. (2019). *Supply Chain Control Tower: The information link for Operations across the Live Enterprise.* Retrieved from `https://www2.deloitte.com/content/dam/Deloitte/de/Documents/operations/10-19-supply-chain-control-tower.pdf`

Dhande, M. (2020). *What is the difference between ai, machine learning and deep learning.* Retrieved from `https://www.geospatialworld.net/blogs/difference-between-ai%EF%BB%BF-machine-learning-and-deep-learning/` ([Online; accessed September 15, 2020])

Ding, H., Benyoucef, L., & Xiaolan, X. (2008). Simulation-based evolutionary multi-objective optimisation approach for integrated decision-making in supplier selection. *International Journal of Computer Applications in Technology*, *31*(3-4), 144–157. doi: 10.1504/IJCAT.2008.018153

Ding, Z., Huang, Y., Yuan, H., & Dong, H. (2020). Introduction to Reinforcement Learning. In H. Dong, Z. Ding, & S. Zhang (Eds.), *Deep reinforcement learning: Fundamentals, research and applications* (pp. 47–123). Singapore: Springer Singapore. Retrieved from `https://doi.org/10.1007/978-981-15-4095-0{_}2` doi: 10.1007/978-981-15-4095-0_2

Do Prado, J. C., & Qiao, W. (2019). A Stochastic Decision-Making MOdel for an Electricity Retailer With Intermittent Renewable Energy and Short-Term Demand Response. *IEEE Transactions on Smart Grid*, *10*(3), 2581–2592. doi: 10.1109/TSG.2018.2805326

Elmaraghy, H. A., & Majety, R. (2008). Integrated supply chain design using multi-criteria optimization. *International Journal of Advanced Manufacturing Technology*, *37*(3-4), 371–399. doi: 10.1007/s00170-007-0974-3

Faggella, D. (2020, February 26). *What is machine learning?* Emerj. Retrieved from `https://emerj.com/ai-glossary-terms/what-is-machine-learning/`

Galasso, F., Mercé, C., & Grabot, B. (2008). Decision Support for Supply Chain Planning Under Uncertainty. *International Journal of Systems Science*, *39*(7), 667–675. doi: 10.1080/00207720802090765

García-Alvarado, M., Paquet, M., & Chaabane, A. (2015). On inventory control of product recovery systems subject to environmental mechanisms. *International Journal of Production*

Economics, *165*, 132–144. doi: 10.1016/j.ijpe.2015.01.009

Gijsbrechts, J., Boute, R. N., Van Mieghem, J. A., & Zhang, D. (2019). Can deep reinforcement learning improve inventory management? performance on dual sourcing, lost sales and multi-echelon problems. *Performance on Dual Sourcing, Lost Sales and Multi-Echelon Problems (July 29, 2019).*

Grieves, M., & Vickers, J. (2017). Digital twin: Mitigating unpredictable, undesirable emergent behavior in complex systems. In *Transdisciplinary perspectives on complex systems* (pp. 85–113). Springer.

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

Heess, N., TB, D., Sriram, S., Lemmon, J., Merel, J., Wayne, G., ... Silver, D. (2017). Emergence of locomotion behaviours in rich environments. *CoRR*, *abs/1707.02286*. Retrieved from `http://arxiv.org/abs/1707.02286`

Heidrich-Meisner, V., Lauer, M., Igel, C., & Riedmiller, M. (2007). Reinforcement Learning in a Nutshell. *ESANN 2007 Proceedings - 15th European Symposium on Artificial Neural Networks*, 277–288.

Henkelmann, R. (2018). *A deep learning based approach for automotive spare part demand forecasting* (Unpublished master's thesis). Otto-von-Guericke Universität Magdeburg, Magdeburg.

Horsch, M. C., & Poole, D. L. (2013). An anytime algorithm for decision making under uncertainty. *CoRR*, *abs/1301.7384*. Retrieved from `http://arxiv.org/abs/1301.7384`

Hosokawa, S., Kato, J., & Nakano, K. (2014). A reward allocation method for reinforcement learning in stabilizing control tasks. *Artif Life Robotics*, 109–114. doi: 10.1007/s10015-014-0146-0

Hu, Q., Boylan, J. E., Chen, H., & Labib, A. (2018). OR in spare parts management: A review. *European Journal of Operations Research*, *266*, 395–414.

Huang, Y. (2020). Deep Q-Networks. In H. Dong, Z. Ding, & S. Zhang (Eds.), *Deep reinforcement learning: Fundamentals, research and applications* (pp. 135–160). Singapore: Springer Singapore. Retrieved from `https://doi.org/10.1007/978-981-15-4095-0{_}2` doi: 10.1007/978-981-15-4095-0_2

Huys, Q. J. M., Cruickshank, A., & Seriès, P. (2014). Encyclopedia of Computational Neuroscience. *Encyclopedia of Computational Neuroscience*, 1–10. doi: 10.1007/978-1-4614-7320-6_674-1

Isbell, C. L. (1992). *Explorations of the practical issues of learning prediction-control tasks using temporal difference learning methods* (Unpublished master's thesis). Massachusetts Institue of Technology, Cambridge, MA.

Jin, Y., Pipe, T., & Winfield, A. (1997). Stable manipulator trajectory control using neural networks. In O. Omidvar & P. van der Smagt (Eds.), *Neural systems for robotics* (p. 117 - 151). Boston: Academic Press. doi: https://doi.org/10.1016/B978-0-08-092509-7.50009-9

Jokinen, H., Konkarikoski, K., Pulkkinen, P., & Ritala, R. (2009). Operations' decision making under uncertainty: Case studies on papermaking. *Mathematical and Computer Modelling of Dynamical Systems*, *15*(5), 435–452. doi: 10.1080/13873950903375429

Kaelbling, L. P., Littman, M. L., & Cassandra, A. R. (1998). Planning and acting in partially observable stochastic domains. *Artificial Intelligence*(101), 99–134.

Kaelbling, L. P., Littman, M. L., & Moore, A. W. (1996). Reinforcement learning: A survey. *Journal of Artificial Intelligence Research*(4), 237–285.

Kara, A., & Dogan, I. (2018). Reinforcement learning approaches for specifying ordering policies of perishable inventory systems. *Expert Systems with Applications*, *91*, 150–158.

Khalifa, D., Hottenstein, M., & Aggarwal, S. (1977). Cannibalization Policies for Multistate Systems. *Operations Research*, *25*(6), 1032–1039.

Kilpi, J., & Vepsäläinen, A. P. (2004). Pooling of Spare Components Between Airlines. *Journal of Air Transport Management*, *10*, 137 – 146. doi: 10.1016/j.jairtraman.2003.09.001

Kim, C. O., Kwon, I.-H., & Baek, J.-G. (2008). Asynchronous action-reward learning for nonstationary serial supply chain inventory control. *Applied Intelligence*, *28*(1), 1–16.

Kirk, D. E. (2004). *Optimal control theory: An introduction* (13th ed.). Dover Publications, Inc.

Kristensen, J. T., & Burelli, P. (2020). *Strategies for using proximal policy optimization in mobile puzzle games.* (To be published in 2020 Foundations of Digital Games conference) doi: 10.1145/3402942.3402944

Law, A. M. (2015). *Simulation modeling and analysis* (Fifth ed.). McGraw-Hill Education.

Lutters, E., & Damgrave, R. (2019). The development of pilot production environments based on digital twins and virtual dashboards. *Procedia CIRP*, *84*, 94–99.

Madni, A. M., Madni, C. C., & Lucero, S. D. (2019). Leveraging digital twin technology in model-based systems engineering. *Systems*, *7*(1), 7.

Marmolejo-Saucedo, J. A. (2020). Design and development of digital twins: a case study in supply chains. *Mobile Networks and Applications*, 1.

Mätäsniemi, T. (2008). *Operational decision making in the process industry: Multidisciplinary approach* (A. Repo, Ed.). Helsinki: VTT Research Notes.

Meisheri, H., Baniwal, V., Sultana, N. N., Khadilkar, H., & Ravindran, B. (2020). *Using reinforcement learning for a large variable-dimensional inventory management problem.*

Mes, M. (2019, May 7). *Operations Research Techniques 2: Stochastic Optimization & Learning [Slides].* University of Twente. Retrieved from `https://canvas.utwente.nl/courses/3143/pages/lecture-slides?module_item_id=80322` (Slides are not publicly available)

Mes, M., & Perez Rivera, A. (2017, March 11). Approximate dynamic programming by practical examples. In R. Boucherie & N. van Dijk (Eds.), *Markov decision processes in practice* (pp. 63–101). Springer. doi: 10.1007/978-3-319-47766-4_3

Mes, M., & van Heeswijk, W. (2020). *Comparison of manual and automated decision-making with a logistics serious game.* (Working paper: Submitted to ICCL2020)

Mnih, V., Badia, A. P., Mirza, M., Graves, A., Lillicrap, T., Harley, T., ... Kavukcuoglu, K. (2016). Asynchronous methods for deep reinforcement learning. In *International conference on machine learning* (pp. 1928–1937).

Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., & Riedmiller, M. (2013). Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*.

Murtaugh, M., & Gladwin, H. (1980). A hierarchical decision-process model for forecasting automobile type-choice. *Transportation Research, Part A: General*, *14 A*(5-6), 337–347. doi: 10.1016/0191-2607(80)90053-9

Nadj, M., Maedche, A., & Schieder, C. (2020). The effect of interactive analytical dashboard features on situation awareness and task performance. *Decision Support Systems*(August 2019). doi: 10.1016/j.dss.2020.113322

Nanduri, V., & Kazemzadeh, N. (2012). Economic impact assessment and operational decision making in emission and transmission constrained electricity markets. *Applied Energy*, *96*, 212–221. Retrieved from `http://dx.doi.org/10.1016/j.apenergy.2011.12.012` doi: 10.1016/j.apenergy.2011.12.012

Nascimento, J., & Powell, W. B. (2013). An optimal approximate dynamic programming algorithm for concave, scalar storage problems with vector-valued controls. In *Ieee transactions on automatic control* (Vol. 58, pp. 2995–3010). IEEE.

Nozhati, S., Sarkale, Y., Ellingwood, B., K.P. Chong, E., & Mahmoud, H. (2019). Near-optimal planning using approximate dynamic programming to enhance post-hazard community resilience management. *Reliability Engineering & System Safety*, *181*, 116 - 126. Retrieved from `http://www.sciencedirect.com/science/article/pii/S0951832018305180` doi:

https://doi.org/10.1016/j.ress.2018.09.011

O'Connor, C. (2017, February 16). *Introduction to Digital Twin: Simple but detailed [Slides].* Munich, Germany: IBM Internet of Things. Retrieved from `https://www.slideshare.net/IBMIoT/ibm-watson-internet-of-things-introducing-digital-twin`

Oroojlooyjadid, A., Nazari, M. R., Snyder, L. V., & Takác, M. (2017). A deep q-network for the beer game with partial information. *CoRR*, *abs/1708.05924*.

Ou, X., Chang, Q., & Chakraborty, N. (2020). A method integrating q-learning with approximate dynamic programming for gantry work cell scheduling. *IEEE Transactions on Automation Science and Engineering*.

Park, K. T., Son, Y. H., & Noh, S. D. (2020). The architectural framework of a cyber physical logistics system for digital-twin-based supply chain control. *International Journal of Production Research*, 1–22.

Peng, B., Li, X., Gao, J., Liu, J., Wong, K.-F., & Su, S.-Y. (2018). Deep dyna-q: Integrating planning for task-completion dialogue policy learning. *arXiv preprint arXiv:1801.06176*.

Pomerol, J.-C. (1997). Artificial Intelligence and Human Decision Making. *European Journal of Operational Research*, *99*(1), 3–25. Retrieved from `https://doi.org/10.1016/S0377-2217(96)00378-5` doi: 10.1016/j.compind.2020.103239

Popkov, T. (2019). *Design Your Supply Chain: Run with Digital Twin [WEBINAR].* anyLogistix: supply chain software. Retrieved from `https://www.anylogistix.com/upload/pdf/suply-chain-digital-twin-and-control-tower-webinar.pdf`

Powell, W. B. (2009). What you should know about approximate dynamic programming. *Naval Research Logistics*, *56*, 239–249. doi: 10.1002/nav.20347

Powell, W. B. (2010). Merging ai and or to solve high-dimensional stochastic optimization problems using approximate dynamic programming. *INFORMS Journal on Computing*, *22*, 2–17.

Ra, J. W. (1991). Hierarchical Decision Process. In *Proceedings of the 1991 portland international conference on management of engineering and technology - picmet '91* (pp. 595–599). Portland, OR, United States: IEEE,Piscataway, NJ, United States.

Ray, T. (2020). *Best masters programs in data science and big data analytics in europe – part 1.* Retrieved from `https://www.stoodnt.com/blog/best-masters-programs-in-data-science-big-data-analytics-in-europe-part-1/` ([Online; accessed September 15, 2020])

Rosen, R., Von Wichert, G., Lo, G., & Bettenhausen, K. D. (2015). About the importance of autonomy and digital twins for the future of manufacturing. *IFAC-PapersOnLine*, *48*(3), 567–572. Retrieved from `http://dx.doi.org/10.1016/j.ifacol.2015.06.141` doi: 10.1016/j.ifacol.2015.06.141

Rossi, R. (2013). On service level measures in stochastic inventory control. In *Ifac proceedings volumes (ifac-papersonline)* (Vol. 46, pp. 1991–1996). Saint Petersburg: IFAC. Retrieved from `http://dx.doi.org/10.3182/20130619-3-RU-3018.00295` doi: 10.3182/20130619-3-RU-3018.00295

Russell, S., & Norvig, P. (2010). *Artificial intelligence: A modern approach* (Third ed.). Prentice Hall.

Ryzhov, I. O., & Powell, W. B. (2011). Bayesian active learning with basis functions. In *2011 ieee symposium on adaptive dynamic programming and reinforcement learning (adprl)* (p. 143-150).

Schmidhuber, J. (2015). Deep learning in neural networks: An overview. *Neural Networks*, *61*, 85–117.

Schulman, J. (2016). *Optimizing expectations: From deep reinforcement learning to stochastic computation graphs* (Unpublished doctoral dissertation). UC Berkeley.

Schulman, J., Filip, W., Dhariwal, P., Radford, A., & Klimov, O. (2017). Proximal policy optimization algorithms. *CoRR*, *abs/1707.06347*. Retrieved from `http://arxiv.org/`

abs/1707.06347

Schulman, J., Levine, S., Moritz, P., Jordan, M. I., & Abbeel, P. (2015). Trust region policy optimization. *CoRR*, *abs/1502.05477*.

Sharad, S., Nitin, D., Styavan, D., Mitul, C., & Shrivastava, S. (2011). Interchangeability of Multisource Pharmaceutical Product: A Review. *International Research Journal of Pharmacy*, *6*(2), 1–10.

Sherbrooke, C. C. (2004). *Optimal Inventory Modeling of Systems* (Second ed.; F. S. Hillier, Ed.). Boston: Kluwer Academic Publishers.

