UNIVERSITY OF TWENTE.

Master Thesis

Exploring the potential of deep Q learning on solving the dynamic storage location assignment problem

by

Jasper Daan de Vries

Industrial Engineering and Management
Specialization Production and Logistics Management
Orientation Supply Chain and Transportation Management
Faculty of Behavioural, Management and Social Sciences

Examination committee
Dr. L. Xie
Dr. B. Alves Beirigo
University of Twente

External supervision
MSc. I. van Dongen
Supply Value

2024

Abstract

This research tackles the dynamic storage location assignment problem with a specialised deep reinforcement learning algorithm. The formulation of the dynamic storage location assignment problem and the deep reinforcement learning aim to build a bridge between theory and practice. The objective of this research is to decrease the order picking time in a warehouse by continuously optimising the storage location assignment. Periodic ABC classification is applied to create benchmark solutions, which are used as performance comparisons. The dynamic storage location assignment problem is a complex problem, that requires an intelligent algorithm to be solved. In this research, several steps are taken to present a solution to the problem at hand. First, the dynamic storage location assignment problem is formulated as a mixed-integer linear program. Then, the formulated program is translated into a Markov decision process. On top of that, a deep Q learning algorithm is formulated and programmed to tackle the formulated Markov decision process. Several experiments are created to obtain a complete perspective on the performance of the deep Q learning algorithm. Finally, the results are presented and analysed, showing that the formulated deep Q learning algorithm does not outperform the proposed benchmarks. However, the thorough analysis of the results provide concrete improvements and other future research directions, that have the potential to improve the performance in the future.

Management Summary

In this management summary, an overview of the MSc thesis is given. This thesis tackles the Dynamic Storage Location Assignment Problem (DSLAP) using the deep reinforcement learning method deep Q learning (DQN).

Objective

The main objective to be achieved by the thesis is to build a bridge between theory and practice, by addressing the DSLAP in a context that has not been explored yet. This context consists of a dynamic warehouse environment, where demand fluctuates due to seasonalities. By placing the DSLAP in this context, the results will be more applicable in practice. On top of that, the DQN implementation is adapted compared to previous researches, which makes the DQN more comprehensible both in theory and in practice. Finally, the performance of the proposed solutions are benchmarked against state-of-the-art periodic ABC classification.

Key Findings

1. **Future research directions:** Literature research shows that solving the DSLAP less frequently has great potential to reduce the distance travelled by order pickers. Order picking makes up for over 50% of the operational costs within an arbitrary warehouse. By continuously finding a viable solution for the DSLAP, these costs can be cut. However, this study area needs additional research to further identify its potential. Therefore, directions for future research are provided in this thesis.

2. **Ability of the DQN to tackle complex environments:** The complexity of placing the DSLAP in a dynamic warehouse environment causes that the DQN proposed in this research is unable to provide solutions that outperform existing benchmarks. Multiple experiments are conducted, where experiments with a smaller sample size show slightly more promising results than other experiments. Future work can build upon the presented results and analyses.

3. **Possible areas of improvement:** In this thesis, many areas of improvement are suggested to improve the performance of the DQN algorithm.

Proposed Solution

The solution presented in this thesis uses advanced machine learning methods to solve the DSLAP in a dynamic warehouse environment.

Solution Components

1. **Mixed-integer linear program:** The basic DSLAP is defined as a mixed-integer linear program (MILP), which forms the basis for the Markov decision process (MDP).

2. **Markov decision process:** Working from the MILP, the DSLAP is defined as a MDP to make it suitable to be solved by the DQN.

3. **Experimental results:** Multiple experiments are defined and executed. The results of these experiments are thoroughly analysed.

4. **Recommendations:** Using the analysis of the results, recommendations are made to continue this research. These recommendations consist of concrete improvement points and a different method to tackle the DSLAP, which is by applying a heterogeneous attention model.

Expected Outcomes

By implementing a DQN to solve the DSLAP, it is expected that the order picking costs decrease. This returns great cost savings, because the order picking costs amount to over 50% of the operational costs in an arbitrary warehouse.

Challenges and Risks

- The main challenge is configuring the DQN, to ensure that it solves the DSLAP with the desired results.

- Once the DQN shows the desired results, it requires implementation in the warehouse. This can result in potential resistance of warehouse workers, who could be afraid of being replaced by the algorithm. It is important to inform them that the machine learning algorithm is a support tool for them, not a replacement.

Next Steps

- An AI specialist is chosen to focus on the subject presented in this thesis. They can continue working, with the proposed solutions in this thesis as a basis.

- A dataset from a real warehouse must be acquired, to test the further improved DQN algorithm in practice.

Executive Summary Highlights

- This thesis takes the first step towards building a bridge between theory and practice in the specific DQN and DSLAP context presented.
- Using this thesis as a basis, concrete future steps are identified to improve the DQN.
- The thesis also suggest a different method for solving the DSLAP.

Conclusion

Tackling the DSLAP with a DQN is a challenging problem, which requires a lot of research and experimenting. The first steps to achieve this are taken in the thesis presented, by creating a DQN and conducting experiments with the DQN on the DSLAP. Finally, future improvement and research directions are discussed to achieve the potential costs savings.

Preface

Dear reader,

In front of you lies my thesis 'Exploring the potential of deep Q learning on solving the dynamic storage location assignment problem' for the MSc Industrial Engineering and Management at the University of Twente.

I would like to take a moment to thank my supervisors Dr. Lin Xie and Dr. Breno Alves Beirigo for their support and feedback throughout my graduation period. Additionally, I would also like to thank my co-supervisor MSc. Laurin Luttmann, who provided me with elaborate suggestions on the deep reinforcement learning part of this master thesis.

Furthermore, I would like to thank all my colleagues at Supply Value, in particular Iris van Dongen and Niek Wattel, for their guidance during the master thesis. I enjoyed my time spent at the office and I appreciate all the opportunities Supply Value provided that allowed me to experience the working life of a consultant.

Finally, I would like to say a huge thank you to my family and friends for always being there for me and making my entire student life successful and enjoyable.

I hope you enjoy reading this thesis!

Jasper de Vries
Enschede
August 31, 2024

Contents

List of Figures

List of Tables

List of Algorithms

Chapter 1

Introduction

Here, the motivation for the research is briefly presented, as well as the context and the gap in literature it fills. On top of that, the research objectives together with the main research question that guides this research are presented. This research question is supported with sub research questions, and all of them are answered in this paper. Finally, the significance and scope of the study are discussed, after which an overview is provided of the thesis structure.

1.1 Background and Context

This research expands the current literature of deep reinforcement learning (DRL) applied on the dynamic storage location assignment problem (DSLAP). The main goal to be achieved by this research, is to build a bridge between theory and practice. In reality, demand is often unstable because of e.g. seasonal factors (Soysal and Krishnamurthi, 2012). Due to this, papers executed using stable demand data are not representative for many practical situations.

This paper aims to build a bridge between theory and practice, by making the warehouse environment more realistic. This is done by working with unstable demand, and by using stock-out moments as period for solving the DSLAP. Solving the DSLAP for every stock-out moments ensures that companies do not need to have access to real-time data and is a realistic way of solving the DSLAP (Troch et al., 2023).

Additionally, the DRL environment presented in this paper does not contain a simulation part, to make the implementation of DRL less complex and less computationally heavy. The absence of simulation also demands less detailed information on the warehouse and its stock keeping units (SKUs). This aids to make this field of research more easily accessible for both researchers and companies, and therefore also contributes to filling the gap between theory and practice.

Currently, many efforts have been made to implement machine learning (ML) on the DSLAP (Li et al., 2019; Waubert de Puiseau et al., 2022; Troch et al., 2023; Rimélé et al., 2021). Most researches focus on supervised or unsupervised learning. However, DRL is able to grasp more complex problems and is therefore expected to revolutionise the field of artificial intelligence (AI) (Arulkumaran et al., 2017). Therefore, the author aims to expand the DRL field of literature by implementing a deep Q learning (DQN) algorithm to a different setting of the DSLAP.

As mentioned before, a different setting of the DSLAP is the focus of this research. Most DRL implementations focus on the setting of the DSLAP where the location must be determined for every incoming SKU (Waubert de Puiseau et al., 2022; Leon et al., 2023; Rimélé et al., 2021). In practice, this requires access to high quality, real-time data that is ready to be used for analysis. Unfortunately, this is often not the case in practice. Therefore, tackling DSLAPs that work with longer periods before reassignment shows potential as a balanced approach. This approach is balanced because it does not necessarily require access to real-time data, since the DSLAP is solved less frequently, while at the same time still harvesting benefits from tackling the DSLAP. Yet, solving DSLAPs with this approach has received little attention in literature.

Recently, Troch et al. (2023) were the first to apply DQN to solve this setting of the DSLAP. First, the paper from Niu and Wang (2019) has shown that DQN works on the original storage location assignment problem (SLAP), which is only solved once. Then, Troch et al. (2023) expanded these insights by showing that the DSLAP with longer periods also benefits from DQN implementations.

However, their paper builds upon stable demand data, which is not always the situation in practice. In addition to that, Troch et al. (2023) use a simulation-based DQN in the external program AnyLogic. Here, agent-based simulation is used to model the human workers, whose tasks are to pick orders and put away new incoming SKUs. Their model simulates every 30 seconds of warehouse activities, while modelling four pickers that work independently. Simulating small time steps and individual workers makes the solving environment very detailed and potentially representative if configured correctly, but also complex and computationally expensive.

This paper aims to show that the DSLAP can also be solved without a detailed simulation, while working with a warehouse environment where demand fluctuates. This is done by using a Markov decision process (MDP) programmed in Python, which replaces the simulation used in Troch et al. (2023). The MDP is easily connected to the DQN, which is also programmed in Python. To create the MDP, the library Gym is used. To create the DQN, the library PyTorch is used. Every stock-out of a product is defined as a period in the MDP, and individual order pickers and their behaviours are not included in the program.

1.2 Problem Statement

The objective of this thesis is to use a DQN without simulation to solve the DSLAP. Here, the DSLAP is solved less often than is done in the current state-of-the-art methods. Additionally, the DSLAP is surrounded by a dynamic environment. The problem is formulated as follows:

Develop a DQN algorithm that is relatively easy to implement in practice, and is used to solve the DSLAP for every stock-out to minimise the travel distance of order pickers in an environment that is representative in reality.

1.3 Research Objectives

This research has multiple different objectives. In this section, these research objectives are briefly presented. The research objectives are:

i To deliver an extensive literature review of the state-of-the-art methods to solve the DSLAP, mainly focused on DRL methods.

ii To define the DSLAP as a MDP, that is relatively accessible.

iii To apply a DQN algorithm to solve the DSLAP, with the goal to minimise the distance travelled during the order picking process.

iv To compare the DQN performance against existing benchmarks.

v To make the application of DQN on this setting of the DSLAP more easily accessible for both theory and practice.

vi To provide concrete recommendations for adaptations and future research.

1.4 Research Questions

To tackle the DSLAP, the following main and sub research questions have been formulated:

1.4.1 Main Research Question

RQ1: *'How can the DSLAP be solved by applying DQN in the context of a dynamic warehouse environment, to decrease the distance travelled during the order picking phase compared to existing benchmarks?'*

Here, it should be emphasised that the definition of a 'dynamic warehouse environment' used in this paper, is a warehouse with SKUs that show fluctuating demand. In the context presented in this paper, fluctuating demand is mainly caused by seasonal factors. Examples of SKUs that are found in a dynamic warehouse environment are e.g. fashion clothing and holiday goods (Soysal and Krishnamurthi, 2012).

1.4.2 Sub-Research Questions

To support the main research question, sub research questions are defined and answered in this paper. The basis for answering these sub research questions is literature research. In this section, the sub research questions are divided into different subjects to make them more comprehensible.

1. **DSLAP**

   - **RQ2:** *'What methods are currently used to solve the DSLAP?'*

     The DSLAP is the problem that is to be solved. Before solving this problem with DQN, first a thorough understanding of state-of-the-art methods

is required. These methods can consequently be used as benchmarks. The term 'methods' includes all types of methods, varying from heuristics to ML methods.

- **RQ3:** *'What data is used to solve the DSLAP?'*

  Data underlie the methods that are used to solve the DSLAP. Therefore, it is essential to investigate what data is useful for solving the DSLAP and how different data performs in the context of the DSLAP.

2. **Machine learning**

   - **RQ4:** *'What are the (dis)advantages of different types of ML methods when solving the DSLAP?'*

     This question is twofold, in the way that it explores both the advantages and the disadvantages. In order to support the decision of using a DRL method, first, insights should be acquired into the advantages and disadvantages of different ML methods in the context of the DSLAP.

   - **RQ5:** *'How should the DSLAP be defined to be solved with DQN?'*

     The DSLAP should be defined in a way that makes it interpretable for DQN. The representation of the DSLAP greatly depends on the chosen ML method.

3. **Performance analysis model**

   - **RQ6:** *'How are the results of the solved dynamic storage location assignment problem being analysed in current literature?'*

     To do a proper analysis of the performance of DQN on the DSLAP, state-of-the-art methods that are common in literature are used as comparison. Additionally, commonly used key performance indicators (KPIs) are identified. This increases the comparability of this work with other works, to provide more meaningful insights.

1.5 Significance of the Study

This research contributes in both a practical and a theoretical way. The practical contribution is delivered by using methods and a warehouse environment that are realistic in practice. The methods proposed are less complex and less computationally heavy compared to previously proposed methods, to make them more easily accessible in practice. On top of that, this research works with a dynamic warehouse environment, which is more realistic than an environment where demand is constant.

The dynamic warehouse environment also contributes to the theoretical significance of the study. It is not usual that a dynamic warehouse is studied in the DSLAP field. The dynamic environment is especially a new contribution for the DSLAP discussed in this paper, in combination with DQN methods. Besides that, this research focuses on a type of DSLAP that has received little attention in literature. This type of DSLAP is compared to state-of-the-art benchmarks in this research. Finally, by showing that this setting of DSLAP can also be solved with a less complex DQN, this field of study becomes more accessible and more interesting for researchers.

1.6 Scope and Limitations

Although an attempt is done to provide a study as comprehensive as possible, the scope of this research is limited. The first limitation is that this study works with a mock-up warehouse and generated datasets. Secondly, not all possible configurations of the DQN are explored. Thirdly, the order picking process is simplified because of the MDP, and distance travelled related to moving SKUs are not included in the study.

1.7 Thesis Structure

The structure of the thesis is defined as follows:

- **Chapter 2: Literature study** - An elaborate review of the current literature is provided, where the sub research questions are answered.
- **Chapter 3: Problem description** - Here, the DSLAP is formally introduced using an MILP, which is then translated into a MDP.
- **Chapter 4: Solution approach** - In this chapter, the DQN is introduced, including its configuration. Additionally, the benchmark methods are presented and exemplified.
- **Chapter 5: Experimental setup** - This chapter discusses the experimental setup, the dataset used, the performance metrics, and the hyperparameter tuning methods.
- **Chapter 6: Results and discussions** - In this chapter, the results are presented, compared and analysed. On top of that, a discussion is provided on the results
- **Chapter 7: Conclusions and recommendations** - In the final chapter, the main findings, contributions, limitations are recapped. Then, the chapter concludes with future research directions and a few brief final remarks.

1.8 Summary

The presented research questions form the basis for this research. Now that they are defined, background information on the SLAP, DSLAP, ML, and MDP is presented. Next, a thorough literature review is provided. Together, the background information and the literature review provide the basis for the further conceptualisation of the research and for answering the research questions. Thirdly, the methodology of this research is presented, followed by the results and a critical analysis of these results. In the final section, the main research question is answered, based upon the information presented in this paper. Additionally, suggestions for future research are made.

Chapter 2

Literature Study

To provide background information on the research, first the SLAP and DSLAP are introduced. Then, a general introduction to ML, including an introduction to the MDP, is provided. Thirdly, state-of-the-art methods used for solving the DSLAP are discussed. Next, the objectives of solving the DSLAP are identified, together with benchmarking tools used to test the performance. Finally, a short summary is provided, which also introduces the methods applied in this research.

2.1 Introduction to the Dynamic Storage Location Assignment Problem

The SLAP, also commonly referred to as the storage assignment problem, is a combinatorial optimization problem (Cruz-Domínguez and Santos-Mayorga, 2016). Other terms include the product/reserve allocation problem, the storage space problem or the slotting problem (Reyes et al., 2019). Frazelle (1992) defines the SLAP as the allocation of SKUs to predefined locations in the warehouse, with the objective to minimise costs. The costs that are to be minimised are the order-picking costs, which make up for over 50% of the total operational costs in the average warehouse (Pierre et al., 2003; Troch et al., 2023; Wutthisirisart et al., 2015).

The main determinant of SKU locations is 'popularity' (Sadiq et al., 1996), of which the definition and methods of measuring are not explicitly explained. Nevertheless, the relation between popularity and the storage location of an SKU is explained: more popular items are located closer to the inbound/outbound (I/O) point. The I/O point is the location where SKUs enter and leave the warehouse, and where order-pickers start and end their order-picking tour. Reyes et al. (2019) identify SKU demand behaviour as an important parameter for solving the SLAP. Demand behaviour could be considered a definition of 'popularity'.

The SLAP is proven to be an NP-hard problem, which makes it infeasible to solve to optimality (Frazelle, 1992). The SLAP can be characterised as a quadratic allocation problem when the number of SKUs is equal to the number of storage locations. It is defined as a knapsack problem if the number of SKUs is larger than the number of storage locations, and each location can store more than one SKU. Because the SLAP is NP-hard, allocation policies, heuristics and metaheuristics play an important role in solving SLAPs.

The dynamic storage allocation problem (DSLAP) builds upon the SLAP. The DSLAP is different than the SLAP in the way that it periodically executes SLAP in an iterative way (Kübler et al., 2020). Formally, this is done by expanding the SLAP to a multi-period problem. Literature suggests to periodically renew the storage layout, especially in situations where demand fluctuates or where stock changes (Cruz-Domínguez and Santos-Mayorga, 2016; Grosse et al., 2013). The periods determine the frequency with which the DSLAP is solved, and can take on different type of values. The values can be related to time, e.g. days or weeks. Periods can also be related to a different type of value, e.g. Troch et al. (2023) present a model where a period cycle is defined by stock-outs of products.

Alike the most important methods for solving the SLAP identified by Frazelle (1992), Waubert de Puiseau et al. (2022) summarise the current state-of-the-art for solving DSLAPs as a mix of heuristics, metaheuristics, and storage-policy-based methods, by applying historical data and simulations. However, at the same time a rise in ML applications is noticed (Waubert de Puiseau et al., 2022). In recent years, especially reinforcement learning (RL) has been receiving increasingly more interest, because it shows enormous potential for industrial applications (Troch et al., 2023). It is, however, still an expanding field that requires a lot of research. ML methods are further elaborated upon in Sections 2.2 and 2.3.