Shou-Wen, J., Ying, T., & Yang-Hua, G. (2013). Study on Supply Chain Information Control Tower System. *Information Technology Journal*, *12*(24), 8488–8493. Retrieved from https://scialert.net/abstract/?doi=itj.2013.8488.8493 doi: 10.3923/itj.2013.8488.8493

Silver, D. (2015). *Advanced Topics: Reinforcement Learning [Slides].* University College London. Retrieved from https://www.davidsilver.uk/teaching/

Silver, D., Huang, A., Maddison, C., Guez, A., Sifre, L., Van Den Driessche, G., . . . Hassabis, D. (2016). Mastering the game of go with deep neural networks and tree search. *Nature*, *529*(7587), 484-489. doi: 10.1038/nature16961

Simao, H., & Powell, W. B. (2009). Approximate dynamic programming for management of high-value spare parts. *Journal of Manufacturing Technology*, *20*, 147–160. doi: 10.1108/17410380910929592

Skoglund, A., Palm, R., & Duckett, T. (2005). Towards a supervised dyna-q application on a robotic manipulator. *Advances in Artificial Intelligence in Sweden*, 148–153.

Smallwood, R. D., & Sondik, E. J. (1973). The optimal control of partially observable markov processes over a finite horizon. *Operations Research*, *21*(5), 1071-1088.

Söderberg, R., Wärmefjord, K., Carlson, J. S., & Lindkvist, L. (2017). Toward a digital twin for real-time geometry assurance in individualized production. *CIRP Annals*, *66*(1), 137–140.

Srai, J., Settanni, E., Tsolakis, N., & Aulakh, P. (2019, September). Supply chain digital twins: Opportunities and challenges beyond the hype. In (pp. 1–6). Cambridge, United Kingdom. doi: 10.17863/CAM.45897

Stadtler, H. (2009). A framework for collaborative planning and state-of-the-art. In H. Günther, Meyr, & Herbert (Eds.), *Supply chain planning: Quantitative decision support and advanced planning solutions* (pp. 3–28). Springer. doi: 10.1007/978-3-540-93775-3

Sultana, N. N., Meisheri, H., Baniwal, V., Nath, S., Ravindran, B., & Khadilkar, H. (2020). Reinforcement learning for multi-product multi-node inventory management in supply chains. *arXiv preprint arXiv:2006.04037*.

Sutton, R. S. (1990). Integrated architectures for learning, planning, and reacting based on approximating dynamic programming. In *Machine learning proceedings 1990* (pp. 216–224). Elsevier.

Sutton, R. S., & Barto, A. G. (2018). *Reinforcement Learning: An Introduction* (Second ed.). Cambridge: The MIT Press.

Szepesvári, C. (2010). *Algorithms for Reinforcement Learning* (R. J. Brachman & T. Dietterich, Eds.). Morgan & Claypool Publishers.

Tadepalli, P. (2010). Average-reward reinforcement learning. In C. Sammut & G. I. Webb (Eds.), *Encyclopedia of machine learning* (pp. 64–68). Boston, MA: Springer US. Retrieved from https://doi.org/10.1007/978-0-387-30164-8_49 doi: 10.1007/978-0-387-30164-8_49

Tesauro, G. (1992, May 01). Practical issues in temporal difference learning. *Machine Learning*, *8*(3), 257-277. doi: 10.1007/BF00992697

Tijsma, A. D., Drugan, M. M., & Wiering, M. A. (2016). Comparing exploration strategies for q-learning in random stochastic mazes. In *2016 ieee symposium series on computational intelligence (ssci)* (pp. 1–8).

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. *European Journal of Operational Research*, *282*(2), 401–414. doi: 10.1016/j.ejor.2019.03.026

Topan, E., Tan, T., van Houtum, G.-J., & Dekker, R. (2018). Using imperfect advance demand information in lost-sales inventory systems with the option of returning inventory. *IISE Transactions*, *50*(3), 246–264.

Trzuskawska-Grzesińska, A. (2017). Control towers in supply chain management– past and future. *Journal of Economics and Management*, *27*(1), 114–133. doi: 10.22367/jem.2017.27.07

Tuner, B. (2003). *Infromation operations in strategic, operational, and tactical levels of war: A balanced systematic approach* (Unpublished master's thesis). Naval Postgraduate School, Monterey, California.

Uhlemann, T. H.-J., Lehmann, C., & Steinhilper, R. (2017). The digital twin: Realizing the cyber-physical production system for industry 4.0. *Procedia Cirp*, *61*, 335–340.

van Doesburg, R. (2011). *Global Supply Cahin Control Towers.* London: Capgemini.

Van Hasselt, H., Guez, A., & Silver, D. (2015). Deep reinforcement learning with double q-learning. *arXiv preprint arXiv:1509.06461*.

Vanvuchelen, N., Gijsbrechts, J., & Boute, R. (2020). Use of Proximal Policy Optimization for the Joint Replenishment Problem. *Computers in Industry*, *119*, 103239. doi: 10.1016/j.compind.2020.103239

Varshavskaya, P., Kaelbling, L. P., & Rus, D. (2006). On scalability issues in reinforcement learning for self-reconfiguring modular robots. In *Robitcs: Science and systems workshop on self-reconfigurable modular robots, philadelphia.*

Wang, L., Zeng, Y., Zhang, J., Huang, W., & Bao, Y. (2006). The Criticality of Spare Parts Evaluating Model Using Artificial Neural Network Approach. In V. N. Alexandrov, G. D. van Albada, P. M. A. Sloot, & J. Dongarra (Eds.), *Computational science – iccs 2006* (Vol. 3991, pp. 728–735). Berlin, Heidelberg: Springer Berlin Heidelberg.

Wang, Y., He, H., & Tan, X. (2019). Truly proximal policy optimization. *CoRR*, *abs/1903.07940*.

Wang, Y., He, H., Tan, X., & Gan, Y. (2019). Trust region-guided proximal policy optimization. *CoRR*.

Wang, Y., Wang, X., & Liu, A. (2020). Digital twin-driven supply chain planning. *Procedia CIRP*, *93*, 198–203.

Wang, Z., Bapst, V., Heess, N., Mnih, V., Munos, R., Kavukcuoglu, K., & de Freitas, N. (2016). Sample efficient actor-critic with experience replay. *arXiv preprint arXiv:1611.01224*.

Winston, W. L. (2004). *Operations research: Applications and algorithms* (Fourth ed.; J. B. Goldberg, Ed.). Brooks/Cole Cengage Learning.

Xu, J., Zhang, J., & Liu, Y. (2009). An adaptive inventory control for a supply chain. In *2009 chinese control and decision conference* (pp. 5714–5719).

Yan, W. J., Tan, P. S., Koh, N. W., Tan, Y. Q., & Zhang, A. N. (2012). Towards better supply chain visibility - The design and implementation of a supply chain system S-ConTrol to support an operational HQ in Singapore. *IEEE International Conference on Industrial Engineering and Engineering Management*, 971–975. doi: 10.1109/IEEM.2012.6837885

Yogeswaran, M., & Ponnambalam, S. (2012). Reinforcement learning: exploration–exploitation dilemma in multi-agent foraging task. *Opsearch*, *49*(3), 223–236.

Zahedi, F. M., Bansal, G., & Ische, J. (2010, October 23). Succes factors in cooperative online marketplaces: Trust as the social capital and value generator in vendors-exchange relationships. *Journal of Organizational Computing and Electronic Commerce*, *4*(20), 295–327. doi: 10.1080/10919392.2010.516626

Zhang, H., Huang, R., & Zhang, S. (2020). Integrating Learning and Planning. In H. Dong, Z. Ding, & S. Zhang (Eds.), *Deep reinforcement learning: Fundamentals, research and* *applications* (pp. 307–316). Singapore: Springer Singapore. Retrieved from `https://doi.org/10.1007/978-981-15-4095-0{_}2` doi: 10.1007/978-981-15-4095-0_2

Zhang, H., & Yu, T. (2020). Taxonomy of Reinforcement Learning Algorithms. In H. Dong, Z. Ding, & S. Zhang (Eds.), *Deep reinforcement learning: Fundamentals, research and applications* (pp. 125–133). Singapore: Springer Singapore. Retrieved from `https://doi.org/10.1007/978-981-15-4095-0{_}3` doi: 10.1007/978-981-15-4095-0_3

Zijm, W. H. (2000). Towards intelligent manufacturing planning and control systems. *OR Spektrum*, *22*(3), 313–345. doi: 10.1007/s002919900032

Appendices

A | Interview Operational Planners

1. Hoe vaak is het nodig om een operationele interventie uit te voeren? (per dag/week)
2. Op welke manier komt een alert tot stand?
   (a) Zijn de alerts altijd correct?
   (b) Controleer je de juistheid van de alerts?
3. Zijn de interventies vooral reactief of proactief?
4. Zijn er alerts die een vaste interventie behoren?
   (a) Zo ja, welke beslissing hoort bij welke situatie?
      i. Stock out
      ii. Reparatie
      iii. Anders, namelijk...
   (b) Zo nee, waarom wijken beslissingen af?
      i. Door gevoel?
      ii. Ervaring?
5. Waarop baseer jij beslissingen voor interventies? [Nooit – Zelden – Soms – Vaak – Heel vaak]
   (a) Intuïtie / ervaring
   (b) Vaste procedures
   (c) Beslismodellen (welke?)
   (d) KPI's
   (e) Anders, namelijk...
6. Welke indicatoren neem jij in acht in de besluitvorming van een operationele interventie?
   (a) Beschikbaarheid van onderdelen (Service Level Agreements)
   (b) Kosten
   (c) Anders, namelijk...
7. In welke situaties zou je elk van onderstaande interventies uitvoeren, en waarom? *(Voorbeeld: Cannibalization - in house reparatie - kosten drukken)*

   (a) Stock reallocation
   (b) Do nothing
   (c) Expediting
   (d) Emergency shipments
   (e) Cannibalization
   (f) Anders, namelijk...

8. Welke van deze interventies gebruik je het meest?
9. Zijn er ook situaties denkbaar waar twee of meerdere interventies plaats vinden voor één alert? *(Voorbeeld: Cannibalization en Emergency shipment voor één kapotte APU)*
   (a) Zo ja, welke combinaties komen vaak voor?
   (b) Welke interventies zijn absoluut niet te combineren?
10. Welke gegevens krijg je vanuit een alert om tot een besluit te komen?
   (a) Zijn deze gegevens voldoende?
      i. Ja
      ii. Nee
      iii. Soms
   (b) Welke gegevens mis je om tot een goed besluit te komen?
   (c) Worden de gegevens op een prettige manier verstrekt?
      i. Zo niet, waarom niet?
      ii. Hoe zou het beter kunnen?
11. Hoe bepaal of meet je de kwaliteit van je beslissing ... en welke tools gebruik je hiervoor?
   (a) ...op het moment van de beslissing...?
   (b) ...achteraf...?
      i. Zijn er beslissingen die je achteraf gezien anders zou maken? (Onderbouw met voorbeeld)
      ii. Welke informatie heb je achteraf gezien wel, die je op het moment van beslissen niet hebt?
      iii. Gebruik je de kennis die je achteraf terug gekoppeld krijgt van een beslissing tijdens het maken van een nieuwe beslissing?
      iv. In welke mate is het persoonlijke verbeterproces onderdeel van ervaring, kennis of intuïtie?
      v. Is deze ervaring, kennis en intuïtie over te dragen aan een ander persoon? Licht toe.
   (c) Zijn er ook situaties waarin je de kwaliteit van een beslissing helemaal niet bepaalt?
      i. Waarom niet en welke situaties zijn dit?
12. Hoe zou je de huidige prestatie/kwaliteit van de beslissingen beoordelen op schaal van 1 tot 10?

   (a) Kun je de score onderbouwen met een aantal voorbeelden?

13. Hoe ver in de toekomst kijk je naar de gevolgen die een beslissing kan hebben? Scoor dit op [Nooit – Zelden – Soms – Vaak – Heel vaak]
    (a) **Alleen** naar de alerts die er **nu/vandaag** zijn
    (b) Naar de verwachte alerts die **alleen komende week** binnen gaan komen
    (c) Naar de verwachte alerts die **binnen de komende** x **maanden** binnen gaan komen
    (d) De **gehele CMA loop** door om te kijken wat er kan komen
14. Hoe beïnvloed een korte termijn beslissing (de interventie) de lange termijn (dus bijvoorbeeld de voorraad modellen) op het gebied van ...?
    (a) ...communicatie met tactische planners (voorraad modellen)
    (b) ...verbeteringen op voorraad modellen
    (c) ...samenwerking op algemeen vlak
    (d) ...kwaliteit van tactische output
    (e) ...andere vlakken
15. Hoeveel vertrouwen heb je in algoritmes of computer modellen bij het maken van vergelijkbare beslissingen? (Score 1-10)
    (a) Waarom denk je dat?
    (b) Wat denk je dat een algoritme mist?
    (c) Welke beslissingen zou je algoritmes wel toevertrouwen?
    (d) Welke factoren hebben invloed op het vertrouwen? Schaal [Geen - Weinig - Gemiddeld - Veel - Heel veel]
        i. Ervaring
        ii. Vooroordeel
        iii. Onbekendheid met modellen
        iv. Anders, namelijk...

B | Model Data

B.1 Data Content

Table B.1: Price information for component PN1 ($)

Condition	**Fmv**	**HighFmv**	**LowFmv**	**Max**	**Mean**	**Min**	**Recent Date**
Serviceable	*This information is removed from the public*						
Overhaul	*version of this Master's thesis*						

Table B.2: Demand data per year

2006	**2007**	**2008**	**2009**	**2010**	**2011**	**2012**	**2013**	**2014**	**2015**	**2016**	**2017**	**2018**	**2019**
This information is removed from the public version of this Master's thesis													

Table B.3: Turn rates per year

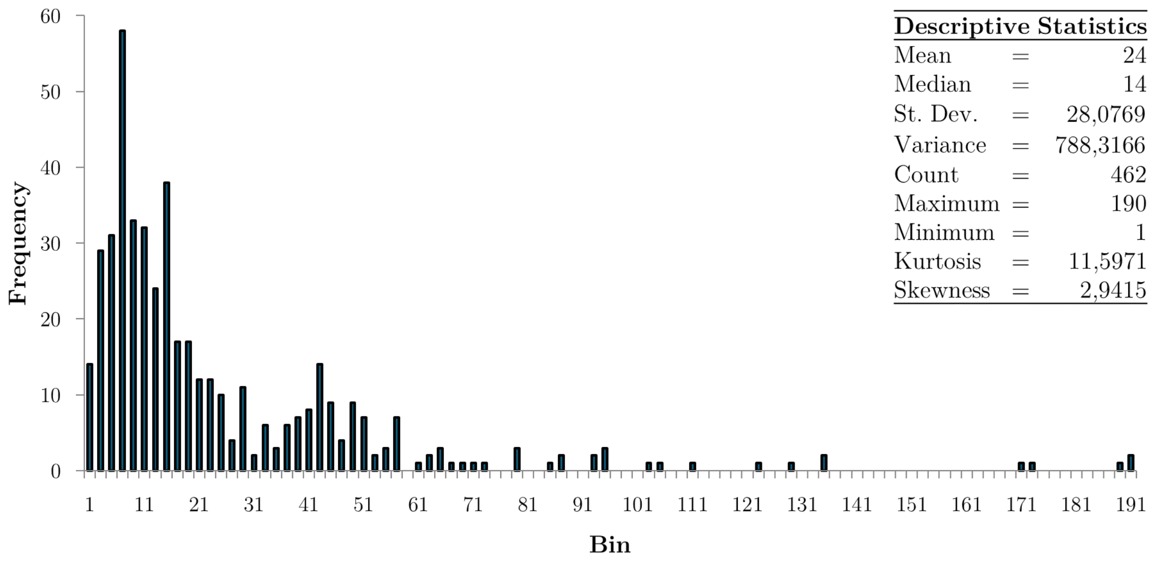
2006	**2007**	**2008**	**2009**	**2010**	**2011**	**2012**	**2013**	**2014**	**2015**	**2016**	**2017**	**2018**	**2019**
This information is removed from the public version of this Master's thesis													

B.2 Statistical Analysis Repair Turnaround Times

Based on the histogram, we are testing two probability distributions; the Normal- and Lognormal Distribution. We visually determine how representative the fitted distributions are following probability plots, P-P plot, and Q-Q plot, respectively. Additionally, we use a *goodness-of-fit test* to formally assess whether the observations are Independent and Identically Distributed (IID) from a particular distribution. Note that we can say the observations to be Lognormal if and only if the natural logarithm of the observations fit a Normal distribution.