2.2 Introduction to Machine Learning

Under the umbrella of AI, one can find ML (Wenzel et al., 2019). ML consists of methods and algorithms that learn from experiences, to constantly improve their capabilities of extracting knowledge from data (Bertolini et al., 2021). Within the field of ML, three main areas are identified: supervised learning (SL), unsupervised learning (UL), and reinforcement learning (RL) (Bertolini et al., 2021; Wenzel et al., 2019). All three areas are discussed in Sections 2.2.1 and 2.2.2. Then, in Section 2.2.3 the Markov decision process is discussed, after which several RL methods are introduced in Section 2.2.4.

2.2.1 Supervised and Unsupervised Learning

SL is the most commonly applied ML type in literature (Wenzel et al., 2019). SL trains the algorithm on labeled example data, where input and output are known. Using the set of rules learned from this, the algorithm aims to apply these on new input data to predict the output values (Bertolini et al., 2021; Wenzel et al., 2019). The variables analysed can either be categorical or continuous. This results in either a classification task (categorical), or a regression task (continuous). Examples of commonly used methods are Neural Networks (NN), Support Vector Machines (SVMs), and Decision Trees (DTs).

Where SL methods base their knowledge on labeled training data, UL tries to find patterns in the training data by itself, by training on unlabeled data. Wenzel et al. (2019) mention that UL can also be described as 'learning without a teacher'. The main tasks in UL are clustering, density estimation, and dimensionality reduction. Bertolini et al. (2021) also briefly touch upon Deep Learning (DL), which can be a combination of finding hidden patterns and relationships in data.

2.2.2 Reinforcement Learning

RL is different than SL and UL, because it takes an action based upon a state, instead of converting an input into an output (Bertolini et al., 2021). RL takes a mathematical approach to learning, where promising decisions are rewarded and poor decisions are punished (Wenzel et al., 2019). After each decision, the current state is updated and a new decision must be made. The algorithm starts its learning process with an exploration phase, after which the exploration gradually makes place for exploitation (Alpaydin, 2021). The final goal of the algorithm is to create an agent that can be used to make decisions, by teaching it a set of rules to maximise rewards. This is done during the training process, by using a mix of exploration and exploitation techniques. Using its experience, the RL agent tries to find the optimal solution in an unrevealed environment.

The quality and quantity of experiences are an essential aspect of training a good reinforcement learning algorithm. Experience replay is used to enhance this training efficiency. The experience replay stores experiences, which can then be re-experienced at a later moment. Lin (1992) proposed this to overcome issues such as correlated updates and the minor impact of rare but important experiences. They did this with success, and therefore experience replay became an important part of the implementation of reinforcement learning techniques, see for example the ground-breaking paper by Mnih et al. (2013). The idea behind experience replay is that it provides a way of offline learning, which enhances the training speed.

In 2015, Schaul et al. (2015) proposed a more extensive version of this method: prioritised experience replay. This enables the agent to focus on experiences that are deemed more important to learn from. Certain experiences can have more impact, can be rarer, or can be more relevant because of a different reason. The method proposed in their paper prioritises experiences where the agent expects to learn most from, determined by the temporal-difference error. The main benefit from this method is that it ensures a faster training process, which is beneficial for solving the proposed MDP because of its size.

Because of its benefits, the experience replay is extended with priorities. Schaul et al. (2015) present the distributed prioritised experience replay. Priorities are assigned to every experience stored in the memory, to focus on the experiences that contribute most to the training phase. This further increases the speed of the learning process and therefore the potential of reinforcement learning.

In broad terms, two types of RL can be identified: value-based and policy-based RL (Waubert de Puiseau et al., 2022). Within value-based RL, the Q-value is computed using a value function. The Q-value represents the value of the discounted cumulative reward, which is to be optimised. Its value is dependent on the action a taken in state s. By constantly updating the value function, the Q-value of all actions in every state can be computed. Using this information, the optimal action in a state can be found. DQN is a commonly applied value-based RL method, with the application of deep neural networks.

Policy-based methods, however, aim to find the policy π that knows the most beneficial action for every state. In other words, policy-based RL directly learns the policy. This policy is a function that outputs an action for every state input. A popular policy-based method is Proximal Policy Optimisation (PPO). PPO is also based upon

the application of deep neural networks.

In the previous paragraphs, two DRL methods were introduced, DQN and PPO. By adding a SL and/or UL algorithm to support the RL agent, such as a neural network, the training process will be improved (Bertolini et al., 2021). This aids the RL agent in its performance and the agent becomes a DRL agent. For a regular RL agent, Q learning is the most popular algorithm applied. Double Q learning is an improved version of regular Q learning, that can be applied by a DRL agent.

2.2.3 Markov Decision Process

The MDP resembles the process that can be analysed by RL methods (Nian et al., 2020; Jia and Wang, 2020). A MDP is defined with states S, actions A, reward function R, a state transition matrix P, and discount factor r. Using these elements, the MDP is defined as the tuple (S, A, P, R, r). The aspects of a RL method can be directly translated into the MDP aspects. Therefore, the RL decision making process is modelled as a MDP. The following list provides a more detailed description of all elements.

- $s \subset S$: state space S, containing all possible states s.
- $a \subset A$: action space A, containing all possible actions a.
- $R \subset \mathbb{R}$: expected reward R the agents receive from taking action a in state s.
- P(s_t, r$|$x, u): state transition matrix. Contains the probabilities of moving from state s_{t-1} to s_t and receiving r after taking action a. Here, the path is only determined using information in the previous timestep $t-1$. This is the Markov property, which assumes that all information from the timesteps in the past are included in the most recent timestep.
- r : discount factor that determines the value of future rewards $(0 \leq r \leq 1)$

In a MDP, an agent starts in state s_0, where it takes action a_0. After taking the action, it moves to state s_1 and receives reward R_1. Repeating these steps and adding the discount factor, the cumulative discounted return formula at time t is created, as can be found in Equation 2.1.

$$G_t = R_{t+1} + rR_{t+2} + r^2R_{t+3}... = \sum_{k=0}^{\infty} r^k R_{t+k+1} \tag{2.1}$$

The cumulative discounted return is the value that the agent aims to maximise.

The infinity symbol is used for infinite MDPs, which introduces the next distinction: Finite and infinite MDPs can be distinguished. A finite MDP has a predetermined terminal state, which after reaching terminates the MDP. An infinite MDP continues for eternity, unless it is stopped manually. Infinite MDPs should always use a discount factor smaller than 1: $r < 1$.

Another way of distinguishing different types of MDPs is to look at observability. Fully observable MDPs (FOMDP) and partially observable MDPs (POMDP) can

be distinguished. Both are applicable in situations where transition time is known and consistent, and the transition dynamics do not change. FOMDPs are applicable when agents are put in a discrete system where all states are observable (Nian et al., 2020). This is, however, usually not the case in industrial applications. Therefore, the POMDP is introduced.

In the POMDP, the agent at time t only has a set of possible observations O, instead of states S. At time t, the input for the agent is observation o_t, which translates into a probability matrix of possible current states. Solving POMDPs to (near-)optimality is a NP-hard problem, due to the unknown states.

To enable solving to optimality using the current information, belief states B are introduced in Nian et al. (2020). Using past observations and actions, an agent creates a belief state b probability matrix, which contains all possible states and their accompanying probability, which should be interpreted as how likely the agent thinks it is that it is in state s. Then, the value function of each state can be formulated to determine the best action in each state. Multiplying the state probabilities with the reward values of action a in state s, the best action can be determined independent of knowing in what state the agent exactly is. This transforms the POMDP into a FOMDP, where the belief states replace the real states.

2.2.4 Reinforcement Learning Methods

Nian et al. (2020) define three branches of reinforcement learning: Dynamic programming (DP), Monte Carlo (MC) methods and temporal difference (TD). All three are briefly discussed in this section. Additionally, an introduction to function approximation methods (FAM) is provided.

Dynamic Programming:

DP is able to find optimal solutions for perfect model MDPs, but are often limited by high computational times. Due to this constraint, they are usually not used in practice. Nian et al. (2020) propose policy iteration and value iteration as the most common DP methods.

Policy iteration aims to find the optimal policy by iteratively executing two steps: policy evaluation and policy improvement. This allows policy iteration to consider a large number of policies. First, the value function of a policy is defined. The value function starts as 0, and is updated by applying the policy in many different situations. Secondly, when a different policy outperforms the current optimal policy, during the policy improvement step this different policy is defined as the new optimal policy. This iterative process continues until the algorithm is unable to find a more optimal policy.

Value iteration is only possible when the MDP is a perfect model. This implies that the state transition probabilities can be derived, which can be used to determine the action in every state that leads to the highest reward. These actions then together form the optimal policy.

Monte Carlo Methods:

MC uses a large number of runs to consider many situations, which contributes to finding the average optimal policy. MC does not need a modelled version of the system, and therefore exploration is mandatory. The idea behind MC is to estimate the average return of several policies by executing them on a large number of sampled situations. They are mostly suitable for finite processes. When the process is infinitily long, a pre-determined end point should be defined. Nian et al. (2020) emphasise that MC does not use bootstrapping, which guarantees that the estimated value functions created with MC methods are all independent and unbiased. MC methods are, however, sensible to data with a large variance.

Temporal Difference:

Here, the idea is to combine both DP and MC in one algorithm, to achieve the best of both worlds. TD methods are relatively simple to implement and computationally light. Just like MC methods, TD does not need a modelled version of the system. However, since TD estimates values based upon earlier estimated values, their results are usually biased. The idea is to learn from experience while bootstrapping, which is data efficient but causes this bias to appear.

The most common TD approaches are Q learning and state-action-reward-state-action (SARSA). Both update the value function directly after every action and sometimes decide to perform an exploration step. SARSA is an on-policy agent, which could cause it to stop exploring during the training phase, once the optimal policy is found. Because of this, it can end up in a local optimum that is not close to the actual global optimum value. Q learning is defined as an off-policy method, which ensures it keeps exploring during the training phase. Then, in the test phase it starts using its found optimal policy. Another advantage of an off-policy method is that it has the potential of finding the optimal policy under specific conditions.

Function Approximation Methods:

Problems that occur in industry are often multi-dimensional and continuous. RL methods are less able to thoroughly analyse these type of problems, which makes these problems less suitable to be solved by RL. However, a function approximation methods (FAM) can be applied to overcome this problem. Therefore, supporting RL applications with a FAM is a common phenomenon in industry.

The main idea of a FAM is that the value function is approximated. This allows the RL to approach the optimal solution, even in situations with high-dimensional and continuous states and actions. SL methods are applied to approximate the value function. A simple example of FAM is explained in the next paragraph.

The value function approximation $\hat{v}(s, w)$ should focus on the most important states, which can be determined by comparing the fraction of time spent in state s. Both state vectors and weight vectors are introduced here. A state vector consists of all features f that define that state, while the weight vector w is defined as all weights attached to those features. The total value of the weight factor is the weight assigned

to that state. Using a mean squared value error function as an objective function, the weight vector of each state vector is determined. Here, the SL method comes into play. which transforms the RL algorithm into a DRL algorithm (Bertolini et al., 2021). As an example of a SL method that can be used here, Nian et al. (2020) mention the stochastic gradient descent (SGD). Then, the value function approximation is defined as follows:

$$\hat{v}(s, w) = \mathbf{w}^T \mathbf{f}(s) \tag{2.2}$$

As a simple basis function, Nian et al. (2020) introduce the polynomial function. It is both flexible and intuitive in its implementation, which makes them suitable for easy implementations. The structure of a polynomial basis function is presented in Equation 2.3, as found in Nian et al. (2020).

$$y = w_1 s_1 + w_2 s_2 + w_3 s_1^2 + w_4 s_2^2 + ... + w_n s_1^{\frac{n}{2}} + w_{n+1} s_1^{\frac{n}{2}+1} + w_{n+2} s_1 s_2 + w_{n+3} s_1^2 s_2 + ... + w_0 \tag{2.3}$$

2.3 State-of-the-art Methods for Solving the Dynamic Storage Location Assignment Problem

DSLAPs are commonly converted into mixed-integer linear programs (MILPs) (Reyes et al., 2019). However, these MILPs are often not solved using exact methods, because it is extremely time-consuming (Reyes et al., 2019). This is due to the DSLAP being NP-hard, as mentioned before. Because of this, storage policies and rules, heuristics, metaheuristics and simulation techniques are commonly used to solve the DSLAP, instead of exact methods (Reyes et al., 2019). Additionally, the DSLAP is also being solved using ML methods.

2.3.1 Policies

There are four main storage allocation policies: random storage, semi-random storage, fixed location storage and class-based storage (Cruz-Domínguez and Santos-Mayorga, 2016). As the term insinuates, the random storage policy randomly assigns SKUs to one of the available storage locations. The semi-random storage policy is different in the way that it has a bias towards available locations that are closer to the I/O point. Other than that, SKUs are still randomly allocated. Under the fixed location policy, all SKUs have one predetermined storage location and cannot deviate from that.

Finally, the class-based storage policy groups SKUs in classes. Within their class, SKUs are stored at random (Petersen et al., 2004). They limit their class-based storage policy to two (AB), three (ABC) and four (ABCD) classes. In the two-class system, a 30-70 division is used for areas A and B. This means that the 30% of SKUs with the highest throughput are assigned to area A, the rest is assigned to area B. For the remaining two policies, they apply 20-30-50 and 10-20-30-40 divisions.

Petersen et al. (2004) show that almost 80% of maximum performance can be achieved by using a two-class policy. Here, maximum performance is defined as the performance achieved by a volume-based storage policy, which assigns *all* SKUs with their own storage location, based upon their demand. The ABC policy achieves 90% of this

performance, while the ABCD policy achieves 94%. All three policies achieve significantly better results than the random storage allocation policy. The partitions used for class-based storage policies differ per study, see e.g. Li et al. (2016).

2.3.2 Heuristics

A large diversity of heuristics are applied to solve DSLAPs in literature. Examples range from sequencing procedures, multi-stages procedures, and hierarchical procedures to multi-product optimisation heuristics, 2-top exchange heuristics and greedy heuristics (Reyes et al., 2019). Two types of heuristics are discussed in this section. Wutthisirisart et al. (2015) show a state-of-the-art multi-stage procedure, whereas the heuristic introduced by Frazelle (1992) is one of the first algorithms known in the field of SLAP.

Wutthisirisart et al. (2015) propose a two-phased heuristic that apply order frequency and order size to determine the relationship between SKUs. In the first phase, linear item sequencing places items that are highly related to other items high in the sequence, after which the second phase determines the storage location of each SKU using the linear item sequence provided. The distance between the available storage locations and the storage location of the most recently assigned SKU determines the best location for the current SKU, to ensure the link between SKUs stays intact. The goal here is take the influence of orders on the SKU allocation into account, because an order is not finished until the last item is picked.

A second heuristic is the cluster-first zone-second algorithm by Frazelle (1992), which consists of two phases: the clustering of SKUs and assigning these clusters a warehouse location. During the first phase, the aim is to cluster SKUs together that have a high probability to be on the same order. More popular items gain priority with this clustering. The second phase aims to locate the clusters that are expected to be the most popular as close as possible to the I/O point.

2.3.3 Metaheuristics

The most common metaheurstics in literature to solve the DSLAP are tabu search (TS) and genetic algorithms (GA). Other metaheurstics, such as located iterated search and simulated annealing are included in research as well (Reyes et al., 2019).

TS is used as an addition on other heuristics, which uses memory to prevent them from getting trapped in local optima (Glover, 1990). A list of a limited number of past states is saved, to force the heuristic to explore into new directions. The list of states is consistently updated, where formerly visited states are deleted and newly visited states are added. Besides solving the DSLAP, other applications for the TS are employee scheduling, travelling salesman problem, and neural network pattern recognition (Glover, 1990).

GAs are algorithms that make are based upon evolutionary concepts (Lambora et al., 2019). The tasks applied in a GA are selection, crossover, and mutation (Lambora et al., 2019). GAs are able to find short paths to optimality, and can be used for machine learning purposes as well (Lambora et al., 2019). Examples of metaheuristics within the DSLAP field are the GAs developed by Pan et al. (2015) and Li et al. (2016). Pan et al. (2015) their GA aims to optimally solve the SLAP, while balancing the workload

of the different zones in the warehouse. Li et al. (2016) developed a greedy GA to determine the warehouse layout using ABC classification.

2.3.4 Simulation

Within simulation, two methods have been researched: discrete event simulation (DES) and agent-based simulation (AB) (Reyes et al., 2019). Here, the application of AB is a more recent development. Troch et al. (2023) defines DES as a *'technique [that] focuses on the modelling of processes using a sequence of separate discrete events, allowing for irrelevant physical details to be abstracted away'*. AB simulation is characterised by a set of agents that interact with each other and their environment in a simulation model (Franzke et al., 2017). AB simulation is applicable for simulations where individual agents interact naturally, where one wants to learn from their behaviour and adapt it, or where the situation or environment the agents are in affects their behaviour.

Franzke et al. (2017) show an application of AB simulation on a warehouse case study. In their research, the goal is to optimise the order picking processes under different storage assignment and order picker-route circumstances. ABS is applied to simulate and observe the behaviour of individual order pickers.

2.4 Supervised and Unsupervised Learning Applied on the Dynamic Storage Location Assignment Problem

Both SL and UL are relevant machine learning methods for industrial processes. Especially SL has been extensively research in the past two centuries. Where SL methods aim to make predictions, the goal of UL is to detect and extract patterns in data (Bertolini et al., 2021). Besides applying them individually, both can be implemented to support RL as well. Some SL methods, such as decision trees, provide the advantage that the decisions made by the agent are transparent and explainable (Berns et al., 2021). Another advantage of SL includes that it is more likely to take human-like decisions, because human input is used to train the algorithm (Mahraz et al., 2022). UL provides the advantage of being able to be its own teacher, which is time efficient (Wenzel et al., 2019).

In industrial applications, SL is commonly applied in four areas: Maintenance management (MM), quality management (QM), production planning and control (PPC), and supply chain management (SCM). UL has only been applied in MM, QM, and PPC. Applications of UL in SCM are not found in the comprehensive literature review done by Bertolini et al. (2021). Examples of SL methods are decision trees, neural networks, and logistic regression. Bertolini et al. (2021) divide UL into three broad methods: clustering, density estimation, and dimensionality reduction.

An example of SL applied on DSLAP is found in Li et al. (2019). They apply a combination of two neural networks, namely Residual Deep Convolutional Networks and Gated Recurrent Unit networks to predict the duration-of-stay (DoS) of incoming SKUs. Using the DoS information, the exact allocation of incoming SKUs is done. Besides the SL method, they include a simulation software to simulate the performance of the system. They implemented the method in multiple warehouse in the USA, where results showed that labour costs were cut by 21%.