The probability plots are a visual measurement of a comparison of estimating a distribution function. A *probability-probability (P-P) plot* represents the probability of the observed values versus the probability function of the compared distribution. The closer the probabilities of the observed values to the distribution are, the closer the P-P plot will be to an approximately linear function. The *quantile-quantile (Q-Q) plot* gives the graph of the q_i-quantile of a fitted distribution, with $i = 1, 2, ..., n$ for n observations. For more information about deriving the probability plots, see (Law, 2015).

Figure B.2 represents the P-P plots of the Normal and Lognormal distributions. Here, we conclude that the Normal distribution (Figure B.2a) is not close to a linear relationship. Therefore, it is not likely that the observed data fit a Normal distribution. When looking at the



Descriptive Statistics		
Mean	=	24
Median	=	14
St. Dev.	=	28,0769
Variance	=	788,3166
Count	=	462
Maximum	=	190
Minimum	=	1
Kurtosis	=	11,5971
Skewness	=	2,9415

Figure B.1: Full histogram of return turnaround times in days

Lognormal distribution (Figure B.2b), we find that the observed probabilities lay close to the linear relationship.

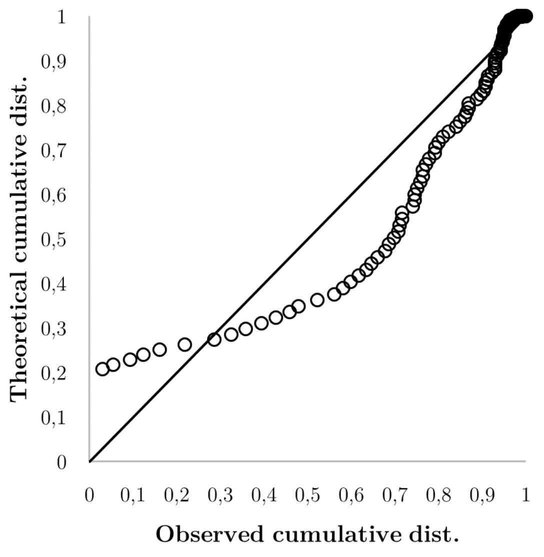
Within the Q-Q plot, we created quantiles with size 0.05, which gives a total of 19 quantiles. For each of these quantiles, the ordinate q values are determined and plotted against each other. Both results are given in Figure B.3 Results of the Normal Q-Q plot is visualized in Figure B.3a. Like the P-P plot, the Q-Q plot of the Normal distribution is far from the linear relationship line and, as shown in Figure B.3b, the Lognormal distribution is close to the linear relationship. For this reason, we expect the observed data to be Lognormally distributed.

Finally, we performed a *goodness-of-fit test*. We started with the chi-square test. Following Law (2015) is the chi-square hypothesis test a more formal comparison of a histogram with a fitted density function. For this test, we must first divide the fitted distribution into k adjacent intervals $[a_0, a_1), ..., [a_{k-1}, a_k)$. A representative way of calculating is $k = (x_{max} - x_{min})/\sqrt{N}$, where x_{max} and x_{min} represent the max and min observation and N is the sample size. Next, the χ^2-value can be calculated by Equation B.1. Here, n_j gives the number of observations in the jth interval, and E_j gives the uniform distributed expected observations in the jth interval.

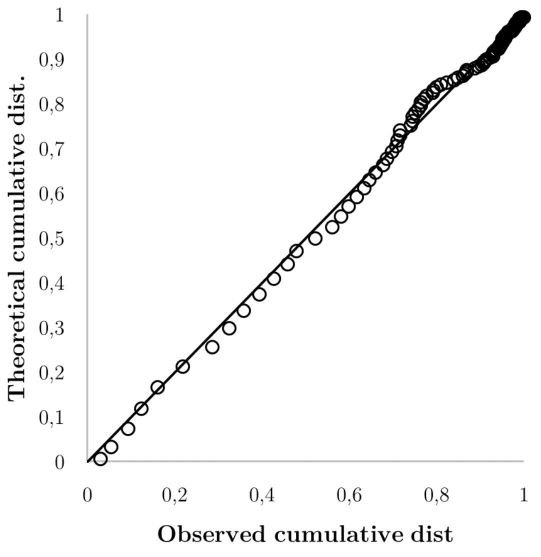
$$\chi^2 = \sum_{j=1}^{k} \frac{n_j - E_j}{E_j} \tag{B.1}$$

We reject the hypothesis H_0 if $\chi^2 > \chi^2_{k-1,1-\alpha}$, the critical point of the chi-square distribution. Within our data setting we have $k = 22$ with $\alpha = 0.05$. This gives us $\chi^2_{21,0.95} = 32.67$. For the Normal distribution, we found a chi-value of $\chi^2 = 528$, and for the Lognormal distribution, we found a value of $\chi^2 = 62$. Since both chi-values exceed the critical value, we have to reject the null hypothesis H_0.

In some cases, the null hypothesis may be rejected when it is actually true, *Type I error*. Therefore, we tested the data with the Kolmogorov-Smirnov (K-S) Test. This type of testing does not require grouping of data, which might cause loss of information in interval specification. To define the K-S statistic, we have to define the maximum vertical distance between the functions

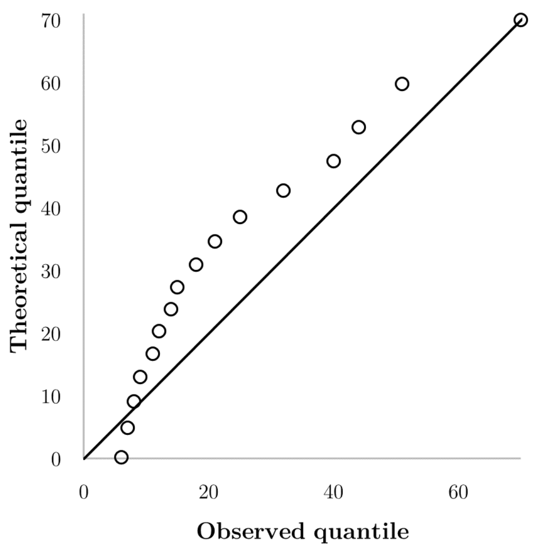


(a) Normal distribution

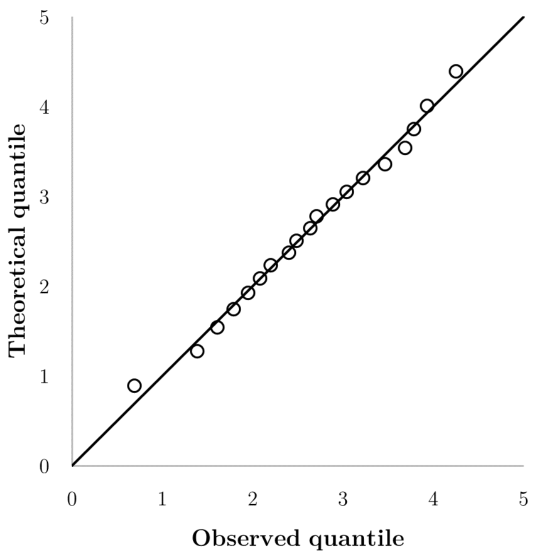


(b) Lognormal distribution

Figure B.2: P-P plots of repair turnaround times for Normal and Lognormal distribution



(a) Normal distribution



(b) Lognormal distribution

Figure B.3: Q-Q plots of repair turnaround times for Normal and Lognormal distribution

$F_n(x)$ and $\hat{F}(x)$. Mathematically, we express this as denoted in Equation B.2 (Law, 2015). Obviously, the smaller the value of D_n, the better the fit of the distribution.

$$D_n = \sup_x \{|F_n(x) - \hat{F}(x)|\} \tag{B.2}$$

We can test the null hypothesis by Equation B.3. If the given equation is true, we can reject H_0. In this equation, $c_{1-\alpha}$ represents the critical value which should be exceeded for rejecting H_0. For $\alpha = 0.05$, we know $c_{0.95} = 1.358$ (Law, 2015). For the Normal distribution we find a test-statistic $D_n = 0.2078$. This gives us the following equation $4.934 > 1.358$, from which we can conclude that we reject the null hypothesis. For the Lognormal distribution $D_n = 0.0519$, which gives $1.122 \not> 1.358$. Therefore, we do not reject the null hypothesis H_0.

$$\left(\sqrt{N} + 0.12 + \frac{0.11}{\sqrt{N}}\right) D_n > c_{1-\alpha} \tag{B.3}$$

C | Extensive Literature

C.1 Interchangeable Components

Within the literature review, in Chapter 3, we discussed several different interventions. We found that the intervention *Interchangeable Components* is not used in literature as preventive intervention from the reviewed literature.

As described in Section 2.3.4 the IAC uses interchangeability of components to prevent backorders from occurring. Interchangeability of components is a long-used concept in manufacturing. Interchangeable components were used for weaponry initially, but currently, the automotive and pharmaceutical industries use this intervention often. The literature points out that an interchangeable approach causes fewer delays, lower costs, and higher access to pharmaceuticals (Sharad, Nitin, Styavan, Mitul, & Shrivastava, 2011). From a manufacturing perspective, interchangeability permits the components to be replaced or assembled without additional treatment. From the design, technology, and operation of machine point of view, this has excellent benefits (Curley, 2016). Besides the mentioned advantages, interchangeability can be very helpful in spare-part models and inventory modeling.

C.2 Extensions to the Markov Decision Process

Within this section, we will briefly introduce the Average Reward Value Function and the Partially Observable Markov Decision Process.

C.2.1 Average Reward Value Function

With the Average Reward Reinforcement Learning (ARL), the algorithms optimize the average reward per time step. The average reward value function determines for any policy π the average reward ρ^π. This average reward is independent of the start state (Silver, 2015).

$$\rho^\pi = \lim_{T \to \infty} \frac{1}{T} \mathbb{E}\left[\sum_{t=1}^{T} R_t\right]$$

Multiple episodic domains have natural termination conditions, such as the ending of a game. For other scenarios, there might not be a termination state. For problems where no fixed termination state is available, ARL is very functional (Tadepalli, 2010). In a given case, we can use the average reward Bellman equation instead of the regular Bellman equations (Bertsekas, 2019).

$$\begin{aligned}\tilde{v}_\pi(s) &= \mathbb{E}_\pi\left[(R_{t+1} - \rho^\pi) + \sum_{k=1}^{\infty}(R_{t+k+1} - \rho^\pi)|S_t = s\right] \\ &= \mathbb{E}_\pi[(R_{t+1} - \rho^\pi) + \tilde{v}_\pi(S_{t+1})|S_t = s]\end{aligned}$$

C.2.2 Partially Observable Markov Decision Process

Markov models can also include hidden states. Specific models are called Partially Observable Markov Decision Process (POMDP). The states are not fully represented by the observation for the agent (Åström, 1965; Z. Ding et al., 2020). Additionally to the MDP tuple, a finite set of observations ($\mathcal{O}$) and an observation function are used ($\mathcal{Z}$) (Silver, 2015). The observation function is defined as follows:

$$\mathcal{Z}^a_{s'o} = \mathbb{P}[O_{t+1} = o | S_{t+1} = s', A_t = a]$$

The POMDP is a distribution over the latent states, given the history. The history (H_t) can be denoted as a sequence of actions, observations and rewards; $H_t = A_0, O_1, R_1, \cdots, A_{t-1}, O_t, R_t$. The distribution given the history is also called the *belief state* ($b(h)$) (Silver, 2015; Sutton & Barto, 2018).

$$b(h) = \mathbb{P}[S_t = s^i | H_t], \forall i \in \{1, 2, \cdots, d\}$$

However, since the environment is partially unknown, finding an acceptable policy is complicated. Randomness allows the agent in this situation to choose different actions in different locations. This prevents the agent from getting stuck in local optima and therefore loops (Kaelbling, Littman, & Cassandra, 1998; Smallwood & Sondik, 1973). In a POMDP, the agent makes an observation based on the action and resulting state, with the same goal; maximizing the discounted rewards. Kaelbling et al. (1998) describes an algorithm to solve the POMDP. Since our problem can be defined as a regular MDP, we will not dive further into this algorithm.

C.3 Solving Algorithms of Reinforcement Learning

This section presents the algorithms of Approximate Dynamic Programming (Section C.3.1), TD-Control (Section C.3.2), Dyna-Q (Section C.3.3), Deep Q-Networks (Section C.3.4), Actor-Critics (Section C.3.5), and Proximal Policy Optimization (Section C.3.6). Each algorithm is provided with a brief explanation.

C.3.1 Approximate Dynamic Programming

The Approximate Dynamic Programming algorithm consists of four main steps. All steps are shown in Algorithm 1 (Mes & Perez Rivera, 2017). The optimality function of the ADP is given by Equation C.1. We will briefly explain the algorithm, step by step next.

$$\bar{V}^n_t(S_t) = \max_{x_t \in \mathcal{X}_t} (C_t(S_t, x_t) + \gamma \mathbb{E}\{\bar{V}^{x,n}_{t+1}(S_{t+1} | S^x_t)\} \tag{C.1}$$

The first step of the ADP algorithm is the initialization step. This step consists of an initial approximation of the value function for all time periods and an initialization for individual start values.

Next, in Step 1. of the algorithm, a sample path is chosen. In each iteration n, the sample path $\omega^n \in \Omega$ is determined. $W_t(\omega^n)$ is denoted the realized sample-path at time t. The sample path represents the stochasticity within the process which occurs during the episode (Mes & Perez Rivera, 2017).

For the computation step, step 2, the ADP will determine the best possible action, update the value function, and determine the next state. The algorithm will perform this step repeatedly for all $t \in T$. The algorithm performs steps 1 and 2 for N episodes before it returns the optimal value function $\bar{V}^N_t(S_t)^{x,n}$.

Algorithm 1 Approximate Dynamic Programming

```
Step 0.  Initialization

         (a)  Choose an initial approximation $\bar{V}_t^0 \forall t \in \mathcal{T}$
         (b)  Set the iteration counter $n = 1$, and set the maximum number of
              iterations $N$.
         (c)  Set the initial state to $S_0^1$

Step 1.  Choose a sample path $\omega^n$

Step 2.  Do for $t = 0, \cdots, T$

         (a)  Solve: $\hat{v}_t^n = \max_{x_t \in \mathcal{X}_t} \left( C_t(S_t, x_t) + \gamma \bar{V}_t^{n-1}(S^{M,x}(S_t, x_T)) \right)$
              Let $x_t^n$ be the best decision and let $S_t^x = S^{M,x}(S_t, x_t)$
         (b)  Update the value function: $\bar{V}_{t-1}^n(S_{t-1}^x) = (1-\alpha)\bar{V}_{t-1}^{n-1}(S_{t-1}^x) + \alpha \hat{v}_t^n$
         (c)  Compute new pre-decision state: $S_{t+1} = S^M(S_t, x_t^n, W_{t+1}(\omega^n))$

Step 3.  Increment $n$. If $n \leq N$ go to Step 1.

Step 4.  Return the value functions $\bar{V}_t^N(S_t)^{x,n} \forall t \in \mathcal{T}, S_t \in \mathcal{S}$
```

C.3.2 TD-Control Method

The algorithm we will further explain is Q-Learning. We keep track of a Q-table $Q(s, a)$ for the regular Q-Learning. So, before starting the Q-Learning algorithm, we initialize the table with preferable values. We will execute the algorithm for N episodes. At the beginning of each episode, we will initialize the current state to the start state S_0.