Yang and Nguyen (2016) their study show an application of UL in solving the SLAP. They apply principal component analysis (PCA) to define clusters in SKU data, which can be used to assign SKUs to the right class. They argue that independent of the size of the warehouse, their algorithm is always able to assign SKUs to the right class based on their characteristics. Therefore, the algorithm is able to handle large datasets that are intractable for humans. Their simulation results show an increase in retrieval efficieny of 33%.

2.5 Reinforcement Learning Applied on the Dynamic Storage Location Assignment Problem

RL is suitable for sequential decision-making tasks, where decisions are impacted by decisions made prior to themselves (Alpaydin, 2021). The DSLAP is an example of this, because it can be modeled using DES (Reyes et al., 2019). According to Waubert de Puiseau et al. (2022), applying RL to solve the DSLAP saves time compared to SL, because RL does not require labelling. This makes the RL agent also more easily generalisble, because once the RL agent is trained, new datasets should only be transformed into the right format before the RL agent starts optimising the warehouse layout using the newly provided data.

Waubert de Puiseau et al. (2022) created a DRL algorithm that can be applied on the DSLAP on combination with a separate simulated environment. The goal is to classify each incoming SKU based upon real-time data. They make use of the ABC-classification heuristic, and real-time SKU and storage data as input. The DRL method applied here is Q learning. More specifically, a policy-based method is used, namely the action-masked version of the PPO. After thorough testing, the results show a significant decrease in transportation costs of 6.3%. In their case study, the method is applied to a semi-automatic high-bay warehouse with two automatic storage and retrieval systems. Moreover, the used RL algorithm is generic and therefore can be applied to other industries as well.

In the paper of Leon et al. (2023), three different RL algorithms are compared to each other and to another benchmark algorithm. They apply the RL algorithms on the DSLAP with ABCD classification, where the location assignment decision is made for every SKU that enters the system. The warehouse for the simulation study is a warehouse where orders are picked manually. The three different algorithms reviewed are: advantage actor critic (A2C), PPO, and DQN. The results show that in general, all three algorithms perform as well as the greedy algorithm they are compared with. In the small experiment, the A2C algorithm slightly outperforms the others based on distance travelled. The large experiment shows that the DQN algorithm has the best performance in terms of distance. When comparing the performance of the algorithms in the small and large problems, the DQN algorithm shows the smallest drop in performance. This implies that it is more consistent in its performance when the size of the problem changes. Leon et al. (2023) suggest to apply different RL algorithms and to benchmark their performances.

Troch et al. (2023) implement RL to solve the DSLAP in an E-commerce distribution center where manual order picking is conducted. They advise to run their optimisation model every night, to revise the warehouse layout and cope with the fluctuating demand. They, however, only implement their algorithm in an environment with con-

stant demand. The DSLAP is modeled as a hybrid simulation, combining sequential decision making and agent-based simulation. 'popularity' is applied as input. The definition of 'popularity' is defined as the number of times SKU x occurred during the previous timestep. The specific applied RL method is DQN, more specifically Ape-X with action-masking. The results show that the RL algorithm can in most cases achieve at least 80% of the optimal solution, starting from any randomised initial warehouse layout. Troch et al. (2023) intend to expand their research by applying it in an environment with fluctuating demand in the future.

Rimélé et al. (2021) apply a RL algorithm on the DSLAP in an e-commerce warehouse, where a robotic mobile fulfillment system is in use. In the warehouse, the items enter the warehouse one-by-one and are batched within the warehouse. An class-based approach is taken to assign all incoming SKUs to a class. The DSLAP is defined as a partially observable Markov decision process, where the warehouse layout is determined by grouping the SKUs using a class-based method. The goal is to decrease the average cycle time of SKUs, which is directly related to the maximal throughput capacity. Rimélé et al. (2021) apply double differential DQN, that uses two neural networks, in combination with a look-ahead strategy using Monte Carlo Tree search. The look-ahead strategy increases the information available for their method, and therefore improves their method. After testing on a case study, the method, including look-ahead strategy, showed a decrease in travel times up to 14%. Rimélé et al. (2021) mention using exact storage allocation instead of zones for future research.

Niu and Wang (2019) define a SLAP as a partially observable Markov decision process and apply model-based DRL on a mock-up warehouse. Their model makes a trade off between reduction in travel costs and reposition costs. Here, they make use of a prediction model, an long short-term memory (LSTM) network with 1,000 lookback, which in turn is trained on a demand predictor to handle the large number of dimensions included in the problem. The LSTM network contains a long short-term memory, and applies value iteration with a feedforward network function (FFNN). The method is tested on stable and unstable (dynamic) demand. In the case with stable demand, the model shows a good performance. In the second case, the method requires requires a few timesteps to readjust to the dynamic environment. After is is adjusted, it performs sensibly well. Additionally, the model performs well under changes in scale.

2.6 Objectives of the Dynamic Storage Location Assignment Problem

The order picking process in a warehouse can be improved by optimising either the storage allocation and routing, or both (Dixit et al., 2020; Pierre et al., 2003). The order picking time is the most important KPI that is used to measure the performance of the order picking performance. The order picking time consists of four parts: the walking or driving time back and forth to the SKU location, the time spent to pick up the item, the acceleration/deceleration time and the time spent on administrative tasks. Waubert de Puiseau et al. (2022) in their paper focus on the first part, reducing the travel time. Leon et al. (2023) explain that travel distance is a common KPI in DSLAP research.

In their literature review on the SLAP, Reyes et al. (2019) compare different KPIs

in terms of how often they are used in literature. The most used KPIs are travel distance and travel time. Together, they are used in 70% of the articles reviewed. Starting from 2015, human factors and infrastructure are also being considered. Other upcoming KPIs are consumption and energy efficiency. They conclude that, besides the optimisation of travel distance and time, human and environmental factors are also increasingly being included in new studies.

In this paper, the objective is to minimise the travel distance within the warehouse. As mentioned before, both the travel distance and travel time is used in almost three-fourths of the researches. Travel distance and travel time are directly correlated. However, travel distance is more easily measured, because a distance matrix can be used. In order to obtain the travel time, the distance matrix must be converted into a time matrix. Travel time, however, might also be dependent on external factors like the individual pickers and the item at hand. Therefore, it is decided to choose minimise travel distance, which is more objective compared to travel time and does not require an additional conversion step.

2.7 Benchmarking the Dynamic Storage Location Assignment Problem

To compare the performance of different models, benchmarking is required. Different benchmarks are used in literature. The examples discussed in this section are taken from papers that apply machine learning on the DSLAP and use benchmarks for performance comparison.

Waubert de Puiseau et al. (2022) make use of four different benchmarks. Their first and simplest benchmark is 'Random', which randomly assigns an item to zone A, B or C. Just-in-Order is their second benchmark. With this benchmark, first zone A is filled, then zone B is filled and once zones A and B are fully utilised, the filling of zone C starts. Thirdly, the regular ABC classification is applied. SKUs are assigned to one of the classes, based upon expert opinions. Finally, with DoS-Quantiles, a benchmark is used that can only be created after all product data in a period is known. DoS is split into three parts: SKUs in the first quantile (with the lowest DoS) are assigned to zone A, SKUs with a DoS between quantile 1 and quantile 2 are assigned to zone B, the other SKUs are assigned to zone C.

Troch et al. (2023) propose to use a genetic algorithm from literature as their benchmark. The reason for this is because their RL method, in combination with its simulated environment, is most similar to the field of meta-heuristics. Because of this, they compare their results to a meta-heuristic, in the form of the genetic algorithm created by Grznár et al. (2021).

Grznár et al. (2021) their genetic algorithm works as follows: First, an initial warehouse layout containing all SKUs is created. Then, this layout is tested in a simulated environment on its performance. In Troch et al. (2023), best 20% performing layouts are selected to be evolved further using crossover and mutation. The permutations are determined based on the Cycle Crossover algorithm. Mutations are implemented using a probability of 0.1.

Leon et al. (2023); Niu and Wang (2019) use randomised layout as their benchmark, where all SKUs are randomly assigned to a location. This is different than the randomised layout in Waubert de Puiseau et al. (2022), where SKUs are assigned to

class. Leon et al. (2023) use a second benchmark in the form of a greedy heuristic as an addition to the first benchmark. This greedy heuristic places the item in the closest available location. Therefore, it aims to decrease the distance travelled as much as possible and serves as a useful performance metric (Leon et al., 2023).

Rimélé et al. (2021) benchmark their performance using three policies: Random, Class-based, Shortest-Leg. Similar to the method applied in Leon et al. (2023), the random policy assigns SKUs to a random location with equal probabilities. The class-based policy applied by Rimélé et al. (2021) is similar to the ABC policy applied in Waubert de Puiseau et al. (2022). With the shortest-leg policy, a new benchmark policy is introduced. This policy results in the lowest average travel time in dual-command cycles (Rimélé et al., 2021). For each SKU, the location that minimises the sum of the distance from the I/O point to the storage location and the distance from the storage location to the retrieval location, is assigned.

In this paper, two benchmarks are applied. First, the random heuristic is executed to obtain a simple lower bound. Second, the ABC classification with different periods is applied. It is shown that an ABC classification is an easy way to achieve 90% of the optimal performance (Petersen et al., 2004). For both benchmarks, periodic optimisation is excluded. The data analysed to determine the benchmarks is the entire dataset.

2.8 Summary

After a thorough literature review, it can be concluded that RL methods applied on the DSLAP is still a strongly growing field. Multiple academics have contributed to its current state-of-the-art, however, they all propose research gaps that are yet to be filled.

This research builds upon current research on DQN. More specifically, this research aims to build upon the paper of Troch et al. (2023), who mention that their periodic DSLAP research with DQN should be tested on an environment with fluctuating demand. In addition to that, the presented research attempts to do this without any additional simulation software, in contrast to Troch et al. (2023).

Most DQN research on the DSLAP is designed in a way where a location decision must be made for all incoming SKUs. This is shown to have the potential to decrease the distance travelled by the order pickers. However, with this strategy it is likely that there will be SKUs with multiple storage locations assigned to them, which is not necessarily desirable. It is for example undesirable when many units of a single SKU need to be picked. In that case, the order picker may need to travel to multiple storage locations for one SKU, which actually results in more distance travelled. To add on that, the most recent data is not always continuously available, which could result in decisions being made on outdated data.

In an attempt to overcome these drawbacks, this research aims to expand the current literature on DQN applications on the DSLAP, where the location decision for SKUs is made less frequently and SKUs are placed at one single storage location. On top of that, it focuses on a dynamic warehouse environment, where demand fluctuates.

The execution of this is done in Python, without applying an external simulation software. The reason for this being, firstly, that an external software slows down the

algorithm. Secondly, the aim is to create a tool that is relatively simple and accessible for users in practice.

An overview of the characteristics of the papers found during literature review and this paper is presented in Table 2.1. The papers in the table are first sequenced based upon their ML method. DRL methods are placed as low as possible, with 'DQN' given priority. That is, papers that apply DQN are placed as low as possible. Then, whether the paper solves the SLAP or the DSLAP determines the sequence of the table, where 'DSLAP' receives priority. Next, the period length is used to determine the priority, where 'Every stock-out' receives the most priority and 'Once' receives the least priority. The demand determines the final priority, with 'Yes', which stands for unstable demand, given most priority.

Table 2.1: A literature review table.

Paper	Mock-up warehouse	(D)SLAP	Period length	Unstable demand	ML method	Instance size
Berns et al. (2021)	No	SLAP	Once	n.s.	DT	12 million entries
Yang and Nguyen (2016)	No	SLAP	Once	n.s.	PCA	n.s.
Li et al. (2019)	No	DSLAP	Every SKU	Yes	NN with simulation	8.5 million entries
Waubert de Puiseau et al. (2022)	No	DSLAP	Every SKU	n.s.	PPO	12.100 entries, 500 SKUs
Niu and Wang (2019)	Yes	SLAP	Once	Yes	DQN	100 products, 10x10 warehouse
Leon et al. (2023)	Yes	DSLAP	Every SKU	n.s.	A2C, PPO, DQN with simulation	241 entries in simulation
Rimélé et al. (2021)	No	DSLAP	Every SKU	Yes	DQN with Monte Carlo Tree search	n.s.
Troch et al. (2023)	Yes	DSLAP	Every stock-out	No	DQN with simulation	160 SKUs
This work	Yes	DSLAP	Every stock-out	Yes	DQN	270 million entries, 150 SKUs

[a] n.s. = not specified

Chapter 3

Problem Description

An elaborate explanation of the use case is presented in this section. First, information on the warehouse is presented. Secondly, the data is explained and the features are identified. Thirdly, the warehouse is presented as an MDP, including all the characteristics that define the MDP. Then, the benchmark tools are shown. Finally, the DQN algorithm is explained in-depth.

3.1 Assumptions

In this paper, the following assumptions are made:

1. The warehouse has a square shape.
2. There is one single input/output (I/O) point.
3. All orders are picked in time on the day they are demanded.
4. As an order picking approach, the picker-to-parts approach is used.
5. Replenishments are ordered, delivered and stored immediately after a stock-out without any travelled distance or costs.
6. Travel times for grabbing the product from the storage location are assumed to be equal for all locations and SKUs and are therefore not included.
7. All storage locations are homogeneous, and every SKU can be stored in each storage location.
8. The storage capacity of all SKU and location combinations are equal.

3.2 Definition

Here, the foundation of the DSLAP is defined, by defining its objectives and parameters. It is the basis for the exact formulation, presented in the rest if this chapter.

3.2.1 Problem Objectives

The main objective in the presented problem is presented here.

- **Minimise Total Travel Distance**: The aim is to minimise the distance travelled of the order pickers. This aids to reduce order picking costs and also improves the throughput time.

3.2.2 Problem Parameters

To create a model that can achieve the main objective, multiple parameters are defined.

- **Storage Locations**: These are the locations at which SKUs can be stored.
- **SKUs**: The SKUs are the items that are to be stored in storage locations.
- **Demand**: Every time step, demand occurs for the SKUs. The demand is one of the main determinants for the storage location of the SKUs.
- **Time steps**: The time steps represent the time window of the problem.
- **I/O Distance Matrix**: This distance matrix contains the distance of each storage location to the I/O point.

The combination of these parameters form the problem at hand.

3.3 Mathematical Formulation

In this research, the basic DSLAP is formulated as a MILP. This is used as the basis for the MDP formulated in Section 3.4. As mentioned in Section 2.1, the DSLAP is formulated by expanding the mathematical formulation of the SLAP to a multi-period model.

The DSLAP consists of T time steps and a given warehouse with a set of storage locations L. A set of SKUs I is to be stored in the warehouse. To determine the location assignment, the distance from every location l to the I/O point, which is defined as d_l, and the demand of item i in time step t, defined as dem_{it}, are used as decision criteria. Every SKU also contains an inventory in t, inv_{it}, and a maximum storage capacity cap_i.

The decision variable is defined by x_{ilt}, which is a binary value that has a value of 1 if i is assigned to l in t. If not, its value is 0. The values of x_{ilt} are defined by solving the DSLAP and x_{ilt} can have different values for every i, l, and t combination.

The objective is to minimise the distance travelled to pick all demand over T, see Equation 3.1.

To comply with the constraints, every i can only be assigned to one l and every l can have only one i assigned to it. This is defined in equations 3.2 and 3.3. On top of that, for every i and every t, the current inv_{it} must be able to fulfil the dem_{it}, but it may not exceed the cap_i. For this, Equations 3.4 and 3.5 are created.

Finally, the binary constraint is defined for the decision variable x_{ilt} in Equation 3.6.

Sets:

- $i \in I$: set of SKUs

- $l \in L$: set of storage locations
- $t \in T$: set of time steps within time horizon T

Parameters:

- d_l : distance from I/O point to location l
- dem_{it} : demand of SKU i in time step t
- inv_{it} : inventory of SKU i in time step t
- cap_i : storage capacity of SKU i

Decision variables:

- $x_{ilt} \in \{0, 1\}$: 1 if SKU i is assigned to storage location l in time period t, 0 if not

The following MILP is obtained:

$$\text{Minimize:} \quad \sum_{i=0}^{I} \sum_{l=0}^{L} \sum_{t=0}^{T} x_{ilt} * dem_{it} * d_l \tag{3.1}$$

Subject to:

$$\sum_{i=0}^{I} x_{ilt} = 1, \qquad \forall l \in L, t \in T \tag{3.2}$$

$$\sum_{l=0}^{L} x_{ilt} = 1, \qquad \forall i \in I, t \in T \tag{3.3}$$

$$inv_{it} \geq dem_{it}, \qquad \forall i \in I, t \in T \tag{3.4}$$

$$inv_{it} \leq cap_i, \qquad \forall i \in I, t \in T \tag{3.5}$$

$$x_{ilt} \in \{0, 1\} \tag{3.6}$$

3.4 DSLAP as Markov Decision Process

To use the DSLAP for DQN purposes, it must be converted into a MDP. In this section, the MILP of the DSLAP is mapped to a MDP.

Using the MDP, the agent tries to optimise a given layout iteratively. Starting from a random initial configuration s_0, the agent experiences an entire year of warehouse activities. Here, every t represents a day, so $T = 365$. An entire year is taken while training to ensure that all seasonalities are experienced.

Multiple episodes E of training are executed during the training process. Every episode e is an individual run, and therefore uses a new dataset, which contains the same SKUs and uses the same seasonality to generate the demand values. In total, 100 episodes of training are conducted. Thus, $E = 100$.

In the MDP, the sets I, L, and T are still used. Likewise, all parameters d_l, dem_{it}, inv_{it}, and cap_i are still in use. In the MDP, SKUs can only be relocated in t if they are to be restocked in t. Therefore, every stock-out moment is defined as an iteration of

the MDP. To model this, the total number of iterations every t, $ITER_t$, is added as a parameter. Every t has a value $ITER_t$, ranging from 0 to I. $ITER_t$ is equal to the total number of stock-outs in t. Every i in t can have either no or one stock-out so_{it}, and $ITER_t = \sum_{i=0}^{I} so_{it}$. The value of so_{it} is defined as in Equation 3.7.

$$so_{it} = \begin{cases} 0, & inv_{it} > dem_{it} \\ 1, & inv_{it} \leq dem_{it} \end{cases} \quad \forall i, t \tag{3.7}$$

For all i where $so_{it} = 1$, a MDP iteration is created within t. The i that experiences the stock-out in t and therefore is to be moved in t, is called $i_{\text{at hand}}$ during that iteration. In this iteration, $i_{\text{at hand}}$ can be moved to a different l. Thus, a period defined in the MDP of this DSLAP is called an iteration, and only the x_{ilt} of $i_{\text{at hand}}$ can be changed in that period.

Sections 3.4.1 up to 3.4.4 continue the mapping from the MILP to the MDP, using the defined sets and parameters.

3.4.1 State Space

The state space vector S at t consists of a demand vector $\overrightarrow{Demand}$ and a vector that represents the current SKU allocation of the warehouse, $\overrightarrow{SKU\ allocation}$. The values of state s change every iteration.