Next, we will perform the following steps repeatedly until the agent visits the terminal state. First, the agent selects an action. Following the book of Sutton and Barto (2018) the action, $A(S_t)$ is given by an ϵ-greedy approach over Q-values by the policy. By this action, the agent will receive a reward R_{t+1} and move to the next state S_{t+1}.

With this observation, we update the Q-table following Equation C.2. By iterating over steps and episodes, our knowledge of the environment will increase. In the update-function, α represents the discount factor and γ the learning rate. Here, the discount factor gives the weight of including current observation, and the learning rate determines the weight of future rewards for the current observation.

$$Q(S, A) \leftarrow Q(S, A) + \alpha[R + \gamma \min_a Q(S', a) - Q(S, A)] \tag{C.2}$$

Q-Learning does not require a model of the environment for the reward and next state probability distributions. By interacting with the environment, the agent can earn rewards and improve. The goal of Q-Learning is to determine the best policy that maximizes the Q-value. The mentioned steps are presented in Algorithm 2.

Algorithm 2 Q-Learning

```
Initialize $Q(s,a), \forall s \in \mathcal{S}, a \in \mathcal{A}(s)$
Do for $N$ episodes:
    (a)  $S \leftarrow$ current (non-terminal) state
    (b)  $A \leftarrow \epsilon$-greedy$(S, Q)$
    (c)  Execute action $A$; Observe resultant reward $R$, Get next state $S'$
    (d)  $Q(S,A) \leftarrow Q(S,A) + \alpha[R + \gamma \max_a Q(S',a) - Q(S,A)]$
```

C.3.3 Dyna-Q

The Dyna-Q algorithm, presented in Algorithm 3 (Sutton & Barto, 2018; Zhang et al., 2020), is based on the Q-Learning algorithm. We see from the algorithm structure that steps (a) until (d) of the Dyna-Q algorithm are the Q-Learning algorithm steps.

Further, in step (e), we will update a model. This model consists of all observations so far. We gather all experience tuples, and we use this as an assumption of the deterministic environment (Sutton & Barto, 2018). With this model, we can use a Model-Based approach within a Model-Free environment. By iterating over this created model for n times, we rapidly create more state-action pairs observations. Note that within the Model-Based approach, we only look at earlier observed actions and states. Updating the Q-table based on the Model-Approach helps converging faster to the optimal solution.

If we want to implement the algorithm within a changing environment, Sutton and Barto (2018) recommends using the Dyna-Q+ algorithm. This algorithm encourages behavior that tests long-untried actions; we give the agent a bonus reward on simulated experiences involving these actions. The extra reward for exploration is defined as $R + \kappa\sqrt{\tau}$. In this reward function, κ is a small number, and τ represents the time steps that we did not try the transition. For further elaboration, see (Sutton & Barto, 2018, Section 8.3).

Algorithm 3 Dyna-Q

```
Initialize $Q(s,a)$ and $Model(s,a)$ $\forall s \in \mathcal{S}$and $a \in \mathcal{A}(s)$
Do forever:
    (a)  $S \leftarrow$ current (non-terminal) state
    (b)  $A \leftarrow \epsilon$-greedy$(S, Q)$
    (c)  Execute action $A$; Observe resultant reward $R$, Get next state $S'$
    (d)  $Q(S,A) \leftarrow Q(S,A) + \alpha[R + \gamma \max_a Q(S',a) - Q(S,A)]$
    (e)  $Model(S,A) \leftarrow R, S'$
    (f)  Repeat $n$ times:
             $S \leftarrow$ random previously observed state
             $A \leftarrow$ random action previously taken in $S$
             $R, S' \leftarrow Model(S,A)$ random action previously taken in $S$
             $Q(S,A) \leftarrow Q(S,A) + \alpha[R + \gamma \max_a Q(S',a) - Q(S,A)]$
```

C.3.4 Deep Q-Networks

As mentioned in Section 3.5, Deep Q-Network (DQN) is an elaborate version of Q-Learning. The first step of improvement concerns the *replay buffer*. For each time step t, the algorithm stores the experience (S_t, A_t, R_t, S_{t+1}) in this replay buffer. Secondly, DQN uses a target network which is used to generate the Q-Learning targets (Huang, 2020). Furthermore, It is difficult to use arbitrary history lengths as input for the Neural Network. Therefore, DQN works with a history function ϕ. This function combines the four most recent frames of history and uses this as input for the Neural Network (Huang, 2020). These history frames are represented by the last four states from the replay buffer. With an ϵ-greedy approach, the algorithm determines the next action, based on the Q-values (Mnih et al., 2013). The algorithm is presented in Algorithm 4, with the goal to maximize Y_j.

With DQN we will train the network based on the Q-Learning function. Again, similar to the Q-Learning algorithm, we will determine the action by an $\epsilon-$greedy approach. After executing A_t, we observe the reward and the image of the next state. With this tuple observation, we can preprocess the equivalent to the state s, which is ϕ. Based on the neural network's parameters θ, we define the network's loss function as the Squared Error between target Q-value and the Q-value output from the network.

Algorithm 4 DQN

```
Initialize replay memory $\mathcal{D}$ to capacity $N$
Initialize action-value function $Q$ with random weights $\theta$
Initialize target action-value function $\hat{Q}$ with parameter $\hat{\theta} \leftarrow \theta$
Do for $N$ episodes:
    Initialize sequence $S_0 = S$ and preprocessed sequence $\phi_0 = \phi(S_0)$
    Do for $T$ steps:
        $A \leftarrow \epsilon$-greedy$(S, Q)$
        Execute action $A$; Observe resultant reward $R$, Get next state $S'$
        $S_{t+1} \leftarrow S_t, A_t, S'$ and preprocess $\phi_{t+1} = \phi(S_{t+1})$
        Store transition $(\phi_t, A_t, R_t, \phi_{t+1})$ in $\mathcal{D}$
        Sampel random minibatch of transitions $(\phi_j, A_j, R_j, \phi_{j+1})$ from $\mathcal{D}$
        if episode terminates at step $j+1$ then:
            $Y_j = R_j$
        else:
            $Y_j = R_j + \gamma \max_{a'} \hat{Q}\Big(\phi_{j+1}, a'; \hat{\theta}\Big)$
        Perform a gradient descent step on $\Big(Y_j - Q(\phi_j, a_j; \theta)\Big)^2$ with respect to $\theta$
        Reset $\hat{Q} = Q$ after every $C$ steps
```

C.3.5 Actor-Critics

The Actor-Critic (AC) method is located in the intersection of policy and value based methods; we learn both policy and value function. Here, AC uses an *actor* (learn policy) and *critic* (learn value) (Z. Ding et al., 2020). The general algorithm of actor-critic is that we run a policy π and collect the corresponding S_t, A_t, R_t, S_{t+1}. Unlike DQN, Actor-Critics learn the S_t, A_t, R_t, S_{t+1} in each step (Z. Ding et al., 2020). Then, we estimate the Q-value (advantage) by bootstrapping. Based on the advantage, we can calculate both policy function $J(\theta)$ and value function $J_{V_{\psi}^{\pi_\theta}}(\psi)$. Within these updating functions, we use θ and ψ as parameters of the policy and value functions.

Additionally to regular actor-critics, we can use Synchronous Advantage Actor-Critic (A2C) which focusses on parallel training of different workers. These workers all contain an own actor and critic and communicate to a master node via a coordinator (Z. Ding et al., 2020). In a A2C setup, the workers are only responsible for the interaction with the environment. Updating happens in the master node. In other words, the worker runs the policy, estimates the advantage, calculates policy and value function, and returns these function values. The master updates the parameters θ and ψ after all workers are finished.

As described in Section 3.5, the critic needs to estimate the value of the current policy of the actor. Since this is a value prediction problem, an algorithm such as SARSA, Q-learning, or TD(λ) is useful (Szepesvári, 2010). The paper of Szepesvári (2010) further explains that an actor can be implemented in two ways: by moving the current policy towards the greedy policy or perform gradient ascent. This results in Algorithm 5, from (Z. Ding et al., 2020). Within the algorithm of A3C the coordinator is removed, compared to A2C. As a result, the master nodes can update whenever the worker is finished with the gradient computation which leads to a better computational efficiency. The algorithm of A3C is presented in Algorithm 5.

Algorithm 5 A3C

```
Master:
Hyperparameters: step size $\eta_\psi$ and $\eta_\theta$, current policy $\pi_\theta$, value function $V_\psi^{\pi_\theta}$
Input: gradients $g_\psi$, $g_\theta$
$\psi = \psi - \eta_\psi g_\psi$; $\theta = \theta + \eta_\theta g_\theta$
Return ($V_\psi^{\pi_\theta}$,$\pi_\theta$)
Worker:
Hyperparameters: reward discount for factor $\gamma$, the length of trajectory $L$
Input: value function $V_\psi^{\pi_\theta}$, policy $\pi_\theta$
$(g_\theta, g_\psi) = (0, 0)$
for $k = 1, 2, \cdots, K$ do
    $(\theta, \psi) =$ Master$(g_\theta, g_\psi)$
    Run policy $\pi_\theta$ for $L$ time steps, collection $\{S_t, A_t, R_t, S_{t+1}\}$
    Estimate advantages $\hat{A}_t = R_t + \gamma V_\psi^{\pi_\theta}(S_{t+1}) - V_\psi^{\pi_\theta}(S_t)$
    $J(\theta) = \sum_t \log \pi_\theta(A_T|S_t)\hat{A}_t$
    $J_{V_\psi^{\pi_\theta}}(\psi) = \sum_t \hat{A}_t^2$
    $(g_\psi, g_\theta) = (\nabla J_{V_\psi^{\pi_\theta}}(\psi), \nabla J(\theta))$
end for
```

C.3.6 Proximal Policy Optimization

The value function (V^θ) from the TRPO algorithm is replaced with a learned value function approximation ($V_\phi(s)$) (Schulman et al., 2017). The algorithm uses iterations of a fixed length over the time window. In the iterations, the algorithm maximizes directly over the expected sum of rewards, C.3. Where τ represents the state-action trajectory $\tau = (s_0, a_0, s_1, \cdots)$, θ (the gradient decent) is the parameter of the stochastic policy, and ρ represents the system dynamics by $\rho_\theta(\tau) = p(s_o)\pi(a_0|s_0)p(s_1|s_0, a_0)\cdots$ (Heess et al., 2017; Z. Ding et al., 2020). In the optimization function ($J_{PPO}(\theta)$) we find the regularization coefficient λ:

$$J_{PPO}(\theta) = \sum_{t=1}^{T} \frac{\pi_\theta(a_t|s_t)}{\pi_{old}(a_t|s_t)}\hat{A}_t - \lambda \mathrm{KL}[\pi_{old}|\pi_\theta] \tag{C.3}$$

The λ is dependent on π_θ and recovers the same optimizer as a constant. The KL-divergence constraint will determine whether the λ should be enlarged or reduced (Z. Ding et al., 2020). The KL parameter represents the desired change in the policy per iteration, supported by scaling term $\alpha > 1$ (Heess et al., 2017). The scaling term adjust the KL-regularization coefficient if the policy falls outside the given interval $[\beta_{low}\mathrm{KL}_{target},\ \beta_{high}\mathrm{KL}_{target}]$.

With an enhanced PPO method, which is trust region-based with rollback, Y. Wang, He, Tan, and Gan (2019) propose an approach for improving PPO on both stability and sample efficiency. Alternatively, resetting the agent's environment can also ensure a more stable and better-enhanced training (Kristensen & Burelli, 2020). Nonetheless, these challenges are mostly common for problems with a very large state-space.

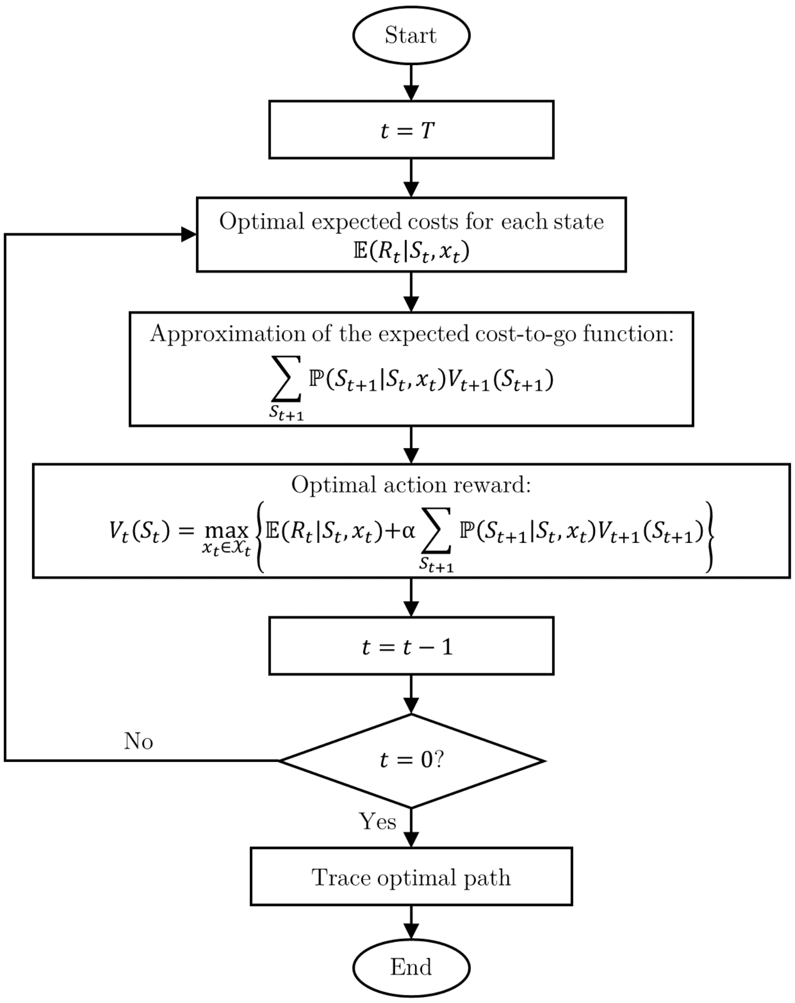
The PPO algorithm has proven to be effective in optimal control for inventory problems (Meisheri et al., 2020). Also in the control tower setting is the PPO algorithm tested and proved itself to be useful (Vanvuchelen et al., 2020). Since PPO is relatively easy to implement (compared to TRPO) and it provides robust and reliable results, is PPO a promising algorithm for our purpose. Algorithm 6 shows the different steps, derived from (Heess et al., 2017).