The demand vector consists of two attributes. First, for every SKU i, the future demand is determined:

$$\sum_{t=t}^{t+14} dem_{it} \forall i \tag{3.8}$$

Then, the demand of $i_{\text{at hand}}$ at t, is included as a second attribute in this vector:

$$dem_{i_{\text{at hand}}t} \tag{3.9}$$

Together, they make the demand vector as found in Equation 3.10.

$$\overrightarrow{Demand} = [\sum_{t=t}^{t+14} dem_{1t}, \sum_{t=t}^{t+14} dem_{2t}, ..., \sum_{t=t}^{t+14} dem_{I-1t}, \sum_{t=t}^{t+14} dem_{It}, dem_{i_{\text{at hand}}t}] \tag{3.10}$$

The SKU allocation vector, which is the current SKU allocation for all L of the warehouse, consists of all SKU numbers. All location-SKU combinations are included in here. Thus, the first value of the vector is the i assigned to $l = 1$, the second value is the i assigned to $l = 2$, until $l = L$ is reached.

To determine the i that is stored in l, Equation 3.11 is created.

$$\sum_{i=1}^{I} ix_{il} \forall l \tag{3.11}$$

Doing this for all l, the SKU allocation vector is created:

$$\overrightarrow{SKU\ allocation} = [\sum_{i=1}^{I} ix_{i1}, \sum_{i=1}^{I} ix_{i2}, ..., \sum_{i=1}^{I} ix_{iL-1}, \sum_{i=1}^{I} ix_{iL}] \tag{3.12}$$

Combining the demand vector and the SKU allocation vector, the state space is formulated as follows:

$$S = [\overrightarrow{Demand}, \overrightarrow{SKU\ allocation}] \tag{3.13}$$

The state space size is therefore $I + L + 1$. All values in the state space are standardised.

3.4.2 Action Space

During every iteration, for $i_{\text{at hand}}$, a value l is chosen, which will be the location of $i_{\text{at hand}}$ starting from the next iteration. This is then translated to the decision variables $x_{i_{\text{at hand}}lt} \forall L$. All locations L are included in the action space A, and thus the action space has size L. An action a_t can lead to one of two possible events: $i_{\text{at hand}}$ stays at the same location and with that all $x_{i_{\text{at hand}}lt}$ values stay the same. Or the location of $i_{\text{at hand}}$ changes , which means that one decision variable value of $i_{\text{at hand}}$ is transformed from 1 to 0 and another is transformed from 0 to 1. Therefore, the $x_{i_{\text{at hand}}lt} \forall L$ can be subject to change in every iteration.

If a change occurs in two of the decision variables of $i_{\text{at hand}}$, this can also have consequences for two decision variables of a different SKU. It could be that the chosen l is occupied, and items must be swapped. Therefore, changing decision variables can cause a change in four decision variables. This is to comply with the location and item constraints found in Equations 3.2 and 3.3 of the MILP.

Take the following example as illustration: there exists a warehouse with two SKUs and two storage locations. In time step t, SKU 1 is located at location 1, and SKU 2 is located at location 2. Thus, $x_{11t} = 1$, $x_{12t} = 0$, $x_{21t} = 0$ and $x_{22t} = 1$. If SKU 1 is moved to location 2, then SKU 2 must be moved to location 1 in order to comply with the location and item constraints found in equations 3.2 and 3.3 in the MILP. With this change, in total four decision variable values change ($x_{11t}, x_{12t}, x_{21t}, x_{22t}$). Therefore, the new decision variable values will be $x_{11t} = 0$, $x_{12t} = 1$, $x_{21t} = 1$ and $x_{22t} = 0$.

3.4.3 Reward Function

The objective in the MDP is to minimise the distance travelled over T. To translate the objective function in Equation 3.1 to a reward function that achieves the desired objective, an adapted version of the reward function $R(s_t, a_t)$ as presented in Troch et al. (2023) is applied.

In Troch et al. (2023), $R(s_t, a_t)$ is defined using the PFS. The original PFS is presented in Equation 3.14. For more details, see their paper.

$$\sum_{i=1}^{I} \left(\frac{occurences(p_i)}{\sum_{j=1}^{I} occurences(p_j)} * \frac{dist(p_i)}{maxDist} \right) \tag{3.14}$$

To translate the PFS to the problem presented in this paper, the following adaptations to the variables are made:

- $occurences(p_i) = \sum_{t=t}^{t+14} dem_{it}$.
- $dist(p_i) = d_l$.
- $maxDist = max(d_l)$.
- The function is multiplied with -1. With this adaptation, higher reward values represent better performance and lower reward values represent worse performance.

This turns the PFS formula into the adapted reward function, as found in Equation 3.15:

$$R(s_t, a_t) = -\sum_{i=1}^{I} \left(\frac{\sum_{t=t}^{t+14} dem_{it}}{\sum_{j=1}^{I} \sum_{t=t}^{t+14} dem_{jt}} * \frac{d_l}{max(d_l)} \right) \tag{3.15}$$

3.4.4 State Transition Matrix

An iteration always ends with a taken action a_t, after which two possible scenarios can occur:

In the first scenario, the next iteration in t commences, with a potentially changed $\overrightarrow{SKU\ allocation}$ because of a_t. The new $i_{\text{at hand}}$ and s_{t+1} are presented immediately. The new $i_{\text{at hand}}$ is determined by looping through all I in increasing order of SKU number, where the first i for which $so_{it} = 1$ becomes the $i_{\text{at hand}}$.

In the second scenario, the iteration was the final iteration of t, therefore the MDP continues to the next time step $t + 1$. Before arriving in the next state s_{t+1}, demand $dem_{it+1} \forall i$ occurs. This new demand changes the inventory $inv_{it+1} \forall i$ in the following way:

$$inv_{it+1} = inv_{it} - dem_{it+1} \forall i \tag{3.16}$$

The changes in inventory determine the values of $so_{it+1}\ \forall i$, and therefore also the value of $ITER_{t+1}$ and the next $i_{\text{at hand}}$. The state transition from s to s_{t+1} is executed in the same way in both occasions.

Table 3.1: Nomenclature table.

Symbol	Description
$i \in I$	Set of SKUs
$i_{\text{at hand}}$	SKU at hand in current iteration
$Length$	Length of the warehouse
$Width$	Width of the warehouse
$l \in L$	Set of storage locations
$t \in T$	Set of time steps or periods
$e \in E$	Set of training episodes
d_l	Distance from I/O point to l
dem_{it}	Demand of i in t
inv_{it}	Inventory of i in t
cap_i	Storage capacity of i
x_{ilt}	Decision variable, whether i is assigned to l or not
so_{it}	Stock-out of SKU i in time step t
$ITER_t$	Total number of iterations (stock-outs) in t
$\overrightarrow{Demand}$	Demand vector that consists of future demand for all SKUs and the current demand of $i_{\text{at hand}}$ at t
$\overrightarrow{SKU\ allocation}$	SKU allocation vector that consists of the SKU allocation at t
S	Set of states, where each state consists of the demand vector and the storage allocation vector
$s_t \in S$	State in time step t
A	Set of actions, where each action represents a location
$a_t \in A$	Action in time step t
$R(s_t, a_t)$	Reward obtained after taking action a_t in state s_t
γ	Discount factor ($0 \leq \gamma \leq 1$)

Table 3.2: Experimental settings.

Symbol	Value
I	150 SKUs
L	150 storage locations
T	365 days
E	100 episodes
$Length$	10 storage locations
$Width$	15 storage locations

3.5 Summary

In summary, in this chapter the DSLAP is formally presented. First, the most important assumptions, together with the problem objective and parameters are discussed. These are combined in the MILP, to form the basic mathematical model of the DSLAP. Then, the MILP is translated into a MDP, which makes it suitable to be solved with DQN. An overview of all parameters and variables with their definition is shown in Table 3.1. The accompanying values are found in Table 3.2. The information presented in this chapter forms the basis for the rest of this paper.

Chapter 4

Solution Approach

In this chapter, the methods used to solve the DSLAP are described in detail. An elaborate description is given about the DQN design and the hyperparameters used. Finally, the benchmarks used are discussed and exemplified.

4.1 Deep Q Learning

In this paper, the DQN approach applied is based upon Horgan et al. (2018). The pseudocode found in their paper is used and also presented in figure 4.1a and Figure 4.1b. More details are discussed in Section 4.2. The mean squared error is used as loss function, Adam is used as optimiser.

4.2 Deep Q Learning Design

The DQN algorithm works in the following way: the actor algorithm, as presented in Figure 4.1a, works within the warehouse environment, experiences different states and acts upon them, to consequently receive a reward. These complete experiences are then stored into the replay memory. The learner algorithm shown in Figure 4.1b then samples experiences from the replay memory, to re-experience them and to update the network parameters.

To reduce overestimation and bias, the DQN uses a double Q network with one learner (online) and one target (offline) network. The learner network is continuously being updated, while the target network is only updated every *target update* steps. *target update* is the hyperparameter that determines the update frequency. This is explained in more detail in Section 4.3. In the next paragraphs, further explanation is given on the key concepts of the DQN algorithm used: the learner network, the target network, the experience replay and the exploration-exploitation trade-off are discussed.

Learner network: This is the network that gets updated continuously. After every experience, it uses this experience to improve its ability to make the right decision in different situations. These experiences are used to update its parameters, and its parameters are used to update the parameters of the target network periodically.

```
1: procedure ACTOR(B, T)                          ▷ Run agent in environment instance, storing experiences.
2:     θ_0 ← LEARNER.PARAMETERS( )                ▷ Remote call to obtain latest network parameters.
3:     s_0 ← ENVIRONMENT.INITIALIZE( )            ▷ Get initial state from environment.
4:     for t = 1 to T do
5:         a_{t-1} ← π_{θ_{t-1}}(s_{t-1})         ▷ Select an action using the current policy.
6:         (r_t, γ_t, s_t) ← ENVIRONMENT.STEP(a_{t-1})   ▷ Apply the action in the environment.
7:         LOCALBUFFER.ADD((s_{t-1}, a_{t-1}, r_t, γ_t))   ▷ Add data to local buffer.
8:         if LOCALBUFFER.SIZE( ) ≥ B then        ▷ In a background thread, periodically send data to replay.
9:             τ ← LOCALBUFFER.GET(B)             ▷ Get buffered data (e.g. batch of multi-step transitions).
10:            p ← COMPUTEPRIORITIES(τ)           ▷ Calculate priorities for experience (e.g. absolute TD error).
11:            REPLAY.ADD(τ, p)                   ▷ Remote call to add experience to replay memory.
12:        end if
13:        PERIODICALLY(θ_t ← LEARNER.PARAMETERS())   ▷ Obtain latest network parameters.
14:    end for
15: end procedure
```

(a) Actor algorithm (Horgan et al., 2018)

```
1: procedure LEARNER(T)                           ▷ Update network using batches sampled from memory.
2:     θ_0 ← INITIALIZENETWORK( )
3:     for t = 1 to T do                          ▷ Update the parameters T times.
4:         id, τ ← REPLAY.SAMPLE( )               ▷ Sample a prioritized batch of transitions (in a background thread).
5:         l_t ← COMPUTELOSS(τ; θ_t)              ▷ Apply learning rule; e.g. double Q-learning or DDPG
6:         θ_{t+1} ← UPDATEPARAMETERS(l_t; θ_t)
7:         p ← COMPUTEPRIORITIES( )               ▷ Calculate priorities for experience, (e.g. absolute TD error).
8:         REPLAY.SETPRIORITY(id, p)              ▷ Remote call to update priorities.
9:         PERIODICALLY(REPLAY.REMOVETOFIT())     ▷ Remove old experience from replay memory.
10:    end for
11: end procedure
```

(b) Learner algorithm (Horgan et al., 2018)

Figure 4.1: Pseudocode of actor and learner algorithm.

Target network: A separate target network is used. It is a copy of the main neural network used by the learner and the actor. The target network is periodically updated with parameters of the learner network. Using a separate target network improves stability and convergence of the learning process.

Experience replay: Distributed prioritised experience replay, which is an extension of the classic experience replay, is used. This is defined as offline learning. The samples collected by the actor algorithm get assigned priorities based upon the importance of these samples. Samples with higher priorities have larger temporal difference errors. Within the learner algorithm, samples are drawn from the experience replay buffer based upon their priorities.

Exploration-exploitation trade-off: The epsilon greedy algorithm is used to determine the proportion of time exploration and exploitation are applied, using several hyperparameters. The algorithm slowly works from a more explorative way of working towards a more exploitative way of working. The epsilon threshold (*eps thresh*) value is determined, using the epsilon start (*eps start*), epsilon end (*eps end*), epsilon decay rate (*eps decay*) hyperparameters. The *steps* parameter is the number of iterations done in the current training session. The resulting epsilon threshold value is always a value between 0 and 1, and represents the proportion of time exploration is applied. The formula used to determine the epsilon threshold value is the epsilon threshold function in Equation 4.1, and computed in every step.

$$eps\ thresh = eps\ end + (eps\ start - eps\ end) * exp(-1 * steps/eps\ decay) \tag{4.1}$$

4.3 Hyperparameters

Several hyperparameters are used to configure the DQN. Hyperparameter tuning is done with different ranges of values for each hyperparameter. The hyperparameter tuning process is further explained in Section 5.2. The ultimately used values are presented in Table 4.2. In this section, all different parameters and their function are explained. A brief description of each hyperparameter is shown in Table 4.1.

Alpha α and beta β are used for sampling experiences from memory. α determines the weight priorities have when selecting experiences. β determines the diversity of samples taken from the memory, by giving experiences with a small priority also a probability of being selected.

The discount factor γ is used to determine the relative importance of future and current rewards. The learning rate *lr* represents the the step size used to update the parameters of the neural network during the training phase. It indirectly impacts the convergence rate and stability of the training. The *replay buffer size* is the size of the memory of the experience replay buffer. This determines how many experiences it can 'remember', and it impacts the training process stability and sample efficiency. The *batch size* equals how many experiences are drawn from the agent memory during the training process, which also impacts the stability of this process. The *target update* is the frequency at which the learner network and the target network are synchronised, by updating the target network.

Table 4.1: Hyperparameter definition table.

Hyperparameter	Description
α	weight of priorities
β	diversity of sample
γ	discount factor
lr	learning rate
$replay\ buffer\ size$	memory size
$batch\ size$	sample size
$target\ update$	frequency of synchronisation
$eps\ start$	maximum value $eps\ thresh$
$eps\ end$	minimum value $eps\ thresh$
$eps\ decay$	decrease rate $eps\ thresh$

Table 4.2: Hyperparameter values table.

Hyperparameter	Value
α	0.8
β	0.4
γ	0.4
lr	0.001
$replay\ buffer\ size$	10,000
$batch\ size$	32
$target\ update$	100
$eps\ start$	1
$eps\ end$	0.1
$eps\ decay$	600,000

The hyperparameters that are linked to the epsilon threshold value *eps thresh*, which is used for the epsilon greedy algorithm, are the *eps start* and *eps end* values together with *eps decay*. The first two parameters determine the maximum and minimum values *eps thresh* can take on. *eps decay* is then used to determine the rate at which *eps thresh* decreases from *eps start* to *eps end*.

4.4 Benchmarks

As benchmark tools, five different storage location assignment policies are used. First, the simple random assignment is used to provide a solution with a relatively low reward value. The other four benchmark tools are all periodic ABC classification, but with different periods. The ABC classes are decided based upon demand volumes. Items are ranked using demand volumes: the top 20% is assigned to class A, the next 30% is assigned to class B, and the final 50% is assigned to class C.

Periodic ABC classification implies that the ABC classification is revised every period. Here, periods can vary. For this paper, ABC classification of 1, 2, 3, and 4 periods per year are used. When ABC classification is done only once, it is based upon the

demand of the entire year. With two periods, it is executed once in the beginning of the year and based upon the demand of the first half of the year. When the first half of the year is over, ABC classification is executed again, but now based upon the demand of the second half of the year. When using three or four periods, the way of working is the same, but with shorter periods.

In this section, two example applications of the periodic ABC classification code are shown. Here, four SKUs with periodic demand must be assigned to locations in a 2x2 warehouse. The demand values are found in Table 4.3 and the I/O distance values for all locations are found in Table 4.4.

Table 4.3: Dummy demand table.

i	Demand period 1	Demand period 2	Total demand
SKU 1	5	1	6
SKU 2	4	4	8
SKU 3	2	3	5
SKU 4	7	2	9

Table 4.4: Dummy distance table.

l	d_l
Location 1	1
Location 2	2
Location 3	3
Location 4	4

ABC classification: 1 period

When applying the ABC classification with 1 period, the classification algorithm uses the cumulative demand of all periods to determine the SKU location assignment. With the highest demand, SKU 4 is assigned to the location with the lowest distance, location 1. SKU 3 has the lowest demand and is therefore assigned to the location with the highest distance, location 4. See Table 4.5 for the complete solution. When applying ABC classification while considering all demand as one period, this is the optimal storage location assignment.

The $R(s_t, a_t)$ value cannot be computed, because not enough demand information is known. However, the travel distance can be computed to compare performance. Here, it is assumed that the order picker can only carry one item at a time. The total distance travelled is computed using Equation 3.1. This results in a total distance travelled of 63.

ABC classification: 2 periods

Here, the ABC classification is conducted using two periods. That means that every period defined in Table 4.3 is considered individually. Tables 4.6 and 4.7 respectively show the complete ABC classification solution of period 1 and period 2 respectively. SKU 4 has the highest demand in period 1, and is therefore assigned to location 1 in

Table 4.5: Dummy solution 1 period.

l	i
Location 1	SKU 4
Location 2	SKU 2
Location 3	SKU 1
Location 4	SKU 3

Table 4.6. In period 2, SKU 2 has the highest demand and is assigned to location 1. This is shown in Table 4.7. Locations 2 and 3 also have different SKUs assigned to them in both solutions shown in Tables 4.6 and 4.7.

Table 4.6: Dummy solution 2 periods (period 1).

l	i
Location 1	SKU 4
Location 2	SKU 1
Location 3	SKU 2
Location 4	SKU 3

Table 4.7: Dummy solution 2 periods (period 2).

l	i
Location 1	SKU 2
Location 2	SKU 3
Location 3	SKU 4
Location 4	SKU 1

Again using Equation 3.1, the total distance travelled can be determined. The total distance travelled in period 0 is 37 and the total distance travelled in period 1 is 20. Thus, the total distance travelled over both periods is 57 when applying an ABC classification with 2 periods. Because the ABC classification with 1 period resulted in a total distance travelled of 63, it is shown that using 2 periods for the ABC classification can outperform a one-period ABC classification with fluctuating demand. See Table 4.8 for a brief overview of the results.

Table 4.8: Results ABC classification comparison.

ABC classification	Total distance travelled
1 period	63
2 periods	57

4.5 Summary

In short, this chapter has provided a thorough elaboration on the approaches used to solve the DSLAP. The features used for the DQN are discussed, together with the supporting hyperparameters and their values. On top of that, the benchmarks for performance comparison are discussed in more detail and exemplified. The next chapter provides the final information on the design of the experiments.

Chapter 5

Experimental Setup

This chapter covers the experimental setup for tackling the DSLAP. First, the data used is discussed in Section 5.1. In Section 5.2, the tuning of the DQN is elaborated upon. Section 5.3 covers alle experiments run for this research. Finally, the subject of performance metrics is discussed in Section 5.4, after which a summary is given of this chapter.

5.1 Data Collection and Preprocessing

Data is essential for the experiments, and it is therefore discussed here. Because a mock-up warehouse is used, artificial data is generated. The datasets are complete and generated in a way that fits the MDP and DQN algorithm, which means that preparing and cleaning the data is not necessary. The data consist of demand values for every i in I and every t in T. Additionally, the inv_{i0} and cap_i are generated for every i in I. $inv_{i0} = 60$ and $cap_i = 120$ for every i in I. The experiments are conducted with both seasonalised and non-seasonalised datasets. Besides a dataset with $I = 150$, a smaller dataset with $I = 30$ is also generated. Both datasets use the same algorithm to generate the demand.

To create the demand, first a base demand value is created for every i, $base\ demand_i$. The value of $base\ demand_i$ ranges from 10 up to 100, and is created using the base demand function in Equation 5.1. In the function, x represents a random real number between 0 and 1.

$$base\ demand_i = int(x * 90) + 10 \tag{5.1}$$

Secondly, the $seasonality\ factor_{it}$ is generated for every i in I and every t in T. This is solely used for the seasonalised demand. First, every item is assigned a $category\ factor_i$, which is a random value between 0.04 and 1.5. This range contains values relatively close to 1, to prevent large fluctuations in demand. Other than that, the range boundaries are chosen arbitrarily. Then, in combination with day_t, which is a parameter representing the current day of the year that is linked to t, the $seasonality\ factor_{it}$ is computed. The seasonality factor function in Equation 5.2 is used for this.

$$seasonality\ factor_{it} = category\ factor_i * (0.5 + 0.5 * \sin((2 * 3.14159 * day_t * category\ factor_i)/365)) \tag{5.2}$$

To compute dem_{it}, two different function are used for seasonalised and non-seasonalised demand. Combining $base\ demand_i$ and $seasonality\ factor_{it}$, seasonalised dem_{it} is computed using the seasonalised demand function in Equation 5.3. The non-seasonalised demand function in Equation 5.4 computes the non-seasonalised dem_{it} for all i in I and all t in T. In both function, x and y are random real numbers between 0 and 1.

$$dem_{it} = int(base\ demand_i * (1 + x - y) * seasonality\ factor_{it}) \tag{5.3}$$

$$dem_{it} = int(base\ demand_i * (1 + x - y)) \tag{5.4}$$

5.2 Hyperparameter Tuning

In this section, the process of tuning the DQN algorithm is discussed. A part of the adaptations presented in this section are included in the experiments section, Section 5.3. The other part of the adaptations are not included in the experiment section, to keep that section comprehensible. The experiments that are not included in Section 5.3, did not yield different results than the results discussed in Chapter 6.

The hyperparameters presented in Section 4.3 are tuned using a grid search strategy. This means that an initial hyperparameter values set is selected, from which all possible combination are used. This predefined set contains a large range of values, as shown in Table 5.1. The specific values chosen for the grid search are presented in Appendix C.

Table 5.1: Hyperparameter tuning table.

Hyperparameter	Range
γ	0.4-0.99
lr	0.0005-0.1
$replay\ buffer\ size$	100-10,000
$batch\ size$	16-512
$target\ update$	10-200
$eps\ start$	0.9-1
$eps\ end$	0-0.1
$eps\ decay$	20,000-1,200,000

For further experimenting, the reward function is adapted from Equation 3.15 to Equation 5.5. This reward function computes a reward focused solely on the l of $i_{\text{at hand}}$. With that, the impact of a_t on i is isolated and sent to the DQN algorithm. This removes the impact of the other SKU, that is moved because of moving $i_{\text{at hand}}$, on the reward value.

$$R(s_t, a_t) = -\left(\frac{\sum_{t=t}^{t+14} dem_{i_{\text{at hand}}t}}{\sum_{j=1}^{I}\sum_{t=t}^{t+14} dem_{jt}} * \frac{d_l}{max(d_l)}\right) \tag{5.5}$$

Additionally, the reward function was adapted to show the improvement in reward value, comparing the reward value in $t-1$ and the reward value in t after moving the SKU at hand. This adaptation is shown in Equation 5.6.

$$R(s_t, a_t) = -\sum_{i=1}^{I}\left(\frac{\sum_{t=t}^{t+14} dem_{it}}{\sum_{j=1}^{I}\sum_{t=t}^{t+14} dem_{jt}} * \frac{d_l}{max(d_l)}\right) + \sum_{i=1}^{I}\left(\frac{\sum_{t=t-1}^{t+14} dem_{it-1}}{\sum_{j=1}^{I}\sum_{t=t-1}^{t+14} dem_{jt-1}} * \frac{d_l}{max(d_l)}\right) \tag{5.6}$$

A second point of adaptation is S. Tests were run with S not being standardised. Additionally, S was also changed to e.g. only containing $dem_{i_{\text{at hand}}t}$, and by including the I/O distance vector $\overrightarrow{I/O\ distance}$ and t in S. The definition of $\overrightarrow{I/O\ distance}$ is presented in Equation 5.7.

$$\overrightarrow{I/O\ distance} = [d_1, d_2, ..., d_{L-1}, d_L] \tag{5.7}$$

All adaptations of S that have been used for experiments are presented in Equations 5.8 to 5.12.

$$S = [dem_{i_{\text{at hand}}t}] \tag{5.8}$$

$$S = [dem_{i_{\text{at hand}}t},\ t] \tag{5.9}$$

$$S = [\overrightarrow{Demand}] \tag{5.10}$$

$$S = [\overrightarrow{Demand},\ \overrightarrow{SKU\ allocation},\ \overrightarrow{I/O\ distance}] \tag{5.11}$$

$$S = [\overrightarrow{Demand},\ \overrightarrow{SKU\ allocation},\ t] \tag{5.12}$$

Thirdly, experiments were run with different neural network sizes. The network depth and layer width have been adapted. The exact values used for the network depth and layer width in the experimentation phase are presented in Table 5.2. All layers within the neural network could have a different width.

Fourthly, as mentioned before, other experimenting included working with a smaller dataset with $I = L = 30$. This immediately decreases the action space as well. Finally, the activation functions were changed to include different functions. The activation functions examined are: Sigmoid, Tanh, ArcTan ReLU, Leaky ReLU, and SoftPlus. All parts of the neural network were using only one and the same activation function in a run, no mixing is done.

Table 5.2: Neural network tuning table.

Neural network	Range
Depth	2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12
Width	16, 32, 64, 128, 256, 512, 1024, 2048

5.3 Experiments

To assess the performance of the DQN in an elaborate way, several experimental experiments are created. Working with different experiments provides more insights, which aids in drawing complete conclusions. Besides, it is also insightful for other researchers.

The experiments presented in this section have a couple of aspects in common. All of them apply DQN to solve the DSLAP. The initial storage location assignment is created randomly.

The experiments consist of different demand behaviour and different DQN configurations. Additionally, a couple of experiments use the individual reward function, as found in Equation 5.5, as $R(s_t, a_t)$.

The experiments are discussed in this section, highlighting their commonalities and differences. Experiments 1, 4, 7 and 10 are considered the main experiments and therefore will be discussed in more detail in Chapter 6. All experiments are summarised in Table 5.3.

5.3.1 Experiment 1

The first experiment is main focus of this paper. In this experiment, everything is as discussed in the previous sections. A dataset with $I = 150$ and seasonalised demand is used to test the performance of the DQN. The DQN is fully implemented, including the experience replay. The reward function $R(s_t, a_t)$ is computed using the adapted reward function in Equation 3.15, where all I are included.

5.3.2 Experiment 2

In this experiment, the individual reward function in Equation 5.5 is used. Besides that, this experiment is the same as experiment 1 in Section 5.3.1. A dataset with $I = 150$ and seasonalised demand is used. Here, the DQN is also implemented with the experience replay included.

5.3.3 Experiment 3

Experiment 3 is also an adaptation of experiment 1 presented in Section 5.3.1. Experiment 3 implements DQN without experience replay. Besides that, this experiment works with a large dataset with seasonalised demand. Equation 3.15 is used.

5.3.4 Experiment 4

This experiment is also considered a main experiment, because it uses the same implementation as experiment 1 discussed in Section 5.3.1. This experiment is created to compare the performances of the DQN on seasonalised demand, as presented in experiment 1 in Section 5.3.1, and on non-seasonalised demand, as presented in this experiment. In this experiment, the large dataset with non-seasonalised demand is used. DQN is fully implemented, including the experience replay. Equation 3.15 is used in experiment 4.

5.3.5 Experiment 5

Experiment 5 works with a large, non-seasonalised demand, dataset. DQN is implemented with experience replay and the individual reward function, as found in Equation 5.5, is used.

5.3.6 Experiment 6

This experiment also works with a large, non-seasonalised, dataset. DQN is implemented without experience replay, and as a reward function Equation 3.15 is used.

5.3.7 Experiment 7

Experiment 7 is also considered one of the main experiments, because it uses the exact same implementation as experiment 1. However, in this experiment a small dataset with $I = 30$ is used. The dataset contains seasonalised demand. The DQN is fully implemented, including the experience replay. Equation 3.15 is used as the reward function.

5.3.8 Experiment 8

In this experiment, Equation 5.5 is used as the reward function. Besides that, a small dataset with seasonalised demand is used. Here, the DQN is also implemented with the experience replay included.

5.3.9 Experiment 9

Experiment 9 implements DQN without experience replay. Besides that, this experiment works with a small dataset with seasonalised demand. Equation 3.15 is used for the reward function.

5.3.10 Experiment 10

This experiment uses the same implementation as experiment 4 discussed in Section 5.3.4, but applied on a smaller dataset. Therefore, experiment 10 is also a main experiment. In this experiment, the dataset used is small with non-seasonalised demand. DQN is fully implemented, including the experience replay. Equation 3.15 is used as reward function in experiment 10.

5.3.11 Experiment 11

Experiment 11 works with a small, non-seasonalised demand, dataset. DQN is implemented with experience replay and Equation 5.5 is used as the reward function.

5.3.12 Experiment 12

This experiment also works with a small, non-seasonalised, dataset. DQN is implemented without experience replay, and as a reward function Equation 3.15 is used.

Table 5.3: Experiments.

Experiment	Dataset	Demand type	Experience replay	$R(s_t, a_t)$
1	Large	Seasonalised	✓	I
2	Large	Seasonalised	✓	$i_{\text{at hand}}$
3	Large	Seasonalised		I
4	Large	Non-seasonalised	✓	I
5	Large	Non-seasonalised	✓	$i_{\text{at hand}}$
6	Large	Non-seasonalised		I
7	Small	Seasonalised	✓	I
8	Small	Seasonalised	✓	$i_{\text{at hand}}$
9	Small	Seasonalised		I
10	Small	Non-seasonalised	✓	I
11	Small	Non-seasonalised	✓	$i_{\text{at hand}}$
12	Small	Non-seasonalised		I

5.4 Performance Metrics

The performance metrics used to determine the quality of the solutions are discussed in this section. As mentioned in Sections 3.4.3 and 5.2, two adaptations of the PFS are used as the reward function to measure the performance. To gain insights in the performance of different experiments, the maximum and minimum reward values achieved per episode are shown in plots. Additionally, the development of the reward function values throughout all iterations are shown as well. These insights provide useful information to compare the performance among different experiments.

Another insightful metric is the loss value. The development over time of the loss values are presented in graphs for every experiment. This provides insights into both the predicting performance of the algorithm, as well as into the convergence of the DQN algorithm.

5.5 Summary

This chapter discussed the data used, ranging from how the data is obtained to how seasonality is generated. Next, the chapter touched upon the subject of hyperparameter tuning. Here, the tuning of the DQN algorithm is discussed in more detail. The

hyperparameters are discussed, as well as the reward function, the state space and the neural network configurations.

Thirdly, all experiments conducted for this research are introduced in Section 5.3. Their characteristics, commonalities and differences are discussed. Finally, the performance metrics used to determine the performance are briefly elaborated upon.

Using the experimental information presented in this chapter, the next chapter discusses the results and analyses obtained from these experiments.

Chapter 6

Results and Discussions

In this chapter, the results of the conducted experiments are presented and discussed. The chapter starts with presenting the performance of the benchmarks. Then, the results of the experiments are presented. Thirdly, the results are analysed by considering its strengths and limitations. Finally, the practical implications of the results are touched upon.

6.1 Benchmark Performance

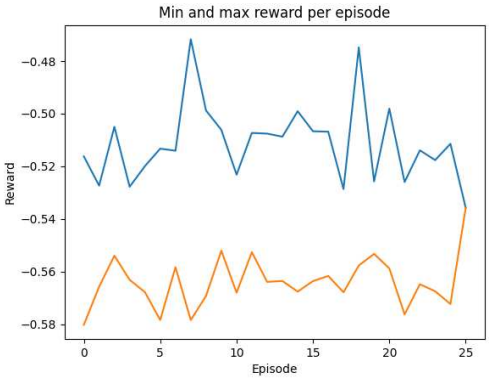
In this section, the minimum and maximum reward values of the benchmarks are presented. First, the benchmark performance on the large dataset is discussed. Secondly, the benchmark performance on the small dataset is discussed. Every section consists of the random solution performance and the performance of the periodic ABC classification benchmark.

6.1.1 Benchmark Performance on Large Dataset

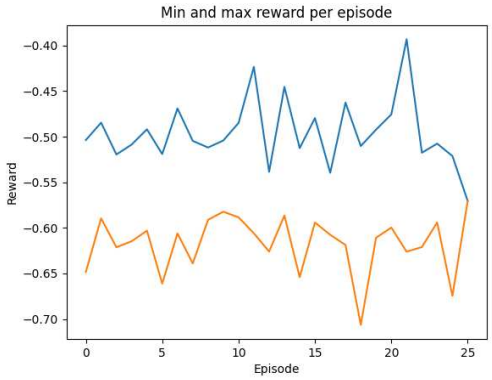
In Figures 6.1a and 6.2, the minimum and maximum reward values per episode are shown for the large dataset. Figure 6.1a represents the minimum and maximum reward of the randomly created storage allocation. Figures 6.2a, 6.2b, 6.2c, and 6.2d represent the minimum and maximum reward of the storage allocation created by applying periodic ABC classification.

The minimum rewards resulting from the random run, found in Figure 6.1a, vary from -0.58 up to -0.55. The minimum rewards from the ABC classification with one period range from -0.56 to -0.54. The two-period ABC classification shows values around -0.54. Three-period ABC classification contains minimum values from -0.54 up to -0.50. Finally, the ABC classification with four periods resulted in minimum values in the range of -0.50 and -0.47.

The maximum rewards shown in Figure 6.1a, range from -0.53 to -0.47. The maximum rewards resulting from the one-period ABC classification range from approximately -0.44 to -0.42. For the two-period ABC classification this ranges from -0.31 to -0.29, for the three-period ABC classification from -0.31 to -0.29, and for the four-period ABC classification from -0.31 to -0.29.



(a) Random solution large dataset



(b) Random solution small dataset

Figure 6.1: Random solution min max reward plot.

6.1.2 Benchmark Performance on Small Dataset

For the small dataset, Figures 6.1b and 6.3 show the minimum and maximum rewards of the benchmarks. Figure 6.1b shows the minimum and maximum rewards achieved using the random solution. Figures 6.3a, 6.3b, 6.3c, and 6.3d show the minimum and maximum values achieved using the one-period, two-period, three-period and four-period ABC classification respectively.

Looking at Figure 6.1b, it is shown that the minimum reward value achieved ranges from -0.70 up to -0.58. The one-period ABC classification, the minimum rewards achieved are within the range of -0.60 and -0.57. Figure 6.3b shows the two-period ABC classification, where the lowest minimum value achieved is -0.58, and the highest minimum value achieved is -0.55. The three-period ABC classification shows minimum values in a range from -0.55 up to -0.52. One outlier is noticed here, which is not included in the results. The four-period also shows one outlier, which is excluded from the results. The range of minimum values is between -0.52 and -0.50.

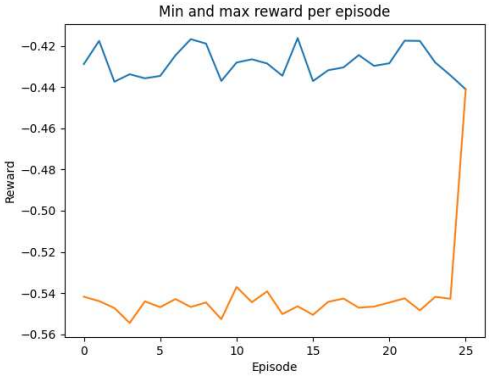
For the maximum reward values, the random solution shows a range between -0.54 and -0.40. The one-period ABC classification shows a range of maximum reward values from -0.44 up to -0.42. The two-period, three-period and four-period ABC classification all show maximum reward values ranging from -0.31 to -0.29.

6.2 Experiments Performance

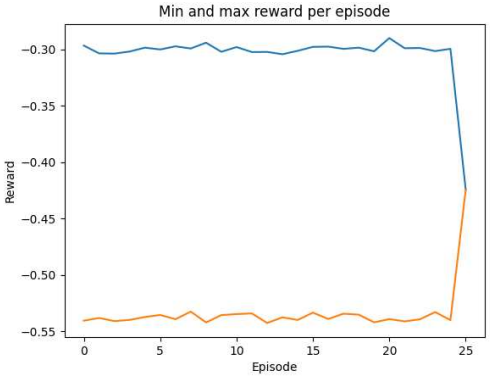
This section first discusses the main experiments, which are experiment 1, 4, 7 and 10. Then, interesting results from the other experiments are discussed. The Figures resulting from these other experiments can be found in appendix D. The experimental configurations are found in Table 5.3.

6.2.1 Experiment 1

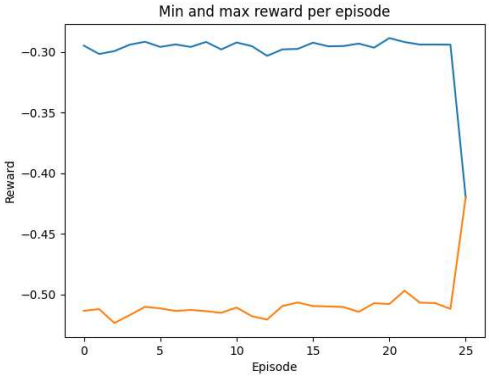
As Figure 6.4b shows, the loss value reaches a peak shortly after starting the training phase. Afterwards, it slowly declines until it stabilises after about 2 million iterations. The loss is approximately between 0.4 and 0.7.