Algorithm 6 Proximal Policy Optimization

```
for $i \in \{1, 2, \cdots, N\}$ do
    Run policy $\pi\theta$ for $T$ time steps, collecting $\{s_t, a_t, r_t\}$
    Estimate advantages $\hat{A}_t = \sum_{t'>t} \gamma^{t'-t} r_{t'} - V_\phi(s_t)$
    $\pi_{old} \leftarrow \pi_\theta$
    for $j \in \{1, 2, \cdots, M\}$ do
        $J_{PPO}(\theta) = \sum_{t=1}^{T} \frac{\pi_\theta(a_t|s_t)}{\pi_{old}(a_t|s_t)} \hat{A}_t - \lambda \mathrm{KL}[\pi_{old}|\pi_\theta]$
        Update $\theta$ by a gradient method with respect to $J_{PPO}(\theta)$
    end for
    for $j \in \{1, \cdots, B\}$ do
        $L_{BL}(\phi) = -\sum_{t=1}^{T} \left(\sum_{t'>t} \gamma^{t'-t} r_{t'} - V_\phi(s_t)\right)$
        Update $\phi$ by a gradient method with respect to $L_{BL}(\phi)$
    end for
    if $\mathrm{KL}[\pi_{old}|\pi_\theta] > \beta_{high}\mathrm{KL}_{target}$ then
        $\lambda \leftarrow \alpha\lambda$
    else if $KL[\pi_{old}|\pi_\theta] < \beta_{low}\mathrm{KL}_{target}$
        $\lambda \leftarrow \frac{\lambda}{\alpha}$
    end if
end for
```

Currently, in the literature multiple flaws are found concerning Proximal Policy Optimization. Although PPO is a data efficient and easier to implement algorithm as TRPO, the performance of PPO can be unstable (Y. Wang, He, & Tan, 2019). The performance is dependent on the effectiveness of the exploratory policy search, and can be caused by bad initialization (Y. Wang, He, Tan, & Gan, 2019). With an enhanced PPO method, which is trust region based with rollback, Y. Wang, He, Tan, and Gan (2019) propose an approach for improving PPO on both stability and sample efficiency. Alternatively, resetting the environment for the agent can also ensure a more stable and better enhanced training (Kristensen & Burelli, 2020). Nonetheless, these challenges are mostly common for problems with a very large state-space.

D | Algorithm Flowcharts



Start
$t = T$
Optimal expected costs for each state $\mathbb{E}(R_t|S_t,x_t)$
Approximation of the expected cost-to-go function: $\sum_{S_{t+1}} \mathbb{P}(S_{t+1}|S_t,x_t)V_{t+1}(S_{t+1})$
Optimal action reward: $V_t(S_t) = \max_{x_t \in \mathcal{X}_t} \left\{ \mathbb{E}(R_t|S_t,x_t) + \alpha \sum_{S_{t+1}} \mathbb{P}(S_{t+1}|S_t,x_t)V_{t+1}(S_{t+1}) \right\}$
$t = t - 1$
$t = 0$?
No
Yes
Trace optimal path
End

Figure D.1: Structure of the SDP algorithm

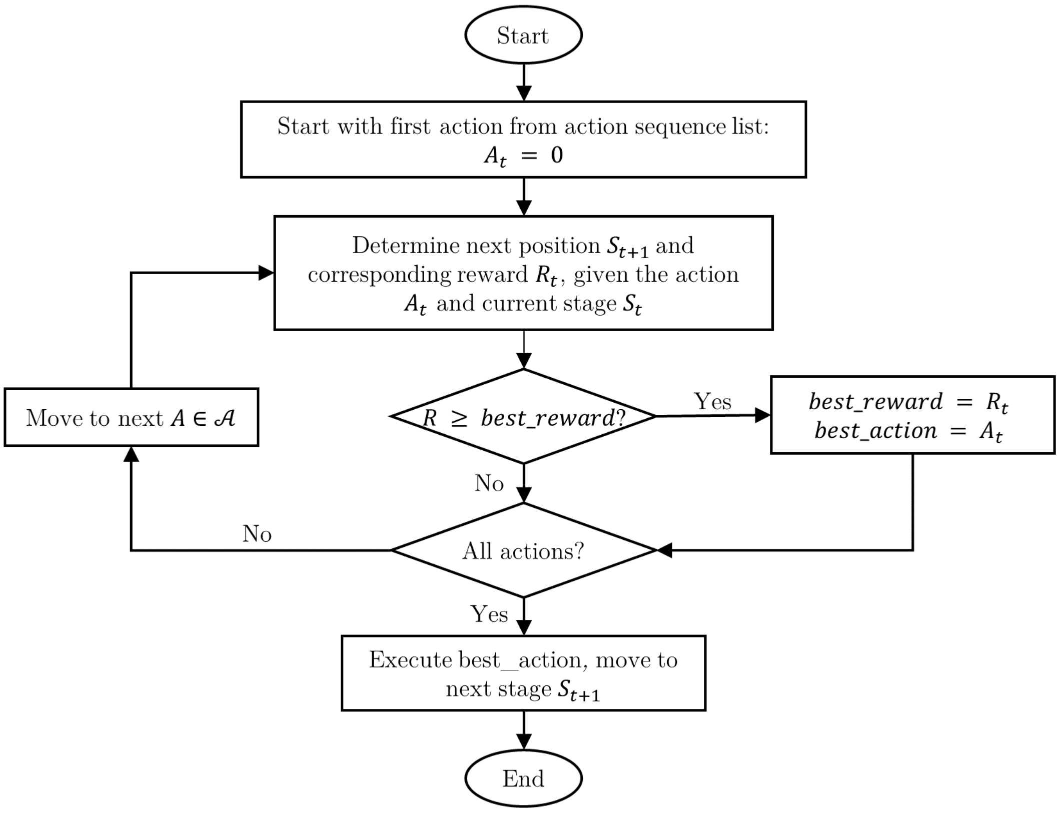

Start
Start with first action from action sequence list: $A_t = 0$
Determine next position S_{t+1} and corresponding reward R_t, given the action A_t and current stage S_t
$R \geq best\_reward$?
Yes
$best\_reward = R_t$
$best\_action = A_t$
No
All actions?
No
Move to next $A \in \mathcal{A}$
Yes
Execute best_action, move to next stage S_{t+1}
End

Figure D.2: Structure of the Simulation with Greedy Heuristic

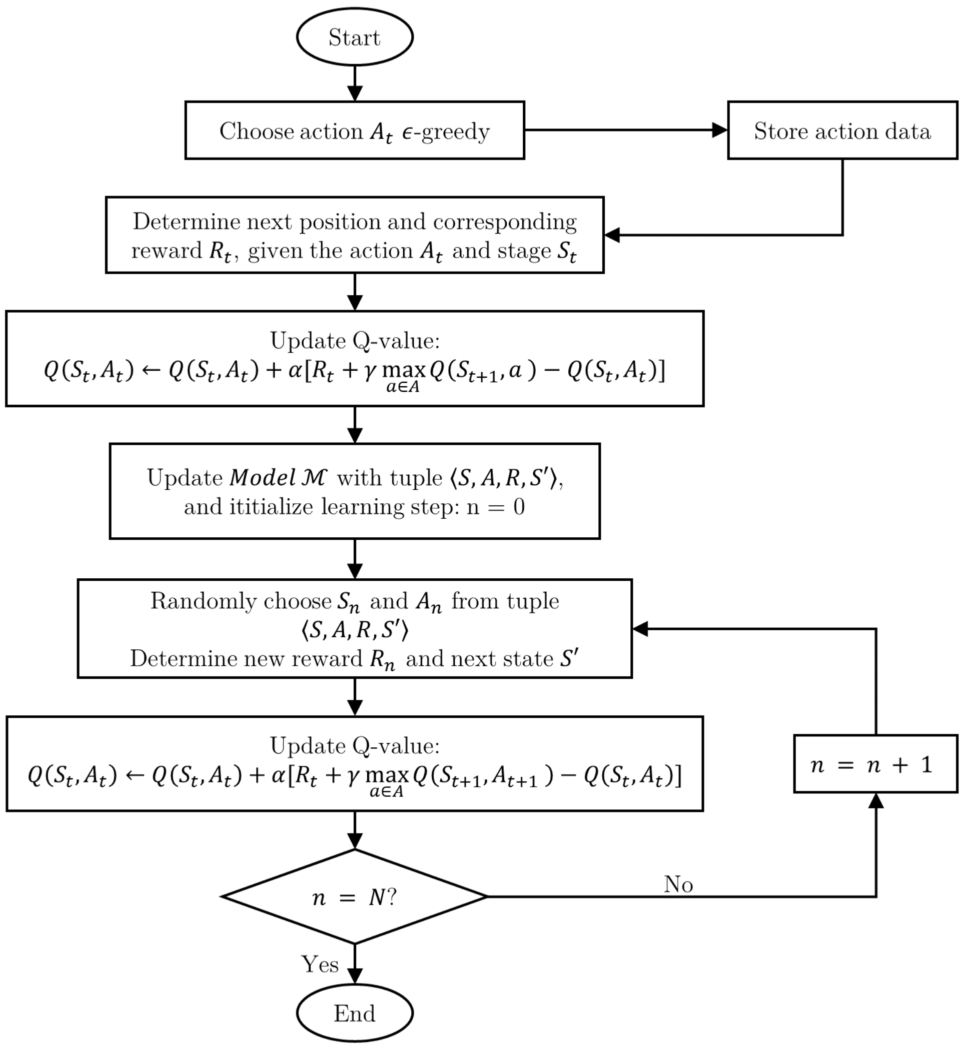

Start
Choose action A_t ϵ-greedy
Store action data
Determine next position and corresponding reward R_t, given the action A_t and stage S_t
Update Q-value:
$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha[R_t + \gamma \max_{a \in A} Q(S_{t+1}, a) - Q(S_t, A_t)]$
Update $Model\ \mathcal{M}$ with tuple $\langle S, A, R, S' \rangle$, and ititialize learning step: n = 0
Randomly choose S_n and A_n from tuple $\langle S, A, R, S' \rangle$
Determine new reward R_n and next state S'
Update Q-value:
$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha[R_t + \gamma \max_{a \in A} Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t)]$
$n = n + 1$
$n = N$?
No
Yes
End

Figure D.3: Structure of the Dyna-Q algorithm

E | Python Code - Import Data

E.1 Acquisition Costs

```python
def LoadAC():
    global ACost, CostTable
    summ = 0.0
    ws = wb['Acquisition']
    last_row = ws.max_row
    ### DETERMINE COST TABLE ###
    arr_row = 0
    for row in range(last_row):
        if ws.cell(row=row+1, column=1).value == 'PN1':
            for col in range(7):
                CostTable[arr_row,col] = ws.cell(row=row+1, column=col+2).value
            arr_row += 1
    for row in range(arr_row):
        summ += CostTable[row, 1]
    ACost = summ / arr_row
```

E.2 Turnover Rate

```python
def LoadToR():
    global turnover
    summ = 0.0
    TTest = False
    years = 0
    prod_row = 0
    ws = wb['TurnoverRate']
    last_row = ws.max_row
    last_col = ws.max_column
    ### CALCULATE MEAN TURNOVER RATE ###
    for row in range(last_row):
        if ws.cell(row=row + 1, column=1).value == 'PN1':
            prod_row = row + 1
        elif row == last_row - 1:
            sys.exit("Sorry, this part number is not found in the data base!")
    for col in range(last_col - 1):
        turnrate = ws.cell(row=prod_row, column=col + 2).value
        if ws.cell(row=2, column=col + 2).value == 2020:
            break
        elif not (turnrate == "") or not (turnrate == 0):
            TTest = True
            summ = summ + turnrate
            years += 1
        elif TTest:
            years += 1
    turnover = summ / years
```

E.3 Demand & Repair

```python
def LoadDemandRepair():
    global observ_data, r_mean, r_exp, r_sigma, invloc, TotalTAT, MATAT, MDrD,
    Tactical, DrD, repairs, customer, poolinv, leadprob, start_tact, repairdate
    col_PN = 1
    col_TAT = 1
    r_exp = 0
    summ = 0

    ws = wb['Transaction_Data']
    last_row = ws.max_row
    last_col = ws.max_column
    ### COLUMN DEFINITION ###
    for col in range(last_col):
        if ws.cell(row=1, column=col + 1).value == 'PARTNUMBER':
            col_PN = col + 1
        elif ws.cell(row=1, column=col + 1).value == 'Shop_TAT':
            col_TAT = col + 1
        elif ws.cell(row=1, column=col + 1).value == 'LINE_ADDED':
            col_add = col + 1
        elif ws.cell(row=1, column=col + 1).value == 'EXCH_TYPE':
            col_type = col + 1

        elif ws.cell(row=1, column=col + 1).value == 'EXCH_TYPE':
            col_type = col + 1
        elif ws.cell(row=1, column=col + 1).value == 'CORE_RCVD':
            col_rcvd = col + 1
        elif ws.cell(row=1, column=col + 1).value == 'STOCK_UPDATED':
            col_stock = col + 1
        elif ws.cell(row=1, column=col + 1).value == 'DELIVERED':
            col_del = col + 1

    ### REPAIR PROBABILITY ###
    for row in range(last_row):
        if ws.cell(row=row + 1, column=col_PN).value == 'PN1':
            observ_data.append(ws.cell(row=row + 1, column=col_TAT).value)
    r_mean = np.mean(np.log(observ_data))
    r_sigma = np.std(np.log(observ_data))
    for t in range(200):
        leadprob.append(lognorm.cdf(x=t + 1, s=r_sigma, scale=np.exp(r_mean)))

    ### DETERMINE DEMAND DAY IN MOVING AVERAGE ###
    for row in range(last_row):
        if ((ws.cell(row=row + 1, column=col_type).value == 'Forward' or
             ws.cell(row=row + 1, column=col_type).value == 'Shop') and
                ws.cell(row=row + 1, column=col_add).value <= datetime.datetime
    (2020, 1, 20)):
            # MOVING DEMAND
            LINEADD = ws.cell(row=row + 1, column=col_add).value
            REPTAT = ws.cell(row=row + 1, column=col_TAT).value
            TotalTAT[LINEADD] = REPTAT
            # MOVING INVENTORY LOCATIONS
            invloc[LINEADD] = {}  # Used in stock levels
    Sort_TAT = sorted(TotalTAT.items(), key=lambda x: x[0], reverse=False)
    FirstDate = Sort_TAT[0][0]

    for i in range(len(Sort_TAT)):
        if (FirstDate + datetime.timedelta(days=365)) <= Sort_TAT[i][0]:
            BeginAvg = i - 1
            break
    for i in range(len(Sort_TAT)):
        summ += Sort_TAT[i][1]
        TATHand = math.ceil((summ / (i + 1)) + 4)
        if i >= BeginAvg:
            MATAT.append(TATHand)
            deltaday = Sort_TAT[i][0] - FirstDate
            movingd = round((i + 1) / deltaday.days, 3)
            MDrD.append(movingd)
            Tactical[Sort_TAT[i][0]] = math.floor(InvPois(movingd * TATHand, 0.95))
    DrD = MDrD[-1]

    ### DETERMINE STOCK LEVELS DURING MOVING TIME PERIOD ###
    listdaterepairs = {}
    for index in range(len(Sort_TAT)):
        InR = 0
        AtC = 0
        daterepair = []
        dt = Sort_TAT[index][0]
        for row in range(last_row - 1):
            RCVD = ws.cell(row=row + 2, column=col_rcvd).value
            STOCK = ws.cell(row=row + 2, column=col_stock).value
            DELIVER = ws.cell(row=row + 2, column=col_del).value
            TYPE = ws.cell(row=row + 2, column=col_type).value
            if (dt >= DELIVER) and (dt <= STOCK):
                if TYPE == 'Forward':
                    if dt < RCVD:
                        AtC += 1
                    else:
                        InR += 1
                        daterepair.append(DELIVER)
                elif TYPE == 'Shop':
                    InR += 1
                    daterepair.append(DELIVER)
        listdaterepairs[dt] = daterepair
        repairs.append(InR)
        customer.append(AtC)
        if index >= BeginAvg:
            for tac in range(len(Tactical)):
                if dt == list(Tactical.keys())[tac]:
                    pool = max(InR + AtC, list(Tactical.values())[tac])
                    poolinv.append(pool)
        invloc[dt] = (InR, AtC)
    start_tact = list(Tactical.values())[-1]
    for i in range(len(list(listdaterepairs.values())[-1])):
        dinrepair = math.floor((datetime.datetime(2020, 1, 20) - list(listdaterepairs.values())[-1][i]).total_seconds() / (24 * 60 * 60))
        repairdate.append(dinrepair)
```