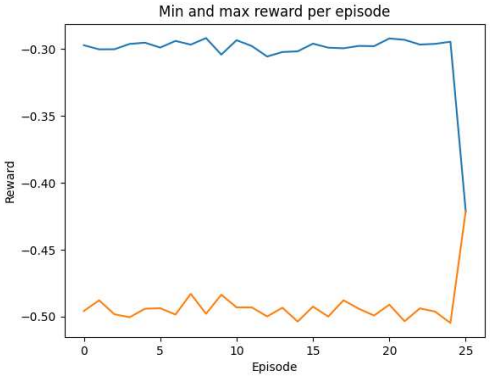
(a) ABC classification with 1 period



(b) ABC classification with 2 periods

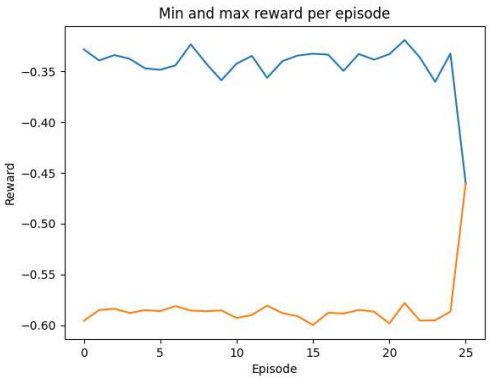


(c) ABC classification with 3 periods

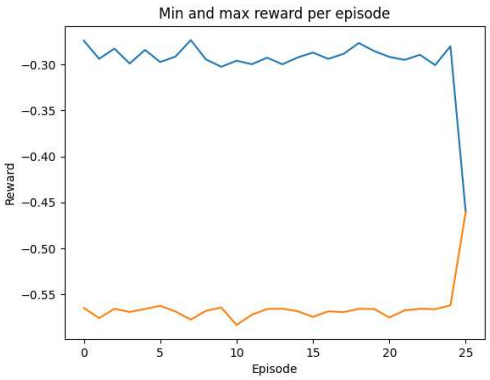


(d) ABC classification with 4 periods

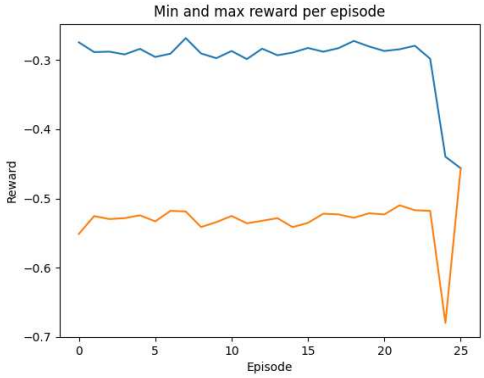
Figure 6.2: ABC classification min max reward plot large dataset.



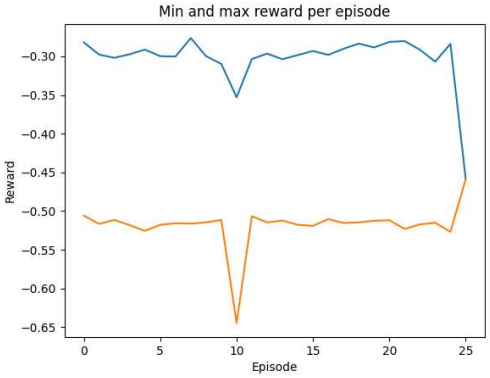
(a) ABC classification with 1 period



(b) ABC classification with 2 periods



(c) ABC classification with 3 periods

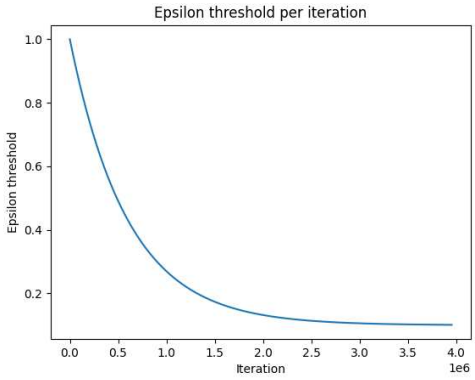


(d) ABC classification with 4 periods

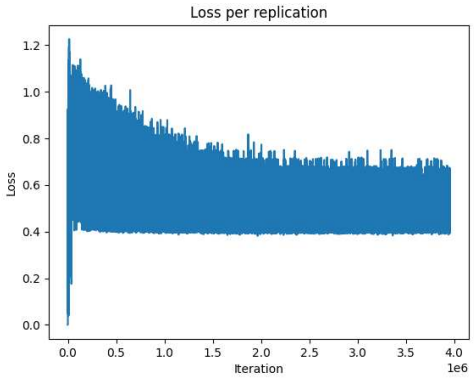
Figure 6.3: ABC classification min max reward plot small dataset.

Figure 6.4c shows behaviour that does not change throughout the training phase. The values consistently stay between approximately -0.58 and -0.50. These values are computed using Equation 3.15.

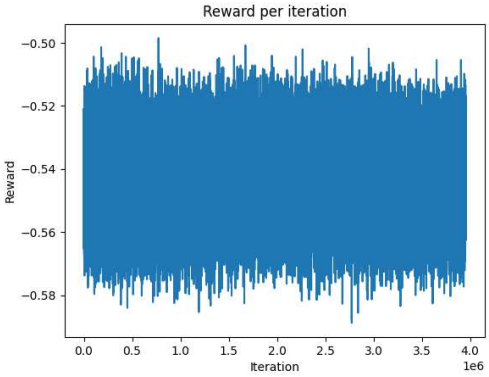
The minimum and maximum achieved reward values per episode are presented in Figure 6.4d. Like the values in Figure 6.4c, the minimum and maximum reward values also consistently show the same behaviour, and do not improve or deteriorate over time. In every episode, the lowest reward value lies around -0.58, while the highest reward value lies around -0.51.



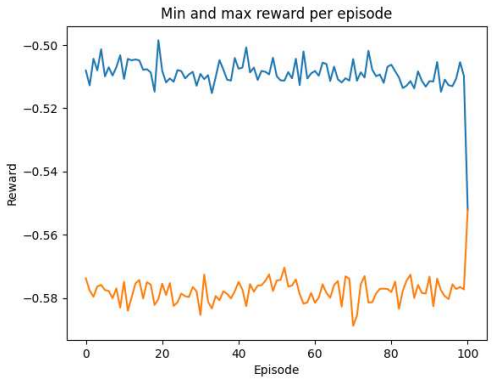
(a) Epsilon threshold per iteration



(b) Loss per iteration



(c) Reward per iteration



(d) Maximum and minimum reward per episode

Figure 6.4: Experiment 1 results.

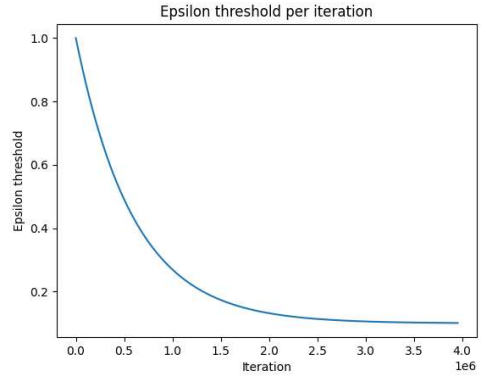
6.2.2 Experiment 4

The dataset that contains demand without seasonality, shows the same behaviour as the dataset discussed in 6.2.1. In Figure 6.5b it is shown that the loss value first reaches a peak, after which it slowly declines until it stabilises after about 2 million iterations. The loss is approximately between 0.4 and 0.8.

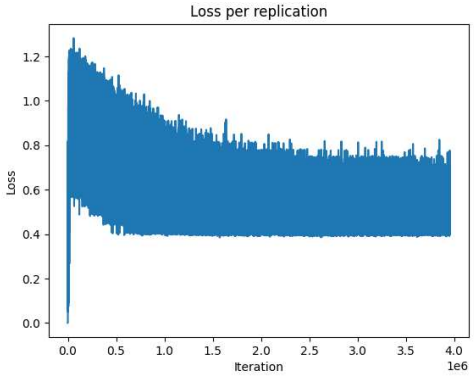
Figure 6.5c also shows behaviour that does not change throughout the training phase. The values consistently stay between approximately -0.58 and -0.50.

The minimum and maximum achieved reward values per episode are presented in Figure 6.5d. Like the values in Figures 6.4c, 6.4d and 6.5c, the minimum and maximum reward values also consistently show the same behaviour, and do not improve over

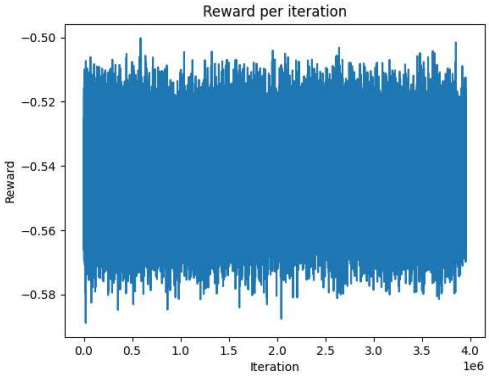
time. In every episode, the lowest reward value lies around -0.58, while the highest reward value lies around -0.51.



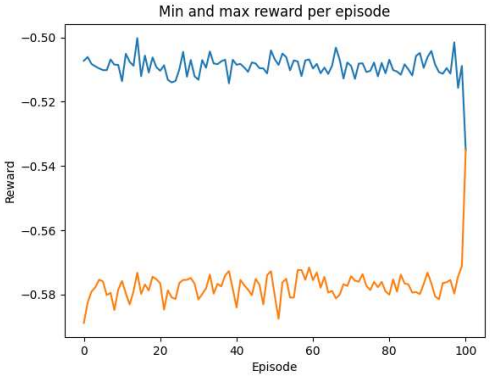
(a) Epsilon threshold per iteration



(b) Loss per iteration



(c) Reward per iteration



(d) Maximum and minimum reward per episode

Figure 6.5: Experiment 4 results.

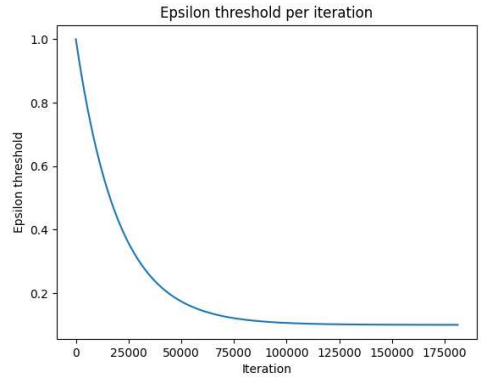
6.2.3 Experiment 7

As discussed in Section 5.3.7, this experiment is the same as experiment 1, but works with a smaller dataset.

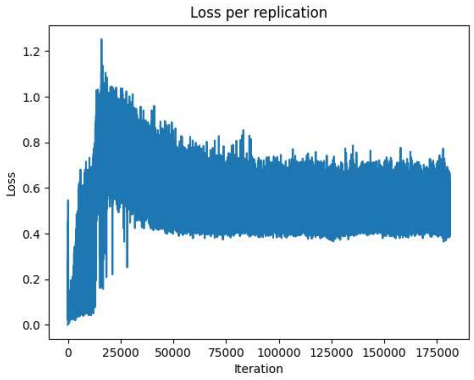
The loss value first reaches a peak, after which is declines. After approximately 75,000 iterations, the loss values stabilise between 0.4 and 0.8. This is shown in Figure 6.6b.

Figure 6.6c shows the reward values, which remain consistent from the start of the training phase. The values are between -0.7 and -0.4, where most values lie within the range of approximately -0.6 and -0.5.

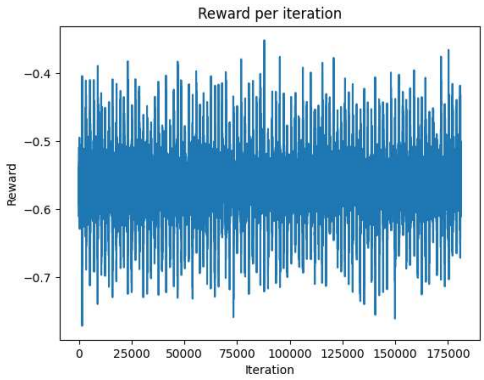
The Figure that shows the minimum and maximum rewards attained in each episode show the ranges of the reward values more clearly. Figure 6.6d shows that the maximum reward achieved in every episode is between -0.45 and -0.4, while the minimum reward experienced in every episode lies between -0.8 and -0.65.



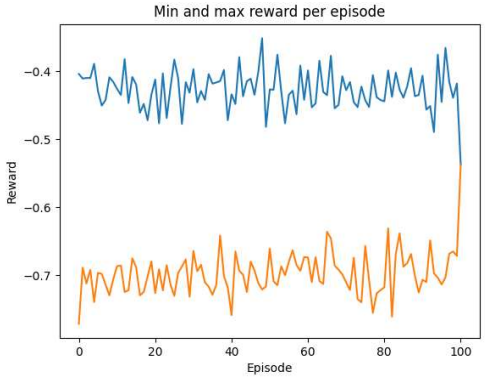
(a) Epsilon threshold per iteration



(b) Loss per iteration



(c) Reward per iteration



(d) Maximum and minimum reward per episode

Figure 6.6: Experiment 7 results.

6.2.4 Experiment 10

Experiment 10 is an adaptation of experiment 4, working with a smaller dataset as is discussed in Section 5.3.10.

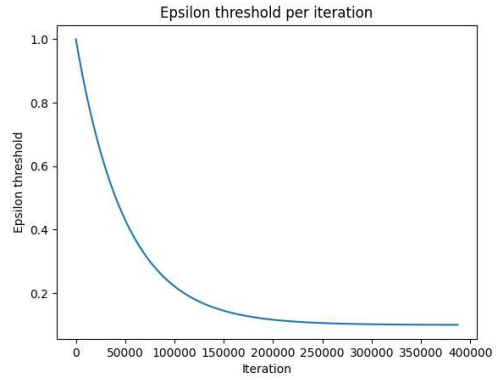
The loss values shown in Figure 6.7b show a high peak in the beginning, after which they drop and slightly decrease between iterations 50,000 and 150,000. After this slight decrease, the loss values remain steady with values ranging from 0.5 up to 0.7.

In Figure 6.7c, the reward values in all iterations are plotted. The reward values remain stabilised from the beginning, and thus show no change in behaviour throughout the training phase. The values are between -0.625 and -0.5.

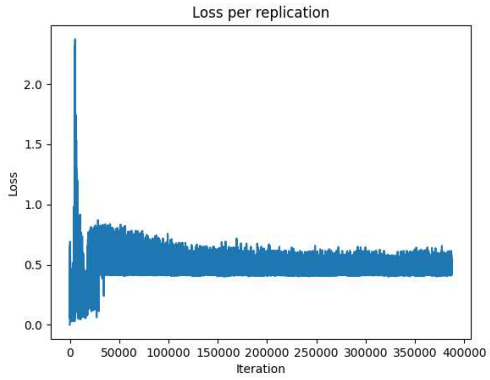
Figure 6.7d shows the minimum and maximum reward obtained in every episode. The maximum rewards achieved in every episode lie approximately in the range of -0.52 up to -0.475. The minimum rewards range from -0.64 up to -0.61.

6.2.5 Other Experiments

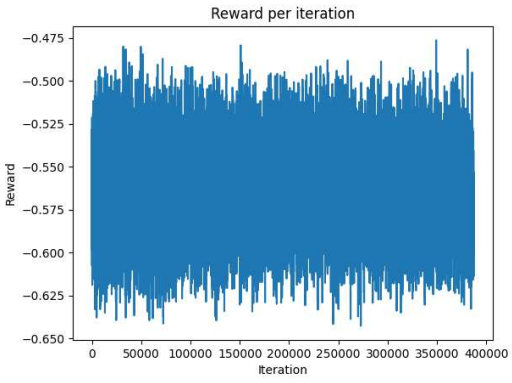
In appendix D, the results of all other experiments are shown. In general, most results show similar behaviour. Most loss plots do not show any improvement or deterioration. This also holds true for the total reward and minimum and maximum reward plots. These show stable behaviour, where values remain within the same range



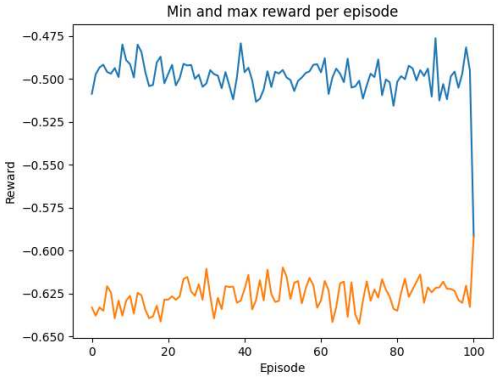
(a) Epsilon threshold per iteration



(b) Loss per iteration



(c) Reward per iteration



(d) Maximum and minimum reward per episode

Figure 6.7: Experiment 10 results.

throughout all episodes.

However, a few loss plots show different behaviour. Figure D.1a shows the loss values of experiment two. The loss plot shows a slow but steady decrease in values. The loss plot shown in Figure D.3a also shows behaviour that is different. In this Figure, the loss drops after approximately 1 million iterations, after which it further decreases. In Figures D.5a, the loss values of experiment 8 are presented. This Figure shows that the loss values slightly decrease in the beginning of the experiment, after which it shows stable behaviour for the remaining experiment. Finally, Figure D.7a also shows different behaviour than the rest of the loss plots. The loss values start with a large rise, after which it decreases and the maximum loss values remain the same throughout all episodes. However, the minimum loss values decrease, which increases the range within which the loss values are situated.

6.3 Discussion of Findings

This section analyses the results that are obtained from the experiments. The performance of the solution presented in the experiments are compared to the benchmarks, and conclusions are drawn from this analysis.

When analysing the results of the experiments that work with a large dataset in

Section 6.2, it becomes clear that the minimum and maximum rewards achieved in all experiments are lower than all periodic ABC classification benchmarks. The minimum and maximum rewards achieved by the experiments are comparable to the rewards of the randomly initialised storage allocation.

The results of the experiments that work with a small dataset show somewhat different results. Experiment 7 shows maximum reward values that are consistently slightly better than the reward values achieved by the random solution. The range of the minimum reward values in experiment 7 is larger than the range of the minimum reward values in the random solution. These observations do not necessarily imply that experiment 7 is learning a suiting policy, because other points of improvements are lacking.