E.4 Grid Definition

```
def DetMaxGrid():
    global MaxBO, MaxInv, Tactical, repairs, customer, start_rep, start_oh, start_cust
    ### MAXGRID DEFINITION - BACKORDERS AND INVENTORY ###
    for k in range(10):
        if poisson.pmf(k, T * DrD) < 0.01:
            MaxBO = k - 1
            break
    MaxInv = max(math.floor(1.5 * start_tact), 2)
    start_cust = customer[-1]
    start_rep = repairs[-1]
    start_oh = max(start_tact - start_cust - start_rep, 0)
```

E.5 Transition Probability Table

```python
def ProbabilityTable():
    global demp, repp, expp, extab, SDP_repp, SDP_expp
    tabular = np.zeros((start_rep, T))
    tabuexp = np.zeros((start_rep, T))
    repp = np.zeros((start_rep, T))
    expp = np.zeros((start_rep, T))
    MH_rep = np.zeros((start_rep, T))
    MH_exp = np.zeros((start_rep, T))

    ### DEMAND PROBABILITY ###
    for k in range(MaxBO + 1):
        demp.append(poisson.pmf(k, DrD))

    ### REPAIR PROBABILITY ###
    for rep in range(start_rep):
        TiR = repairdate[rep]
        for t in range(T):
            repp[rep, t] = leadprob[t + TiR]
            MH_rep[rep, t] = 0.5 * repp[rep, t] / (0.5 * repp[rep, t] + (1 -
    repp[rep, t]))
            expp[rep, t] = 2 * repp[rep, t] - repp[rep, t] ** 2
            MH_exp[rep, t] = 0.5 * expp[rep, t] / (0.5 * expp[rep, t] + (1 -
    expp[rep, t]))

    if start_rep < 0:
        sys.exit(f"System can't run for stock level in repair shop of: {
    start_rep}.")
    elif start_rep == 0:
        SDP_repp = np.full(T, 1)
        SDP_expp = np.full(T, 1)
    elif start_rep == 1:
        SDP_repp = np.zeros((start_rep + 1, T))
        SDP_expp = np.zeros((start_rep + 1, T))
        for t in range(T):
            SDP_repp[0, t] = 1 - MH_rep[0, t]
            SDP_repp[1, t] = MH_rep[0, t]
            SDP_expp[0, t] = 1 - MH_exp[0, t]
            SDP_expp[1, t] = MH_exp[0, t]
    elif start_rep > 1:
        hulparr = list(product([0, 1], repeat=start_rep))
        sumarr = np.sum(hulparr, axis=1)
        SDP_repp = np.zeros((start_rep + 1, T))
        SDP_expp = np.zeros((start_rep + 1, T))
        for t in range(T):
            for row in range(len(hulparr)):
                rv = 1
                ev = 1
                for col in range(len(hulparr[0])):
                    if hulparr[row][col] == 0:
                        rv *= (1 - MH_rep[col, t])
                        ev *= (1 - MH_exp[col, t])
                    elif hulparr[row][col] == 1:
                        rv *= MH_rep[col, t]
                        ev *= MH_exp[col, t]
                SDP_repp[sumarr[row], t] += rv
                SDP_expp[sumarr[row], t] += ev
```

E.6 General Import

```python
def ImportAllData():  # Load all data
    start_time = time.time()
    LoadAC()
    LoadToR()
    LoadDemandRepair()
    DetMaxGrid()
    ProbabilityTable()
    t = round(time.time() - start_time, 5)
    print(f"Import complete in {t} seconds")
    wb.close()
```

F | Python Code - Algorithms

F.1 General Functions

F.1.1 Reward

```python
def detReward(action):
    ### REWARD FUNCTION ###
    i_oh, i_rep, i_cust, t = nxt_state
    global bo_count, I_total, update_t
    cbuy = 0

    ### INTERVENTION COSTS ###
    if (action == 'Expedite') or (action == 'ExpBuy1') or (action == 'ExpBuy2')
    or (action == 'ExpBuy3'):
        if Sim:
            expfee = round(random.uniform(0, 450), 2)
        else:
            expfee = (450 + 0) / 2
    else:
        expfee = 0
    K = 150
    if (action == 'Buy1') or (action == 'ExpBuy1'):
        loop = 1
    elif (action == 'Buy2') or (action == 'ExpBuy2'):
        loop = 2
    elif (action == 'Buy3') or (action == 'ExpBuy3'):
        loop = 3
    else:
        loop = 0
        K = 0
    for _ in range(loop):
        if Sim:
            row = np.random.randint(0, 1)
            col = np.random.randint(1, 6)
            cbuy += CostTable[row, col]
        else:
            cbuy += ACost
    if action == 'Do Nothing':
        cbuy = 0
        expfee = 0
        K = 0

    c_int = cbuy + expfee + K

    ### BACKORDER CALCULATION ###
    if i_oh < 0:
        if t > update_t and COUNTING:
            bo_count *= 1.15
            update_t = t
        c_bo = 0.5 * bo_count * ACost * abs(i_oh) * ToR
    else:
        c_bo = 0

    ### TERMINAL COST CALCULATION ###
    if t == THORIZON:
        I_total = max(i_oh, 0) + i_rep + i_cust
        if (I_total > I_tactical) and not (I_total == max(0, start_oh) +
    start_rep + start_cust):
            c_term = ACost * (I_total - I_tactical) ** 2
        elif I_total < I_tactical:
            c_term = 0.5 * ACost * (I_tactical - I_total) ** 2 * ToR
        elif (I_total == I_tactical or I_total == start_oh + start_rep +
    start_cust) and i_oh < 0:
            c_term = 0.5 * ACost * (abs(i_oh)) * ToR
        else:
            c_term = 0
    else:
        c_term = 0

    ###TOTAL COSTS ###
    total_cost = c_bo + c_int + c_term
    return round(total_cost, 2)
```

F.1.2 Next Position

```
def nxtPosition(action, Z_rr, Z_er, Z_customer, Demand):
    ### NEXTPOSITION FOR SIMULATION AND DYNA-Q ###
    global I_total, agent_state
    i_oh, i_rep, i_cust, t = agent_state
    a = ACTIONS[action]
    buy = 0
    Z_repair = Z_rr

    if a == 'Expedite':
        Z_repair = Z_er
    elif a == 'Buy1':
        buy = 1
    elif a == 'Buy2':
        buy = 2
    elif a == 'Buy3':
        buy = 3

    elif a == 'ExpBuy1':
        Z_repair = Z_er  # New probability for repair
        buy = 1
    elif a == 'ExpBuy2':
        Z_repair = Z_er  # New probability for repair
        buy = 2
    elif a == 'ExpBuy3':
        Z_repair = Z_er  # New probability for repair
        buy = 3
    else:  # equal to "Do Nothing"
        pass

    ### BOUNDARIES ###
    usedbuy = buy
    if i_oh + Z_repair - Demand > maxinventory:
        usedrep = 0
    else:
        usedrep = Z_repair

    if max(0, i_oh) + i_rep + i_cust + usedbuy > GridMax and usedbuy > 0:
        usedbuy = max(0, GridMax - i_cust - i_rep - max(0, i_oh))
    if max(0, i_oh) + usedbuy + usedrep > maxinventory:
        usedbuy = max(0, maxinventory - max(0, i_oh) - usedrep)

    if i_oh + usedrep + usedbuy - Demand >= 0:
        useddem = Demand
        if i_oh < 0:
            custdem = Demand + min(abs(i_oh), usedbuy + usedrep)
        else:
            custdem = Demand
        if usedbuy > Demand - i_oh - usedrep and i_cust + custdem == GridMax:
            usedbuy = useddem - i_oh - usedrep
        elif i_oh + i_rep + i_cust + usedbuy > GridMax:
            usedbuy = GridMax - (i_oh + i_rep + i_cust)
    elif i_oh + usedrep + usedbuy - Demand < 0:
        custdem = max(i_oh, 0) + usedrep + usedbuy
        if i_cust + custdem > GridMax:
            useddem = i_oh + usedrep + usedbuy
        elif i_oh + usedrep + usedbuy - Demand < -maxbackorders:
            useddem = maxbackorders + i_oh + usedrep + usedbuy
        else:
            useddem = Demand

    I_total = i_oh + i_rep + i_cust + buy
    i_oh = i_oh + usedrep - useddem + usedbuy
    i_rep = i_rep - usedrep
    i_cust = i_cust + custdem

    return i_oh, i_rep, i_cust, t + 1
```

F.1.3 Initialization

```
def __init__():
    global START_STATE, Q_values, BR_values, nxt_state, agent_state, bo_count,
    COUNTING
    START_STATE = (start_oh, start_rep, start_cust, 0)
    random.seed(5)
    np.random.seed(5)
    COUNTING = False
    repair_in_shop = start_rep
    if start_q_table is None:
        for cus in range(GridMax + 1):
            if cus == GridMax:
                repair_in_shop = 0
            for curr_rep in range(repair_in_shop + 1):
                RInv = [i for i in range((-maxbackorders), min(maxinventory + 1,
     GridMax - curr_rep - cus + 1))]
                for inv in RInv:
                    for tijd in range(THORIZON):
                        Q_values[inv, curr_rep, cus, tijd] = {}
                        for a in ACTIONS:
                            if curr_rep == 0 and (
                                    a == 'Expedite' or a == 'ExpBuy1' or a == '
    ExpBuy2' or a == 'ExpBuy3'):
                                dirRew = 99999999999
                            else:
                                nxt_state = nxtPosition(ACTIONS.index(a), 0, 0,
    0, 0)
                                dirRew = detReward(a)
                            Q_values[inv, curr_rep, cus, tijd][a] = dirRew
    else:
        with open(start_q_table, "rb") as f:
            Q_values = pickle.load(f)

    if start_br_table is not None:
        with open(start_br_table, "rb") as f:
            BR_values = pickle.load(f)
```

F.1.4 Choose Action

```
def chooseAction(epsilon=0.1):
    ### DETERMINE ACTION E-GREEDY ###
    mx_nxt_reward = 9999999999

    if np.random.uniform(0, 1) <= epsilon:
        action = np.random.choice(ACTIONS)
    else:
        cur_pos = agent_state
        if len(set(val for dic in Q_values for val in Q_values[cur_pos].values()
)) == 1:
            action = np.random.choice(ACTIONS)
        else:
            for a in ACTIONS:
                nxt_reward = Q_values[cur_pos][a]
                if nxt_reward <= mx_nxt_reward:
                    action = a
                    mx_nxt_reward = nxt_reward
    if agent_state[1] == 0:
        if action == "Expedite":
            action = "Do Nothing"
        elif action == "ExpBuy1":
            action = "Buy1"
        elif action == "ExpBuy2":
            action = "Buy2"
        elif action == "ExpBuy3":
            action = "Buy3"

    return action
```

F.2 Learning

F.2.1 Backward Recursion

```
def LearnSDP():
    ### CALCULATE STOCHASTIC DYNAMIC PROGRAMMING SOLUTION ###
    global agent_state, nxt_state, bo_count, COUNTING, Sim, SDP_p, SDP_e
    # INITIALIZE SDP ARRAYS
    DP_values = {}
    DP_actions = {}
    DP_backorders = {}
    repair_in_shop = start_rep

    for cus in range(GridMax + 1):
        if cus == GridMax:
            repair_in_shop = 0
        for curr_rep in range(repair_in_shop + 1):
            RInv = [i for i in range((-maxbackorders), min(maxinventory + 1,
GridMax - curr_rep - cus + 1))]
            for inv in RInv:
                DP_values[inv, curr_rep, cus] = {}
                DP_actions[inv, curr_rep, cus] = {}
                DP_backorders[inv, curr_rep, cus] = {}
                for t in range(THORIZON):
                    DP_values[inv, curr_rep, cus][t] = 0
                    DP_actions[inv, curr_rep, cus][t] = 0
```

```python
                    DP_backorders[inv, curr_rep, cus][t] = 0

    # INITIALIZE OTHER SETTINGS FOR SDP
    COUNTING = False
    Sim = False
    bc = 1.1
    bo_count = bc ** THORIZON

    # CALCULATE TERMINAL STATE
    xlrow = 2
    repair_in_shop = start_rep
    for cus in range(GridMax + 1):
        if cus == GridMax:
            repair_in_shop = 0
        for curr_rep in range(repair_in_shop + 1):
            RInv = [i for i in range((-maxbackorders), min(maxinventory + 1, GridMax - curr_rep - cus + 1))]
            for inv in RInv:
                nxt_state = (inv, curr_rep, cus, THORIZON)
                DP_values[inv, curr_rep, cus][THORIZON - 1] = detReward(ACTIONS[0])
                DP_actions[inv, curr_rep, cus][THORIZON - 1] = 0
                DP_backorders[inv, curr_rep, cus][THORIZON - 1] = abs(min(0, inv))

    # CALCULATE OTHER TIME PERIODS WITH BACKWARD RECURSION
    for tijd in range(THORIZON - 1):
        t = THORIZON - tijd - 2
        bo_count = bc ** t #np.sqrt(t + 1)
        xlrow = 2
        repair_in_shop = start_rep
        for curr_cus in range(GridMax + 1):
            if curr_cus == GridMax:
                repair_in_shop = 0
            for curr_rep in range(repair_in_shop + 1):
                RInv = [i for i in range((-maxbackorders), min(maxinventory + 1, GridMax - curr_rep - curr_cus + 1))]
                for curr_inv in RInv:
                    best_opt_sf = 10000000
                    best_action = 0
                    for a in range(len(ACTIONS)):
                        agent_state = (curr_inv, curr_rep, curr_cus, t)
                        sum_nxt_state = 0
                        bo_nxt_state = 0
                        exp_reward = 0
                        if a == 2 or a == 5:
                            buy = 1
                        elif a == 3 or a == 6:
                            buy = 2
                        elif a == 4 or a == 7:
                            buy = 3
                        else:
                            buy = 0
                        if a == 0 or a == 2 or a == 3 or a == 4:  # if not expedite
                            rep_tab = SDP_p
                        else:
                            rep_tab = SDP_e

                        for dem in range(maxbackorders + 1):
                            for rep in range(curr_rep + 1):
                                usedbuy = buy
                                if curr_inv + rep - dem > maxinventory:
```

```
                                usedrep = 0
                            else:
                                usedrep = rep
                            ### HIER STOND DE BUY CODE
                            if max(0, curr_inv) + curr_rep + curr_cus + usedbuy > GridMax and usedbuy > 0:
                                usedbuy = max(0, GridMax - curr_cus - curr_rep - max(0, curr_inv))
                            if max(0, curr_inv) + usedbuy + usedrep > maxinventory:
                                usedbuy = max(0, maxinventory - max(0, curr_inv) - usedrep)

                            if curr_inv + usedrep + usedbuy - dem >= 0:
                                useddem = dem
                                if curr_inv < 0:
                                    custdem = dem + min(abs(curr_inv), usedbuy + usedrep)
                                else:
                                    custdem = dem
                                if usedbuy > dem - curr_inv - usedrep and curr_cus + custdem == GridMax:
                                    usedbuy = useddem - curr_inv - usedrep
                                elif curr_inv + curr_rep + curr_cus + usedbuy > GridMax:
                                    usedbuy = GridMax - (curr_inv + curr_rep + curr_cus)
                            elif curr_inv + usedrep + usedbuy - dem < 0:
                                custdem = max(curr_inv, 0) + usedrep + usedbuy
                                if curr_cus + custdem > GridMax:
                                    useddem = curr_inv + usedrep + usedbuy
                                elif curr_inv + usedrep + usedbuy - dem < -maxbackorders:
                                    useddem = maxbackorders + curr_inv + usedrep + usedbuy
                                else:
                                    useddem = dem

                            nxt_inv = curr_inv + usedrep - useddem + usedbuy
                            nxt_rep = curr_rep - usedrep
                            nxt_cust = curr_cus + custdem
                            if curr_rep == 0:
                                use_trans = dem_probs[dem]
                            else:
                                use_trans = rep_tab[rep, t] * dem_probs[dem]

                            nxt_state = (nxt_inv, nxt_rep, nxt_cust, t + 1)
                            exp_reward += use_trans * detReward(ACTIONS[a])
                            sum_nxt_state += use_trans * DP_values[nxt_inv, nxt_rep, nxt_cust][t + 1]
                            bo_nxt_state += use_trans * DP_backorders[nxt_inv, nxt_rep, nxt_cust][t + 1]

                    action_value = exp_reward + alpha * sum_nxt_state

                    if best_opt_sf >= action_value:
                        best_opt_sf = action_value
                        best_action = a
                        best_bo = bo_nxt_state

                DP_values[curr_inv, curr_rep, curr_cus][t] = best_opt_sf
                DP_actions[curr_inv, curr_rep, curr_cus][t] = best_action
```

```
                        DP_backorders[curr_inv, curr_rep, curr_cus][t] = best_bo
```

F.2.2 Dyna-Q

```
def PlayDynaQ(n_steps=0, TotalEpisodes=1000, eps=1, LR=0.1, DISCOUNT=1, EP_DECAY=1):
    global agent_state, costs_per_episode, nxt_state, bo_count, update_t, COUNTING, Sim
    __init__()
    COUNTING = True
    Sim = True
    start_eps = eps
    costs_per_episode = []

    for ep in range(TotalEpisodes):
        #if ep == 12000:
        #    eps = 0.5
        agent_state = START_STATE
        ep_cost = 0
        bo_count = 1
        update_t = 0
        repcmpt = []
        for sr in range(start_rep):
            repcmpt.append(sr)
        for t in range(THORIZON):
            if t < THORIZON - 1:
                D_rand = np.random.poisson(DrD)
                Z_rr = 0
                Z_er = 0
                n_ret = []
                e_ret = []
                for rep in repcmpt:
                    rnd = np.random.uniform(0, 1)
                    if rnd <= rep_probs[rep, t]:
                        Z_rr += 1
                        n_ret.append(rep)
                    if rnd <= exprep_probs[rep, t]:
                        Z_er += 1
                        e_ret.append(rep)
                Z_cust = np.random.binomial(agent_state[2], p_customer)
                action = chooseAction(epsilon=eps)
                best_a = ACTIONS.index(action)
                if best_a == 1 or best_a == 5 or best_a == 6 or best_a == 7:  # If best action is expedite, pop expedite list
                    for p in range(len(e_ret)):
                        repcmpt.pop(p)
                else:
                    for p in range(len(n_ret)):
                        repcmpt.pop(p)
                nxt_state = nxtPosition(ACTIONS.index(action), Z_rr, Z_er, Z_cust, D_rand)
                reward = detReward(action)
                ep_cost += reward

                Q_values[agent_state][action] += LR * (
                        reward + DISCOUNT * np.min(list(Q_values[nxt_state].values())) - Q_values[agent_state][action])

                agent_state = nxt_state
            else:
                action = ACTIONS[0]
                nxt_state = agent_state[0], agent_state[1], agent_state[2], agent_state[3] + 1
                state_actions.append((agent_state, action))
                reward = detReward(action)
                ep_cost += reward

                Q_values[agent_state][action] += LR * (reward - Q_values[agent_state][action])

            ### MODEL PREDICTION ###
            if agent_state not in model.keys():
                model[agent_state] = {}
            model[agent_state][action] = (reward, nxt_state)

            ### DYNA ARCHITECTURE ###
            for _ in range(n_steps):
                rand_idx = np.random.choice(range(len(model.keys())))
                _state = list(model)[rand_idx]
                rand_idx = np.random.choice(range(len(model[_state].keys())))
                _action = list(model[_state])[rand_idx]

                _reward, _nxtState = model[_state][_action]

                Q_values[_state][_action] += LR * (_reward + DISCOUNT * np.min(list(Q_values[_nxtState].values()) -
                                                  Q_values[_state][_action])

        eps *= EP_DECAY
        costs_per_episode.append(ep_cost)
```

F.3 Evaluating

```
def EvaluSim(TotalEpisodes=0, Method=""):
    global agent_state, bo_count, update_t, nxt_state, Sim, COUNTING, costs_per_episode
    __init__()
    Sim = True
    COUNTING = True

    Evaluation = Method
    wb = xl.load_workbook(filename="Evalu_SIM.xlsx")
    ws1 = wb[’SIM-C’] # Cost Data
    ws2 = wb[’SIM-D’] # Action Data
    ws3 = wb[’SIM-B’] # Backorder Data
    costs_per_episode = []
    for ep in range(TotalEpisodes):
        agent_state = (start_oh, start_rep, start_cust, 0)
        ep_cost = 0
        bo_count = 1
        update_t = 0
        repcmpt = []
        for sr in range(start_rep):
            repcmpt.append(sr)
        for t in range(THORIZON):
```

```python
            best_r = 10000000000
            best_a = 0
            D_rand = np.random.poisson(DrD)
            Z_rr = 0
            Z_er = 0
            n_ret = []
            e_ret = []
            for rep in repcmpt:
                rnd = np.random.uniform(0, 1)
                if rnd <= rep_probs[rep, t]:
                    Z_rr += 1
                    n_ret.append(rep)
                if rnd <= exprep_probs[rep, t]:
                    Z_er += 1
                    e_ret.append(rep)
            Z_cust = np.random.binomial(agent_state[2], p_customer)

            if Evaluation == "Exact":
                best_a = BR_values[start_ip][agent_state]
                nxt_state = nxtPosition(best_a, Z_rr, Z_er, Z_cust, D_rand)
                best_r = detReward(ACTIONS[best_a])
                best_loc = nxt_state
            elif Evaluation == "Greedy":
                for a in range(len(ACTIONS)):
                    nxt_state = nxtPosition(a, Z_rr, Z_er, Z_cust, D_rand)
                    curr_reward = detReward(ACTIONS[a])
                    if curr_reward <= best_r:
                        best_r = curr_reward
                        best_a = a
                        best_loc = nxt_state
            elif Evaluation == "Dyna-Q":
                action = chooseAction(epsilon=0)
                best_a = ACTIONS.index(action)
                nxt_state = nxtPosition(ACTIONS.index(action), Z_rr, Z_er, Z_cust, D_rand)
                best_r = detReward(action)
                best_loc = nxt_state

            if best_a == 1 or best_a == 5 or best_a == 6 or best_a == 7: # If best action is expedite, pop expedite list
                for p in range(len(e_ret)):
                    repcmpt.pop(p)
            else:
                for p in range(len(n_ret)):
                    repcmpt.pop(p)

            agent_state = best_loc
            ep_cost += best_r
        costs_per_episode.append(ep_cost)
```

G | Results - Learn

G.1 Backward Recursion

Table G.1: Translation table from interventions to abbreviations

Action	Do Nothing	Expedite	Buy1	Buy2	Buy3	ExpBuy1	ExpBuy2	ExpBuy3
Abbr.	*dn*	*exp*	*b1*	*b2*	*b3*	*eb1*	*eb2*	*eb3*

Table G.2: Optimal decisions for backward recursion with $I_{total} = 0$

oh	**rep**	**cust**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**
-3	0	0	b2	b2	b2	b2	b1	b1	b1	b1	b1	b1	exp	dn	dn	dn
-2	0	0	b1	b1	b1	b1	b1	b1	b1	b1	b1	b1	b1	dn	dn	dn
-1	0	0	exp	exp	exp	exp	exp	exp	exp	exp	exp	exp	exp	dn	dn	dn
0	0	0	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn
1	0	0	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn
2	0	0	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn
3	0	0	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn
-3	0	1	b1	b1	b1	b1	exp	exp	exp	exp	dn	dn	dn	dn	dn	dn
-2	0	1	b1	b1	b1	b1	exp	exp	exp	exp	exp	exp	dn	dn	dn	dn
-1	0	1	exp	exp	exp	exp	exp	exp	exp	exp	exp	exp	exp	dn	dn	dn
0	0	1	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn
1	0	1	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn
2	0	1	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn
3	0	1	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn
-3	0	2	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn
-2	0	2	exp	exp	exp	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn
-1	0	2	exp	exp	exp	exp	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn
0	0	2	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn
1	0	2	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn
2	0	2	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn
3	0	2	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn
-3	0	3	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn
-2	0	3	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn
-1	0	3	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn
0	0	3	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn
1	0	3	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn
2	0	3	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn
-3	0	4	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn
-2	0	4	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn

Continued on next page

Table G.2 – continued from previous page

oh	rep	cust	1	2	3	4	5	6	7	8	9	10	11	12	13	14
-1	0	4	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn
0	0	4	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn
1	0	4	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn
-3	0	5	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn
-2	0	5	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn
-1	0	5	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn
0	0	5	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn

Table G.3: Optimal decisions for backward recursion with $I_{total} = 1$

oh	rep	cust	1	2	3	4	5	6	7	8	9	10	11	12	13	14
			1	2	3	4	5	6	7	8	9	10	11	12	13	14
-3	0	0	b3	b3	b3	b3	b2	b2	b2	b2	b2	b2	b1	dn	dn	dn
-2	0	0	b2	b2	b2	b2	b2	b2	b2	b2	b2	b2	b2	dn	dn	dn
-1	0	0	b1	b1	b1	b1	b1	b1	b1	b1	b1	b1	b1	dn	dn	dn
0	0	0	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn
1	0	0	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn
2	0	0	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn
3	0	0	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn
-3	1	0	b2	eb2	eb2	eb2	eb1	eb1	eb1	eb1	eb1	eb1	exp	exp	exp	dn
-2	1	0	b1	eb1	eb1	eb1	eb1	eb1	eb1	eb1	eb1	eb1	eb1	exp	exp	dn
-1	1	0	dn	exp	exp	exp	exp	exp	exp	exp	exp	exp	exp	exp	exp	dn
0	1	0	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn
1	1	0	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn
2	1	0	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn
3	1	0	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn
-3	0	1	b2	b2	b2	b2	b1	b1	b1	b1	dn	dn	dn	dn	dn	dn
-2	0	1	b2	b2	b2	b2	b1	b1	b1	b1	b1	b1	dn	dn	dn	dn
-1	0	1	b1	b1	b1	b1	b1	b1	b1	b1	b1	b1	b1	dn	dn	dn
0	0	1	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn
1	0	1	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn
2	0	1	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn
3	0	1	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn
-3	1	1	b1	eb1	eb1	dn	dn	dn	dn	dn	exp	exp	exp	exp	exp	dn
-2	1	1	b1	eb1	eb1	eb1	exp	exp	exp	exp	exp	exp	exp	exp	exp	dn
-1	1	1	dn	exp	exp	exp	exp	exp	exp	exp	exp	exp	exp	exp	exp	dn
0	1	1	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn
1	1	1	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn
2	1	1	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn
3	1	1	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn
-3	0	2	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn
-2	0	2	b1	b1	b1	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn
-1	0	2	b1	b1	b1	b1	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn
0	0	2	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn
1	0	2	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn
2	0	2	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn
3	0	2	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn
-3	1	2	dn	dn	dn	dn	dn	dn	dn	dn	exp	exp	exp	exp	exp	dn

Continued on next page

Table G.3 – continued from previous page

oh	rep	cust	1	2	3	4	5	6	7	8	9	10	11	12	13	14
-2	1	2	dn	exp	exp	exp	exp	exp	exp	exp	exp	exp	exp	exp	exp	dn
-1	1	2	dn	exp	exp	exp	exp	exp	exp	exp	exp	exp	exp	exp	exp	dn
0	1	2	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn
1	1	2	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn
2	1	2	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn
-3	0	3	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn
-2	0	3	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn
-1	0	3	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn
0	0	3	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn
1	0	3	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn
2	0	3	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn
-3	1	3	dn	dn	dn	dn	dn	dn	dn	dn	exp	exp	exp	exp	exp	dn
-2	1	3	dn	exp	exp	exp	exp	exp	exp	exp	exp	exp	exp	exp	exp	dn
-1	1	3	dn	exp	exp	exp	exp	exp	exp	exp	exp	exp	exp	exp	exp	dn
0	1	3	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn
1	1	3	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn
-3	0	4	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn
-2	0	4	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn
-1	0	4	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn
0	0	4	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn
1	0	4	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn
-3	1	4	dn	dn	dn	dn	dn	dn	dn	dn	exp	exp	exp	exp	exp	dn
-2	1	4	dn	exp	exp	exp	exp	exp	exp	exp	exp	exp	exp	exp	exp	dn
-1	1	4	dn	exp	exp	exp	exp	exp	exp	exp	exp	exp	exp	exp	exp	dn
0	1	4	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn
-3	0	5	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn
-2	0	5	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn
-1	0	5	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn
0	0	5	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn

Table G.4: Optimal decisions for backward recursion with $I_{total} = 2$

oh	rep	cust	1	2	3	4	5	6	7	8	9	10	11	12	13	14
			1	2	3	4	5	6	7	8	9	10	11	12	13	14
-3	0	0	b3	b3	b3	b3	b2	b2	b2	b2	b2	b2	b1	dn	dn	dn
-2	0	0	b2	b2	b2	b2	b2	b2	b2	b2	b2	b2	b2	dn	dn	dn
-1	0	0	b1	b1	b1	b1	b1	b1	b1	b1	b1	b1	b1	dn	dn	dn
0	0	0	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn
1	0	0	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn
2	0	0	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn
3	0	0	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn
-3	1	0	b2	eb2	eb2	eb2	eb1	eb1	eb1	eb1	eb1	eb1	exp	exp	exp	dn
-2	1	0	b1	eb1	eb1	eb1	eb1	eb1	eb1	eb1	eb1	eb1	eb1	exp	exp	dn
-1	1	0	dn	exp	exp	exp	exp	exp	exp	exp	exp	exp	exp	exp	exp	dn
0	1	0	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn
1	1	0	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn
2	1	0	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn
3	1	0	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn

Continued on next page

Table G.4 – continued from previous page

oh	**rep**	**cust**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**
-3	0	1	b2	b2	b2	b2	b1	b1	b1	b1	dn	dn	dn	dn	dn	dn
-2	0	1	b2	b2	b2	b2	b1	b1	b1	b1	b1	b1	dn	dn	dn	dn
-1	0	1	b1	b1	b1	b1	b1	b1	b1	b1	b1	b1	b1	dn	dn	dn
0	0	1	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn
1	0	1	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn
2	0	1	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn
3	0	1	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn
-3	1	1	b1	eb1	eb1	dn	dn	dn	dn	dn	exp	exp	exp	exp	exp	dn
-2	1	1	b1	eb1	eb1	eb1	exp	exp	exp	exp	exp	exp	exp	exp	exp	dn
-1	1	1	dn	exp	exp	exp	exp	exp	exp	exp	exp	exp	exp	exp	exp	dn
0	1	1	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn
1	1	1	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn
2	1	1	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn
3	1	1	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn
-3	0	2	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn
-2	0	2	b1	b1	b1	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn
-1	0	2	b1	b1	b1	b1	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn
0	0	2	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn
1	0	2	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn
2	0	2	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn
3	0	2	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn
-3	1	2	dn	dn	dn	dn	dn	dn	dn	dn	exp	exp	exp	exp	exp	dn
-2	1	2	dn	exp	exp	exp	exp	exp	exp	exp	exp	exp	exp	exp	exp	dn
-1	1	2	dn	exp	exp	exp	exp	exp	exp	exp	exp	exp	exp	exp	exp	dn
0	1	2	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn
1	1	2	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn
2	1	2	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn
-3	0	3	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn
-2	0	3	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn
-1	0	3	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn
0	0	3	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn
1	0	3	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn
2	0	3	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn
-3	1	3	dn	dn	dn	dn	dn	dn	dn	dn	exp	exp	exp	exp	exp	dn
-2	1	3	dn	exp	exp	exp	exp	exp	exp	exp	exp	exp	exp	exp	exp	dn
-1	1	3	dn	exp	exp	exp	exp	exp	exp	exp	exp	exp	exp	exp	exp	dn
0	1	3	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn
1	1	3	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn
-3	0	4	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn
-2	0	4	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn
-1	0	4	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn
0	0	4	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn
1	0	4	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn
-3	1	4	dn	dn	dn	dn	dn	dn	dn	dn	exp	exp	exp	exp	exp	dn
-2	1	4	dn	exp	exp	exp	exp	exp	exp	exp	exp	exp	exp	exp	exp	dn
-1	1	4	dn	exp	exp	exp	exp	exp	exp	exp	exp	exp	exp	exp	exp	dn
0	1	4	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn
-3	0	5	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn
-2	0	5	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn

Continued on next page

Table G.4 – continued from previous page

oh	**rep**	**cust**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**
-1	0	5	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn
0	0	5	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn

Table G.5: Optimal decisions for backward recursion with $I_{total} = 3$

oh	**rep**	**cust**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**
			1	2	3	4	5	6	7	8	9	10	11	12	13	14
-3	0	0	b3	b3	b3	b3	b3	b3	b3	b3	b3	b3	b1	dn	dn	dn
-2	0	0	b2	b2	b2	b2	b2	b2	b2	b2	b2	b2	b2	dn	dn	dn
-1	0	0	b1	b1	b1	b1	b1	b1	b1	b1	b1	b1	b1	dn	dn	dn
0	0	0	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn
1	0	0	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn
2	0	0	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn
3	0	0	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn
-3	1	0	b2	eb2	eb2	eb2	eb2	eb2	eb2	eb2	eb2	eb2	exp	exp	exp	dn
-2	1	0	b1	eb1	eb1	eb1	eb1	eb1	eb1	eb1	eb1	eb1	eb1	exp	exp	dn
-1	1	0	dn	exp	exp	exp	exp	exp	exp	exp	exp	exp	exp	exp	exp	dn
0	1	0	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn
1	1	0	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn
2	1	0	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn
3	1	0	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn
-3	0	1	b2	b2	b2	b2	b2	b2	b2	b2	b2	b2	dn	dn	dn	dn
-2	0	1	b2	b2	b2	b2	b2	b2	b2	b2	b2	b2	dn	dn	dn	dn
-1	0	1	b1	b1	b1	b1	b1	b1	b1	b1	b1	b1	b1	dn	dn	dn
0	0	1	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn
1	0	1	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn
2	0	1	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn
3	0	1	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn
-3	1	1	b1	eb1	eb1	eb1	eb1	eb1	eb1	eb1	eb1	eb1	eb1	exp	exp	dn
-2	1	1	b1	eb1	eb1	eb1	eb1	eb1	eb1	eb1	eb1	eb1	eb1	exp	exp	dn
-1	1	1	dn	exp	exp	exp	exp	exp	exp	exp	exp	exp	exp	exp	exp	dn
0	1	1	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn
1	1	1	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn
2	1	1	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn
3	1	1	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn
-3	0	2	dn	b1	b1	b1	b1	b1	b1	b1	b1	b1	dn	dn	dn	dn
-2	0	2	b1	b1	b1	b1	b1	b1	b1	b1	b1	b1	b1	dn	dn	dn
-1	0	2	b1	b1	b1	b1	b1	b1	b1	b1	b1	b1	b1	dn	dn	dn
0	0	2	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn
1	0	2	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn
2	0	2	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn
3	0	2	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn
-3	1	2	dn	dn	dn	dn	dn	dn	dn	dn	exp	exp	exp	exp	exp	dn
-2	1	2	dn	exp	exp	exp	exp	exp	exp	exp	exp	exp	exp	exp	exp	dn
-1	1	2	dn	exp	exp	exp	exp	exp	exp	exp	exp	exp	exp	exp	exp	dn
0	1	2	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn
1	1	2	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn
2	1	2	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn

Continued on next page

Table G.5 – continued from previous page

oh	rep	cust	1	2	3	4	5	6	7	8	9	10	11	12	13	14
-3	0	3	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn
-2	0	3	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn
-1	0	3	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn
0	0	3	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn
1	0	3	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn
2	0	3	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn
-3	1	3	dn	dn	dn	dn	dn	dn	dn	dn	exp	exp	exp	exp	exp	dn
-2	1	3	dn	exp	exp	exp	exp	exp	exp	exp	exp	exp	exp	exp	exp	dn
-1	1	3	dn	exp	exp	exp	exp	exp	exp	exp	exp	exp	exp	exp	exp	dn
0	1	3	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn
1	1	3	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn
-3	0	4	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn
-2	0	4	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn
-1	0	4	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn
0	0	4	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn
1	0	4	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn
-3	1	4	dn	dn	dn	dn	dn	dn	dn	dn	exp	exp	exp	exp	exp	dn
-2	1	4	dn	exp	exp	exp	exp	exp	exp	exp	exp	exp	exp	exp	exp	dn
-1	1	4	dn	exp	exp	exp	exp	exp	exp	exp	exp	exp	exp	exp	exp	dn
0	1	4	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn
-3	0	5	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn
-2	0	5	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn
-1	0	5	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn
0	0	5	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn

Table G.6: Optimal decisions for backward recursion with $I_{total} = 4$

oh	rep	cust	1	2	3	4	5	6	7	8	9	10	11	12	13	14
			1	2	3	4	5	6	7	8	9	10	11	12	13	14
-3	0	0	b3	b3	b3	b3	b3	b3	b2	b2	b2	b2	b1	dn	dn	dn
-2	0	0	b2	b2	b2	b2	b2	b2	b2	b2	b2	b2	b2	dn	dn	dn
-1	0	0	b1	b1	b1	b1	b1	b1	b1	b1	b1	b1	b1	dn	dn	dn
0	0	0	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn
1	0	0	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn
2	0	0	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn
3	0	0	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn
-3	1	0	b2	b2	b2	b2	b2	eb2	eb1	eb1	eb1	eb1	exp	exp	exp	dn
-2	1	0	b1	eb1	eb1	eb1	eb1	eb1	eb1	eb1	eb1	eb1	eb1	exp	exp	dn
-1	1	0	dn	exp	exp	exp	exp	exp	exp	exp	exp	exp	exp	exp	exp	dn
0	1	0	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn
1	1	0	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn
2	1	0	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn
3	1	0	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn
-3	0	1	b3	b3	b3	b3	b3	b3	b3	b3	b3	b3	dn	dn	dn	dn
-2	0	1	b3	b3	b3	b3	b3	b3	b1	b1	b1	b1	dn	dn	dn	dn
-1	0	1	b1	b1	b1	b1	b1	b1	b1	b1	b1	b1	b1	dn	dn	dn
0	0	1	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn
1	0	1	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn

Continued on next page

Table G.6 – continued from previous page

oh	rep	cust	1	2	3	4	5	6	7	8	9	10	11	12	13	14
2	0	1	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn
3	0	1	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn
-3	1	1	b2	eb2	eb2	eb2	eb2	eb2	eb2	eb2	eb2	eb2	eb2	exp	exp	dn
-2	1	1	b2	b2	b2	b2	b2	b2	exp	exp	exp	exp	exp	exp	exp	dn
-1	1	1	dn	exp	exp	exp	exp	exp	exp	exp	exp	exp	exp	exp	exp	dn
0	1	1	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn
1	1	1	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn
2	1	1	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn
3	1	1	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn
-3	0	2	b2	b2	b2	b2	b2	b2	b2	b2	b2	b2	dn	dn	dn	dn
-2	0	2	b2	b2	b2	b2	b2	b2	b2	b2	b2	b2	b2	dn	dn	dn
-1	0	2	b2	b2	b2	b2	b2	b2	dn	dn	dn	dn	dn	dn	dn	dn
0	0	2	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn
1	0	2	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn
2	0	2	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn
3	0	2	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn
-3	1	2	b1	eb1	eb1	eb1	eb1	eb1	eb1	eb1	eb1	eb1	eb1	eb1	eb1	dn
-2	1	2	b1	eb1	eb1	eb1	eb1	eb1	eb1	eb1	eb1	eb1	eb1	eb1	eb1	dn
-1	1	2	b1	b1	b1	b1	b1	b1	b1	b1	b1	b1	b1	b1	b1	dn
0	1	2	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn
1	1	2	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn
2	1	2	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn
-3	0	3	dn	b1	b1	b1	b1	b1	b1	b1	b1	b1	b1	b1	b1	dn
-2	0	3	b1	b1	b1	b1	b1	b1	b1	b1	b1	b1	b1	b1	b1	dn
-1	0	3	b1	b1	b1	b1	b1	b1	b1	b1	b1	b1	b1	b1	b1	dn
0	0	3	b1	b1	b1	b1	b1	b1	b1	b1	b1	b1	b1	b1	b1	dn
1	0	3	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn
2	0	3	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn
-3	1	3	dn	dn	dn	dn	dn	dn	dn	dn	exp	exp	exp	exp	exp	dn
-2	1	3	dn	exp	exp	exp	exp	exp	exp	exp	exp	exp	exp	exp	exp	dn
-1	1	3	dn	exp	exp	exp	exp	exp	exp	exp	exp	exp	exp	exp	exp	dn
0	1	3	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn
1	1	3	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn
-3	0	4	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn
-2	0	4	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn
-1	0	4	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn
0	0	4	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn
1	0	4	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn
-3	1	4	dn	dn	dn	dn	dn	dn	dn	dn	exp	exp	exp	exp	exp	dn
-2	1	4	dn	exp	exp	exp	exp	exp	exp	exp	exp	exp	exp	exp	exp	dn
-1	1	4	dn	exp	exp	exp	exp	exp	exp	exp	exp	exp	exp	exp	exp	dn
0	1	4	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn
-3	0	5	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn
-2	0	5	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn
-1	0	5	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn
0	0	5	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn	dn

G.2 Dyna-Q

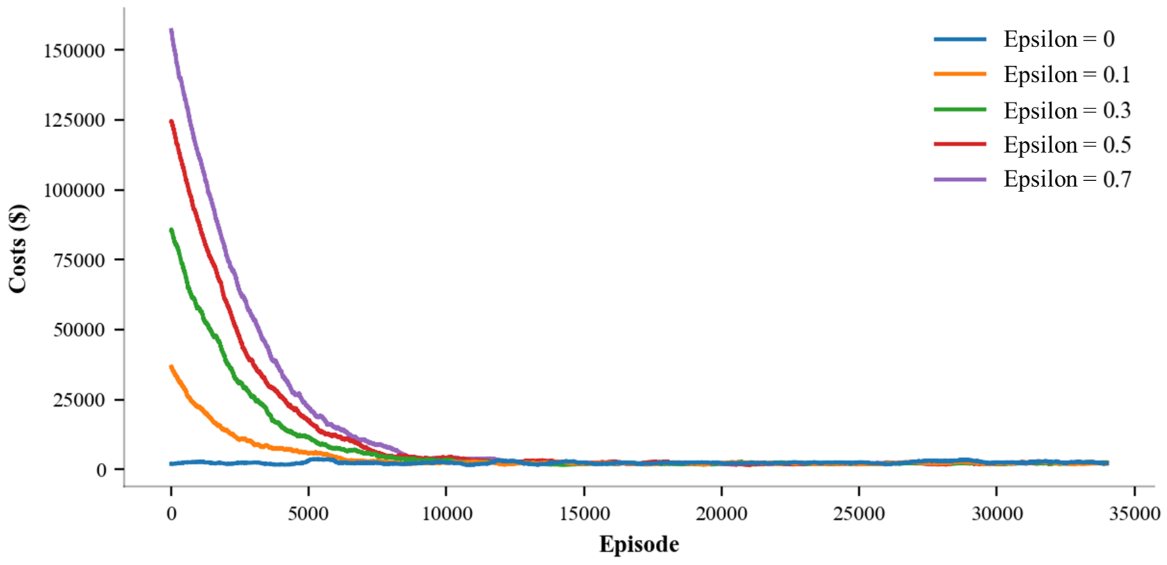


Figure G.1: Moving average of initialization of epsilon with $n = 0$ in original scenario

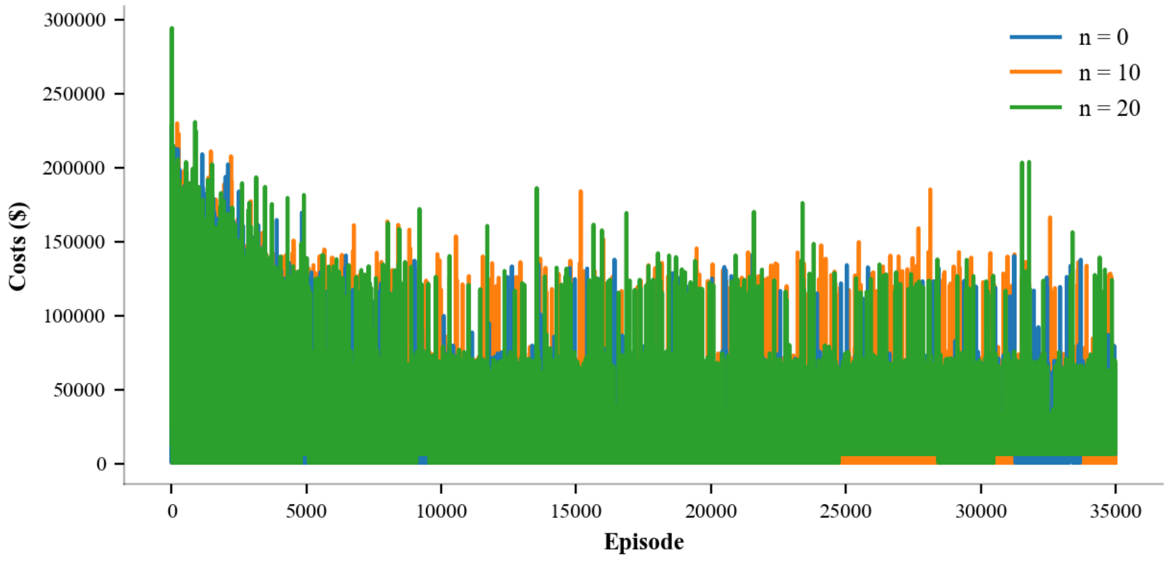


Figure G.2: Moving average of initialization of epsilon with $n = 0$ in original scenario