Focusing on experiment 10, the minimum and maximum reward values of experiment 10 are comparable to the performance of the random solution. When comparing the loss values of experiment 10 to the loss values of experiments 1, 4 and 7, experiment 10 seems to have difficulty with improving the loss values. This could imply that there is a difference in learning capabilities of the DQN algorithm between experiment 10 and experiments 1, 4 and 7.

To provide a few general remarks: when looking at the gap between the achieved minimum and maximum reward values of individual episodes, the Figures show a relatively large difference between the two. This can be caused by inconsistent performance, possibly due to the randomness that is still in there, because the DQN algorithm was not able to learn a good policy.

Other evidence that shows that the DQN is unable to learn a good policy, is because the rewards shown in e.g. Figure 6.4c are constantly oscillating, without showing any improvements. However, the loss values in Figures 6.4b, 6.5b and 6.6b do show that the DQN is improving in its prediction abilities of the expected current and future reward values. This is shown because the loss values decrease over time and stabilise. However, the DQN algorithm is unable to translate this knowledge into actions that improve the rewards achieved.

Finally, the performance of the DQN on the seasonalised and non-seasonalised datasets is also similar. The DQN is still unable to show performance improvements when working with demand without seasonality. This could partially be due to the fact that the presented DQN, including e.g. its observation space, is created for working with seasonalised demand. However, it is still another result that shows the DQN seems unable to learn under the current circumstances.

An interesting side note, that is conform the expectations, is that the ABC classification performance improves when increasing the number of periods. At least, when increasing from 1 period to 4 periods, more periods lead to better performance. Increasing from 1 period to 2 periods, both the minimum and maximum rewards achieved improve. Increasing from 2 to 3, the maximum rewards achieved remain the same, but the minimum rewards achieved improve. Increasing from 3 to 4, the maximum rewards again stay the same, but the minimum rewards improve. Thus, in these last steps, the best maximum reward stays the same, but the range of the reward values achieved decreases.

6.3.1 Strengths of the Presented Results

This section discusses the strengthw of the presented results. It sums up why these results are relevant in theory and in practice.

Strengths:

- It is the first study to tackle unstable demand for this type of DSLAP with DQN.
- Provides insights of multiple experiments, with different instance sizes, different types of demand, different DQN configurations and different reward functions.
- Makes this type of DSLAP more easily accessible for researchers because a MDP replaces a detailed simulation.
- Connects theory with practice by applying a different type of DSLAP in a dynamic warehouse environment, without a detailed simulation on the warehouse processes.

6.3.2 Limitations and Areas for Improvement

To explore the reasons of why the DQN did not perform as desired, multiple subjects are discussed. Several potential causes are identified, and for every point the used methods are discussed first, after which possible improvements are proposed. Multiple aspects have already been explored, as discussed in Section 5.2.

Neural network architecture

The neural network is an essential part of DQN and can be modified in an infinitely number of ways. Where currently the FFNN is used, choosing a different architecture could lead to huge changes in performance. When working with a FFNN, configuration of the neural network is one of the main determinants of the model performance, and changes in this configuration could potentially improve the model performance drastically.

Neural network size

Starting with the neural network size, experimenting with both the neural network depth and width could lead to changes in the learning behaviour of the model. A large neural network is of importance because it allows for the comprehension of complex decision environments. The network will be able to learn patterns that smaller networks may not be able to learn. A wide network is required when there are complicated relationships within the decision environment. Making the neural network too deep and too wide can, however, lead to overfitting, making it less applicable to new datasets.

With a neural network depth of 9 layers and a size that varies across these layers, being either 512 or 1024, it is possible that the neural network is unable to translate the input variables into the required action. Even though multiple experiments were run with different neural network sizes, many possible configurations have not been explored due to the infinitely large configuration space. Because of this, the neural

network size is a potential bottleneck in the DQN performance. It is recommended to optimise this by running more experiments and by using heuristics to determine the optimal neural network size.

Activation function

The choice of activation functions also plays a role in the model performance. Like network depth and width, choosing the right activation functions allows the network to model complex environments. Additionally, it can enable stability when training and testing the network. Because of these reasons, the activation function has a great impact on the performance of the neural network and should be carefully chosen.

For this research, multiple different activation functions have been investigated. However, in every experiment, only one activation function has been used for all layers. No distinction has been made in activation functions among neural network layers. This decision is taken to reduce the experimental configuration space. However, as the DQN does not show the desired performance, limiting the experiment space has shown to be an impactful decision. Complicated environments might not be fully comprehended by the current neural network, and therefore it is recommended to simultaneously apply multiple different activation functions within the same neural network.

Optimiser choice

The optimiser of choice has an impact on a wide range of aspects. The most important aspect that is impacted by the optimizer is its ability to escape local optima. Not being able to escape local optima could be the potential cause for the inability to learn of the DQN algorithm in this paper. Gradients that show unstable behaviour can also be tackled with choosing the right optimizer, The problem type and the network architecture are determinants of which optimiser fits best.

In the experiments conducted, multiple optimisers have been applied with different performances. The current optimiser, Adam, showed the best performance. Therefore experimenting more extensively with optimisers is not the first priority in an attempt to improve the DQN. However, when configuration choices in the neural network change, it might be worthwhile to revisit the choice of optimiser. Certain changes in configuration could require a different optimiser to achieve the desired performance.

Weight initialisation

Setting the weights of each layer in a neural network to the right values can make a difference. First of all, setting the weights to values that are too large can cause the neural network to end up in a sub-optimal state. Secondly, setting all weights within a layer to the same value can prevent the neural network from learning in an effective way. The neural network could also become prevented from learning in specific cases where the gradients explode or vanish during backpropagation. This is also caused under certain circumstances, where the initial weight configurations cause the problem.

There are many other events that are impacted by weight initialisation, but the aforementioned events mentioned are the most relevant for the performance of the

DQN presented in this paper. While experimenting, the weight initialisation was not taken into consideration to keep the experiment space within certain bounds. However, as mentioned above, in the case of a DQN that does not show the desired performance, it is recommended to experiment with network weight initialisation as well. This makes weight initialisation a promising path to dive into when building upon this research.

Choosing a different architecture

Instead of further experimenting with the current neural network architecture, exploring different neural network architectures to solve the problem has the potential to enable performance improvements. While currently a FFNN is used, pointer networks have proven to be an interesting architecture to explore as well. An example of these networks are the heterogeneous Attention Model (HAM) presented in Luttmann and Xie (2024).

In their paper, Luttmann and Xie (2024) solve the Mixed-shelves Picker Routing Problem by defining it as a MDP, with a heterogeneous graph representing the problem state. This is done to be able to input the problem to their HAM. Two different note types are distinguished in the graph: shelves and SKUs. Only one unique edge is used, namely the assignment edge. This edge connects shelves and SKUs. The nodes include their respective features, such as the demand of an SKU.

The problem presented in this paper is also defined as a MDP, which is the first step to making it eligible to be solved by HAM. Additionally, Abdel-Hamid and Borndörfer (1994) explain how the SLAP can be transformed into a bipartite graph. The problem being able to be defined as a bipartite graph is the next step to make it eligible to be solved by HAM. It should still be investigated how the different nodes and edges, including the context node, should be adapted to fit the problem and the architecture. However, the aforementioned points, together with the promising results of HAM, show that the HAM is an interesting architecture to explore further by applying it to solve the DSLAP.

Feature space

Like the neural network architecture, the feature space is an essential contributor to the DQN performance and can also be configured in many different ways. There is not one best configuration, and multiple different configurations may work. Still, the configuration has a big impact on the model performance. The neural network performance is only as good as the quality of the features it receives as input.

Identifying relevant features

Defining the feature space starts with identifying the relevant features. Here, relevant is defined as informative for the neural network, containing the data that is required to make informed decisions. This requires a thorough understanding of the data and the goal of the DQN.

The features identified as being relevant are discussed in Section 3.4.1. For future research, it is important to identify the relevant features again and to eliminate irrelevant features. Irrelevant features increase the complexity of the neural network,

without making a positive contribution to its performance. Including solely relevant input features enhances the convergence rate of the training process.

Multiple techniques that aid in feature selection exist. They are useful to find the set of features that fits the problem and dataset best. Quantitative techniques include methods that compute the correlation between features and the output variable. A qualitative option is to gather the opinions of warehousing experts. Both methods have not been conducted in this research, but could contribute to DQN performance by increasing the understanding of which features are most relevant.

Feature engineering

Feature engineering has the potential to improve the quality and relevance of features, by performing transformations to existing features. In the current paper, each set of features is standardised, leading to values between -1 and 1. This reduces the scale of the features, making them more easily interpretable for the neural network. Besides standardisation, no other methods of feature engineering are conducted. Nevertheless, feature engineering should be investigated when improving the DQN at hand because it can potentially lead to improvements in performance.

Action space

Like constructing a suitable neural network architecture and defining a relevant feature space, a well-considered action space is as important of a contributor to the model performance. This is mainly because the action space is part of the determinants of the complexity of the problem at hand. A smaller and well-defined action-space allows the algorithm to learn more effectively and efficiently.

In the world of the DSLAP problem, the action space is fairly easy to define in broad terms. Every action space is linked to the decision of where to store a SKU within the warehouse. One main issue faced within this problem, is how to include the decision of whether or not moving a SKU.

To solve this problem, it is possible to expand the decision phase to two stages. The first stage being whether or not to move the SKU, and the second stage being the exact location to which the SKU is moved to. This second stage is only executed when it is decided to move the SKU in the first stage. If it is decided to leave the SKU at its current location, no second decision stage is executed. In the presented DQN, only the second decision stage is included in the action space. Here, one of the decision locations is the current location, which results in the SKU staying at the location it currently is. Therefore, this decision is included but not explicitly programmed.

The reason to include this first decision is because it is possible that in many out-of-stock cases, it is undesirable to move a SKU. Moving a SKU costs time, and this time has to be compensated for by the time that is saved during order-picking after the SKU is moved. Therefore, it is expected that including this first decision leads to a more realistic and better performance in practice. On top of that, it may also stop the random behaviour the algorithm is currently showing. The disadvantage is, however, that including an extra stage makes the action space more complex.

A way of decreasing the complexity of the action space, would be to group the storage locations. Instead of using all warehouse storage locations, making use of a class-based storage allocation helps to make the problem more easily solvable for the

model. This is shown in Rimélé et al. (2021), who apply DQN with Monte Carlo search to solve the online DSLAP by using class-based storage allocation. This decreases the action space and therefore decreases the complexity as well. It is likely to enhance the learning process and possibly the performance as well.

Reward function

Currently, adaptions of the reward function proposed by Troch et al. (2023) are applied in the DQN. Within literature, many other reward functions have been proposed and applied. Choosing a suitable reward function is essential for the DQN, because it is the main way for the algorithm to receive feedback on the actions taken and thus to learn from its actions. A reward function that is in line with the objectives helps the DQN to learn the desired behaviours and to escape local optima, with the result of learning more effectively.

As mentioned, multiple versions of the presented reward function have been included in experiments. Since the performance is not as desired, experimenting with other reward functions may be beneficial. Additionally, including a way to take into account the costs of moving an SKU is an interesting future research prospect. This can not only be done by adapting the action space, but also by adapting the reward function by including costs when moving a SKU.

Hyperparameter tuning

As a final and obvious point, further hyperparameter experimentation should be conducted to comb as many hyperparameter combinations as possible. In addition to that, hyperparameter tuning is a reoccurring process that should be conducted again when changes to the DQN are implemented. This makes it a constant important subject, even though it has already been done. Since the hyperparameters have a direct impact on the model performance, it is essential to thoroughly conduct this stage in different experiments.

6.3.3 Practical Implications and Real-World Applicability

Even though one of the objectives of this research is to bridge the gap between theory and practice, the methods presented in this research are not applicable in practice yet. Further research is required, first to improve the performance of the DQN algorithm on the DSLAP in a dynamic warehouse environment.

Once the DQN algorithm shows promising results in a mock-up warehouse, the next step towards real-world applicability can be taken. This next step includes testing the DQN on an existing warehouse with its data. Only after this step, the first conclusions can be drawn about the real-world applicability of the DQN in this setting.

6.4 Summary

The experiments presented in this chapter do not perform better than the benchmarks presented. Some experiments show that the DQN algorithm improves the loss values. However, these improvements are not translated to improvements in reward values.

Generally speaking, there are many different factors that influence the performance of the DQN algorithm. These factors are presented one by one in this chapter. A final factor that impacts the performance of a research, is by making many changes compared to previous research.

As mentioned before, this paper differs from the research of Troch et al. (2023) on several core components. The main differences were removing the simulation part of the DQN and working with a dynamic warehouse environment. Changing multiple core characteristics compared to another research greatly increases the probability of wrong configurations. This could potentially be the main cause of the model performance.

In the next chapter, the final conclusions and recommendations are presented. This consists of a summary of the findings, the contributions of this research, the limitations and future research directions.

Chapter 7

Conclusions and Recommendations

This paper aims to answer the research question *'How can the DSLAP be solved by applying DQN in the context of a dynamic warehouse environment, to decrease the distance travelled during the order picking phase compared to existing benchmarks?'*, by applying a literature review, experimenting with a model and analysing the results. In this section, first the main takeaways from the research are presented. Then, the theoretical and practical relevance is discussed, after which the discussion continues with the limitations of this research. Finally, directions for future research are touched upon.

7.1 Summary of Findings

After an extensive literature review, it can be concluded that ML methods can be effective in tackling the DSLAP. Especially many papers that successfully apply SL and UL to this problem have been published. The combination of DRL with the DSLAP is still a developing area, where little research has been done. Even though there have been a few successful attempts, many attempts were focused on small warehouses with limitations, such as stable demand or clustered storage allocation. This paper aims to contribute to the DRL part of the DSLAP research, by applying DQN without simulation to a different context of the DSLAP within a dynamic warehouse environment.

Using a reward function that aims to minimise the distance travelled while order picking, the DQN is built and experiments are run and analysed. The configured DQN without simulation presented in this paper did not show improvements when applied to the DSLAP. Therefore, this paper does not present an immediate example of how the DSLAP can be solved by applying a less complex version of the DQN. However, it still provides a first step into the subject of applying DQN with a less complex configuration on a different type of DSLAP within a dynamic warehouse environment. After analysing the results, improvement points and future research directions are presented, including the promising field of pointer networks. These insights aid researchers to further extend this field of research.

7.2 Contribution to Theory and Practice

As the literature review section shows, DQN has the potential of being a useful method to solve the DSLAP. Despite the potential of DQN, this research shows that DQN is a

method that requires careful configuration to make use of its potential. This makes the DQN less accessible in practice, because it takes time and expert knowledge to properly apply the method. However, once configured right, literature has shown that the performance of DQN is almost human-like.

Since the focus in this research is on a dynamic warehouse environment, the complexity of the fluctuating demand makes the work of the DQN more complicated. This research is one of the few DQN applications on the DSLAP where the location decision is not made for every incoming SKU, and that takes place within a dynamic warehouse environment. Additionally, it does this without a simulation environment. With these configurations, it hopefully aids and stimulates colleagues to further dive into the topic of DRL applied on this type of DSLAP within a dynamic warehouse environment.

Focusing research on a dynamic warehouse environment makes it more realistic, and therefore more relevant in practice. The DQN may work well on this type of DSLAP with stable demand, but the question remains: is DQN as still valuable for a warehouse in a less static environment? This research aims to trigger researchers in answering that question, by providing a starting point, by presenting a thorough analysis of the results, and by guiding towards future research directions.

7.3 Limitations and Future Research Directions

As discussed in Section 6.3.2, the network architecture is expected to be one of the main limitation of the research presented. The depth and width of the architecture are arguably too simple to comprehend complex environments. Additionally, the activation functions have not been fully investigated, because only one single activation function for the entire network was used. Like adapting the network size, adjusting the activation functions also allows for the comprehension of more complex environments. Two final, possibly impactful, adaptations that can be done to the current network architecture are choosing different weight initialisation and a different optimizer. Both can prevent the solver to get stuck in local optima and therefore enhance its learning potential.

Being a more impactful decision, the choice for the network architecture is discussed separately. While a FFNN is currently being run, colleagues may decide to choose different architectures in the future. A promising architecture, but non-existent in the field of the DSLAP, are pointer networks. An example of this is the HAM presented by Luttmann and Xie (2024). The main requirements for applying this network is that the problem must be formulated in a MDP, using a heterogeneous bipartite graph with two types of nodes and one type of edge. The MDP representation of the DSLAP as proposed in this paper can be used for this. Abdel-Hamid and Borndörfer (1994) show how the DSLAP can be translated into a bipartite graph with two types of nodes and one type of edge. This information, combined with the paper of Luttmann and Xie (2024), provides the first step towards implementing pointer networks to solve the DSLAP.

Another determinant for the model performance is the feature space. It is essential to provide the right information to the model, because without this it will never perform as desired. Future research should focus on applying methods, such as correlation methods, to identify the features that have the best prediction ability for the

decisions made in the DSLAP. In addition to that, feature engineering should be explored for the identified features, to comprehensibly define the relevant features.

Thirdly, the action space is identified as one of the limitations. The current action space is large, which makes the problem even more complex in combination with the feature space. Instead of using all warehouse storage locations, making use of e.g. class-based locations could make the problem more easily solvable for the model. This has been done in present papers as well, see e.g. Rimélé et al. (2021).

For future papers, it is also interesting to include the decision of whether or not to move a product. This might make the problem more realistic, because this is the first decision to be taken in a real-life warehouse. Since changing a product location also costs time, it may be that most of the time moving a product is not desirable. This can be better simulated when the decision of whether or not to move the problem is included in the action space.

The costs that arise when moving a product can also be included in the reward function. With this fourth and final limitation and direction for future research, a way to make the simulation more realistic is presented. Future research could dive into the reward function, to include this addition. On top of that, the reward function in general is an important determinant for the model behaviour, because it is the only way for the model to receive feedback. Since the used reward function does not include the costs of moving a product, this could be part of the reason why it shows random behaviour. This causes the product to continuously keep moving products, because there is no costs attached to moving products.

As a conclusion, as mentioned in Section 6.4, changing multiple core components compared to previous research makes this research prone to errors. Therefore, for future research it is suggested to take Troch et al. (2023) as a foundation, to then first change one main characteristic and check the performance. From there, one can work towards a different model step-by-step. An example would be to only change the warehouse environment to a dynamic one. From there, the model presented by Troch et al. (2023) can be adapted to suit the dynamic warehouse environment.

7.4 Final Remarks

In this paper, information is gathered to provide researchers with a stepping stone towards DRL applied on the DSLAP solved for every stock-out SKU, because the DSLAP with different and longer periods is still a promising field to explore further. This paper shows an attempt to solve this type of DSLAP with DQN, and presents information to build upon this attempt for future research. This information contains both possible adaptations of the current DQN configuration, as well as new venues that should be explored.

Appendix A

Glossary

Here, a list of definitions of abbreviation used in this paper is presented.

- SLAP - Storage location assignment problem
- DSLAP - Dynamic storage location assignment problem
- MDP - Markov decision process
- RL - Reinforcement learning
- DRL - Deep reinforcement learning
- SL - Supervised learning
- UL - Unsupervised learning
- DL - Deep learning
- FFNN - Feedforward neural network
- HAM - Heterogeneous attention model
- KPI - Key performance indicator
- ML - Machine learning
- SKU - Stock keeping unit
- I/O - Inboud/outbound
- NN - Neural network
- SVM - Support vector machine
- DT - Decision tree
- DQN - Deep Q learning
- PPO - Proximal policy optimisation
- A2C - Advantage actor critic

- POMDP - Partially observable MDP
- FOMDP - Fully observable MDP
- DP - Dynamic programming
- MC - Monte carlo
- TD - Temporal difference
- FAM - Function approximation methods
- SARSA - State-action-reward-state-action
- SGD - Stochastic gradient descent
- MILP - Mixed-integer linear programs
- TS - Tabu search
- GA - Genetic algorithm
- DES - Discrete event simulation
- AB - Agent-based simulation
- MM - Maintenance management
- QM - Quality management
- PPC - Production planning and control
- SCM - Supply chain management
- DoS - Duration-of-stay
- PCA - Principal component analysis
- LSTM - Long short-term memory
- PFS - Pick frequency score

Appendix B

Pseudocode Markov Decision Process

The MDP consists of three main parts: the initialise part, the reset part and the step part. The initialise part is only run once, when building the environment. The reset part is run at the beginning of each episode, to begin with a clean environment. The step part is the main part, and this is run to execute all required action every time step.

Algorithm 1: Pseudocode MDP: initialise

Input: I; L; W; dem_{it} $\forall I, T$ matrix; inv_{it} $\forall I, T$ matrix; cap_i $\forall I$ matrix
Output: None

Import datasets
Create warehouse L and W
    Assert: $I < L * W$
Create d_l $\forall L$ matrix
Determine A and S

Algorithm 2: Pseudocode MDP: reset

Input: None
Output: s_{t+1}

Initialise all variables and parameters
Create initial x_{ilt} $\forall I, L, T$ matrix
Find first out-of-stock item
    First out-of-stock item becomes $i_{\text{at hand}}$
Retrieve $dem_{i_{\text{at hand}}t}$
    Loop through T
        Update inv_{it} $\forall I$ every t
        Loop through I
            If $inv_{i_{\text{at hand}}t} \leq 0$
                $ITER_t$ += 1
                Replenish $inv_{i_{\text{at hand}}t}$
                Break both loops
Create s_{t+1}

Algorithm 3: Pseudocode MDP: initialise

Input: a_t
Output: s_{t+1}; $R(s_t, a_t)$; done

If $t > T$:
    Done = true
    Terminate simulation
Elif action location is different than current location:
    If action location is already taken:
        Swap location
    Else:
        Put item in action location
        Empty old location
Compute $R(s_t, a_t)$
Find new $i_{\text{at hand}}$
Create s_{t+1}

Appendix C

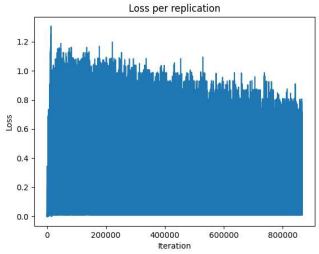
Hyperparameter Tuning

Table C.1: Hyperparameter tuning table.

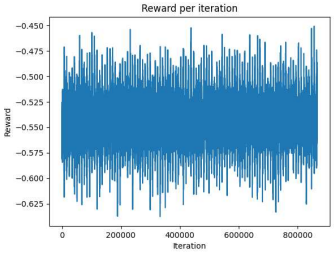
Hyperparameter	Range
γ	0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 0.95, 0.99
lr	0.0005, 0.001, 0.005, 0.01, 0.05, 0.1
replay buffer size	100, 500, 1,000, 3,000, 6,000, 10,000
batch size	16, 32, 64, 128, 256, 512
target update	10, 20, 40, 80, 100, 200
eps start	0.9, 0.95, 1
eps end	0, 0.05, 0.1
eps decay	20,000, 40,000, 60,000, 80,000, 100,000, 400,000, 800,000, 1,200,000

Appendix D

Experimental Results



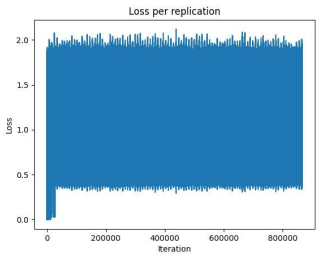
(a) Total loss plot



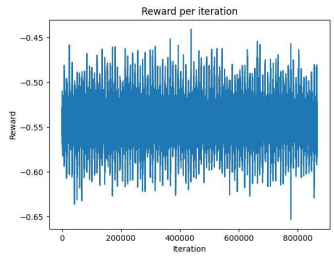
(b) Total reward plot

(c) Minimum and maximum reward plot

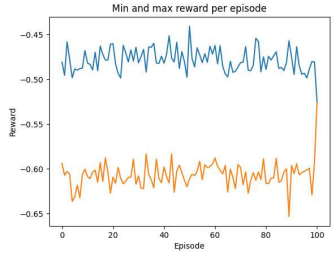
Figure D.1: Experiment 2 results.



(a) Total loss plot

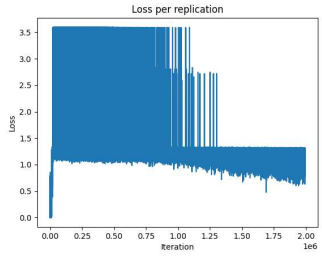


(b) Total reward plot

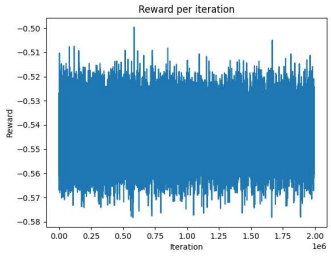


(c) Minimum and maximum reward plot

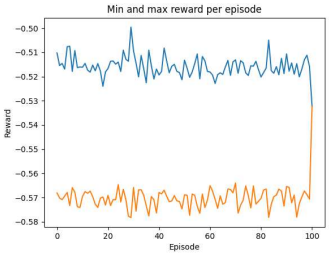
Figure D.2: Experiment 3 results.



(a) Total loss plot

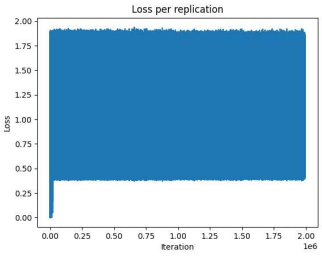


(b) Total reward plot

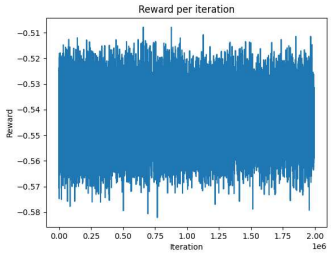


(c) Minimum and maximum reward plot

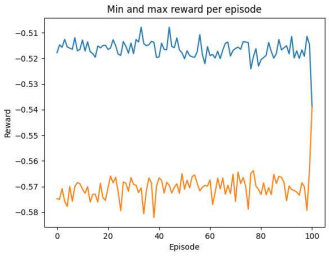
Figure D.3: Experiment 5 results.



(a) Total loss plot

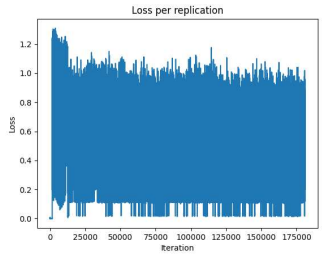


(b) Total reward plot

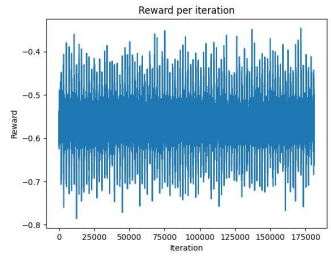


(c) Minimum and maximum reward plot

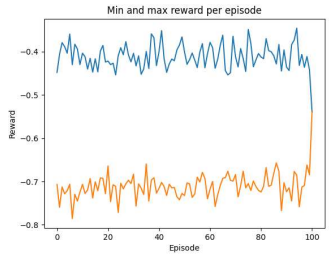
Figure D.4: Experiment 6 results.



(a) Total loss plot

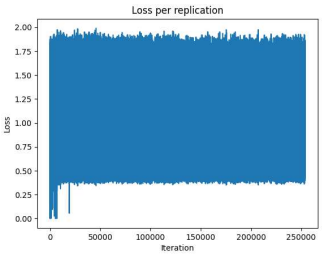


(b) Total reward plot

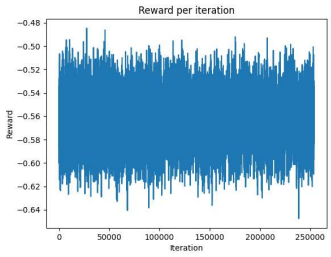


(c) Minimum and maximum reward plot

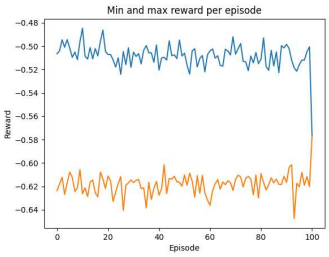
Figure D.5: Experiment 8 results.



(a) Total loss plot

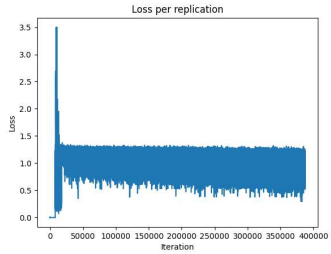


(b) Total reward plot

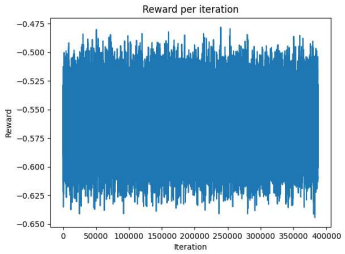


(c) Minimum and maximum reward plot

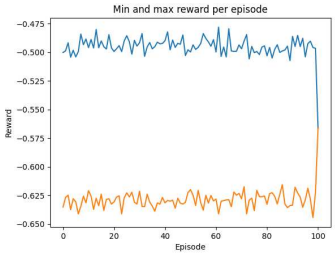
Figure D.6: Experiment 9 results.



(a) Total loss plot

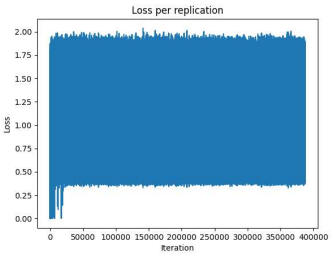


(b) Total reward plot

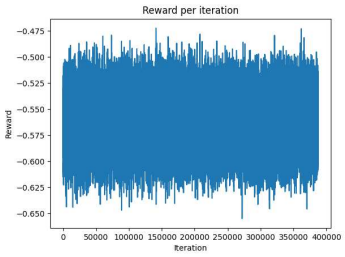


(c) Minimum and maximum reward plot

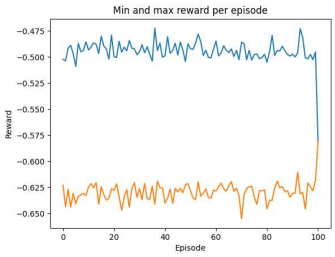
Figure D.7: Experiment 11 results.



(a) Total loss plot



(b) Total reward plot



(c) Minimum and maximum reward plot

Figure D.8: Experiment 12 results.

Bibliography

Abdel-Hamid, A. A.-A. and Borndörfer, R. (1994). On the complexity of storage assignment problems.

Alpaydin, E. (2021). *Machine learning*. Mit Press.

Arulkumaran, K., Deisenroth, M. P., Brundage, M., and Bharath, A. A. (2017). Deep reinforcement learning: A brief survey. *IEEE Signal Processing Magazine*, 34(6):26–38.

Berns, F., Ramsdorf, T., and Beecks, C. (2021). Machine learning for storage location prediction in industrial high bay warehouses. In *Pattern Recognition. ICPR International Workshops and Challenges: Virtual Event, January 10–15, 2021, Proceedings, Part IV*, pages 650–661. Springer.

Bertolini, M., Mezzogori, D., Neroni, M., and Zammori, F. (2021). Machine learning for industrial applications: A comprehensive literature review. *Expert Systems with Applications*, 175:114820.

Cruz-Domínguez, O. and Santos-Mayorga, R. (2016). Artificial intelligence applied to assigned merchandise location in retail sales systems. *South African Journal of Industrial Engineering*, 27(1):112–124.

Dixit, A., Shah, B., and Sonwaney, V. (2020). Picking improvement of an fmcg warehouse: a lean perspective. *International Journal of Logistics Economics and Globalisation*, 8(3):243–271.

Franzke, T., Grosse, E. H., Glock, C. H., and Elbert, R. (2017). An investigation of the effects of storage assignment and picker routing on the occurrence of picker blocking in manual picker-to-parts warehouses. *The International Journal of Logistics Management*, 28(3):841–863.

Frazelle, E. H. (1992). Stock location assignment and order picking productivity.

Glover, F. (1990). Tabu search: A tutorial. *Interfaces*, 20(4):74–94.

Grosse, E. H., Glock, C. H., and Jaber, M. Y. (2013). The effect of worker learning and forgetting on storage reassignment decisions in order picking systems. *Computers & Industrial Engineering*, 66(4):653–662.

Grznár, P., Krajčovič, M., Gola, A., Dulina, L., Furmannová, B., Mozol, Š., Plinta, D., Burganová, N., Danilczuk, W., and Svitek, R. (2021). The use of a genetic algorithm for sorting warehouse optimisation. *Processes*, 9(7):1197.

Horgan, D., Quan, J., Budden, D., Barth-Maron, G., Hessel, M., Van Hasselt, H., and Silver, D. (2018). Distributed prioritized experience replay. *arXiv preprint arXiv:1803.00933*.

Jia, J. and Wang, W. (2020). Review of reinforcement learning research. In *2020 35th Youth Academic Annual Conference of Chinese Association of Automation (YAC)*, pages 186–191. IEEE.

Kübler, P., Glock, C. H., and Bauernhansl, T. (2020). A new iterative method for solving the joint dynamic storage location assignment, order batching and picker routing problem in manual picker-to-parts warehouses. *Computers & Industrial Engineering*, 147:106645.

Lambora, A., Gupta, K., and Chopra, K. (2019). Genetic algorithm-a literature review. In *2019 international conference on machine learning, big data, cloud and parallel computing (COMITCon)*, pages 380–384. IEEE.

Leon, J. F., Li, Y., Martin, X. A., Calvet, L., Panadero, J., and Juan, A. A. (2023). A hybrid simulation and reinforcement learning algorithm for enhancing efficiency in warehouse operations. *Algorithms*, 16(9):408.

Li, J., Moghaddam, M., and Nof, S. Y. (2016). Dynamic storage assignment with product affinity and abc classification—a case study. *The International Journal of Advanced Manufacturing Technology*, 84:2179–2194.

Li, M. L., Wolf, E., and Wintz, D. (2019). Duration-of-stay storage assignment under uncertainty. *arXiv preprint arXiv:1903.05063*.

Lin, L.-J. (1992). Self-improving reactive agents based on reinforcement learning, planning and teaching. *Machine learning*, 8:293–321.

Luttmann, L. and Xie, L. (2024). Neural combinatorial optimization on heterogeneous graphs: An application to the picker routing problem in mixed-shelves warehouses. In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 34, pages 351–359.

Mahraz, M.-I., Benabbou, L., and Berrado, A. (2022). Machine learning in supply chain management: A systematic literature review. *International Journal of Supply and Operations Management*, 9(4):398–416.

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

Nian, R., Liu, J., and Huang, B. (2020). A review on reinforcement learning: Introduction and applications in industrial process control. *Computers & Chemical Engineering*, 139:106886.

Niu, F. and Wang, Z. (2019). Model based reinforcement learning for simplified storage assignment optimization.

Pan, J. C.-H., Shih, P.-H., Wu, M.-H., and Lin, J.-H. (2015). A storage assignment heuristic method based on genetic algorithm for a pick-and-pass warehousing system. *Computers & Industrial Engineering*, 81:1–13.

Petersen, C. G., Aase, G. R., and Heiser, D. R. (2004). Improving order-picking performance through the implementation of class-based storage. *International Journal of Physical Distribution & Logistics Management*, 34(7):534–544.

Pierre, B., Vannieuwenhuyse, B., Dominanta, D., and Van Dessel, H. (2003). Dynamic abc storage policy in erratic demand environments. *Jurnal Teknik Industri*, 5(1):1–12.

Reyes, J., Solano-Charris, E., and Montoya-Torres, J. (2019). The storage location assignment problem: A literature review. *International Journal of Industrial Engineering Computations*, 10(2):199–224.

Rimélé, A., Grangier, P., Gamache, M., Gendreau, M., and Rousseau, L.-M. (2021). E-commerce warehousing: learning a storage policy. *arXiv preprint arXiv:2101.08828*.

Sadiq, M., Landers, T. L., and Don Taylor, G. (1996). An assignment algorithm for dynamic picking systems. *IIE transactions*, 28(8):607–616.

Schaul, T., Quan, J., Antonoglou, I., and Silver, D. (2015). Prioritized experience replay. *arXiv preprint arXiv:1511.05952*.

Soysal, G. P. and Krishnamurthi, L. (2012). Demand dynamics in the seasonal goods industry: An empirical analysis. *Marketing Science*, 31(2):293–316.

Troch, A., Mannens, E., and Mercelis, S. (2023). Solving the storage location assignment problem using reinforcement learning. In *Proceedings of the 2023 8th International Conference on Mathematics and Artificial Intelligence*, pages 89–95.

Waubert de Puiseau, C., Nanfack, D. T., Tercan, H., Löbbert-Plattfaut, J., and Meisen, T. (2022). Dynamic storage location assignment in warehouses using deep reinforcement learning. *Technologies*, 10(6):129.

Wenzel, H., Smit, D., and Sardesai, S. (2019). A literature review on machine learning in supply chain management. In *Artificial Intelligence and Digital Transformation in Supply Chain Management: Innovative Approaches for Supply Chains. Proceedings of the Hamburg International Conference of Logistics (HICL), Vol. 27*, pages 413–441. Berlin: epubli GmbH.

Wutthisirisart, P., Noble, J. S., and Chang, C. A. (2015). A two-phased heuristic for relation-based item location. *Computers & Industrial Engineering*, 82:94–102.

Yang, C.-L. and Nguyen, T. P. Q. (2016). Constrained clustering method for class-based storage location assignment in warehouse. *Industrial Management & Data Systems*, 116(4):667–689.