Deep reinforcement learning for solving a multi-objective online order batching problem

Beeks, M.S.

Award date:
2021

Department of Industrial Engineering and Innovation Sciences
Information System Group

Master thesis: Deep reinforcement learning for solving a multi-objective online order batching problem

Martijn Beeks (1389440)

In partial fulfillment of the requirements for the degree of **Master of science in Operations Management and Logistics**

Supervisors:
Dr. Y. (Yingqian) Zhang - TU/e
Dr. R. (Remco) Dijkman - TU/e
C. (Claudy) van Dorst - Vanderlande Industries
S. (Stijn) de Looijer - Vanderlande Industries

Eindhoven, Wednesday 14th April, 2021

Abstract

The primary purpose of this master thesis project is to develop a deep reinforcement learning approach to solve an online order batching problem. This project takes place at Vanderlande Industries (VI), the global market leader for value-added logistic process automation at airports, the parcel market and the warehousing market. In the warehousing market, VI designs warehouse fulfilment solutions which incorporate receiving, storing, order picking and the consolidation of goods. The specific problem concerns an Online Order Batching and Sequencing Problem in a warehousing concept called DeepRele. DeepRele consists of a storage area for manual picking with order pickers, a storage area for automated picking with shuttles and some subsequent consolidation processes. In order to reflect real-world characteristics, this problem is addressed as a multi-objective problem, large instances are considered and a dynamic environment is used that is subjected to change. In order to cope with these challenges, this work presents a Deep Reinforcement Learning (DRL) approach and a Bayesian optimization technique for reward shaping. The first objective within this multi-objective problem is the percentage of tardy orders (orders that are finished after their respective cutoff time). The second objective is the order picking costs per order. The proposed solutions are used in a benchmark study together with traditional heuristics to asses performance. The DeepRele system is modeled as a Semi Markov Decision Process that allows for different order batching and sequencing policies to be tested. The DRL approach in combination with reward shaping using Bayesian optimization outperforms these heuristics on average with 50% on the percentage of tardy orders and with 5% in terms of average order picking costs over the four conducted experiments. With this, the DRL approach with reward shaping provides a new intuitive solution for multi-objective online order batching problems. Lastly, this work provides additional analysis on the learned DRL approaches including approximating the learned DRL policies using decision trees to infer logic and provide explainable decision rules.

Executive summary

Problem context
This thesis project takes place at Vanderlande industries and presents a deep reinforcement learning approach to solve a multi-objective online order batching problem. VI is the global market leader for value-added logistic process automation at airports, the parcel market and the warehousing market. In this warehousing market, VI designs warehouse fulfilment solutions which incorporate receiving, storing, order picking and the consolidation of goods. In order to tackle arising challenges in the e-commerce market, VI has developed a new fulfillment concept called DeepRele. DeepRele consists of a storage area for manual picking with order pickers, a storage area for automated picking with shuttles and some subsequent consolidation processes. This new warehousing concept tries to tackle several challenges within the e-commerce market: strong expected order growth, next day or even same day deliveries and highly volatile demand during days as Single Day, Black Friday and Thanksgiving. Within this new warehousing concept, this thesis project focuses on solving the online Order Batching and Sequencing Problem (Online Order Batching and Sequencing Problem (OOBSP)) where for each order, either a picked-by-batch or picked-by-order decision needs to be made and sequenced in real-time. The subsequent batching operation is done using a heuristic. These batching and sequencing decisions can have extensive operational consequences as a pick-by-batch decision reduces picking costs by more efficient picking but requires a longer lead time as orders within a batch needs to wait on each other. Contrary, a pick-by-order decision has shorter lead times but higher picking costs as the picking operation is less efficient. By this batching decision and the subsequent sequencing decision of batches, the system tries to minimize the number of tardy orders (late orders that leave the system after their respective cutoff time) and order picking costs. Similar sequential decision making problems have been addressed by DRL but somewhat lacked real-world characteristics (Waschneck et al., 2018; Hubbs et al., 2020; Shi et al., 2020). This specific problem has been addressed by DRL before by Cals et al. (2021) but contains several assumptions that result in a theoretical environment. In order to reflect real-world characteristics, this problem is addressed as a multi-objective problem, considers large problem instances and uses a dynamic environment that is subjected to change.

Research approach
In the recent years, Reinforcement Learning (RL) has demonstrated impressive results for sequential decision making problems in a wide range of domains. This approach is the study on how an agent can interact with an environment to learn a policy which maximizes expected cumulative rewards for a task (Sutton et al., 1998). In addition to RL, deep reinforcement learning uses recent advancements in Deep Neural Networks (DNN) to cope with the curse of dimensionality resulting in massive state spaces. In this work, DRL addresses the OOBSP of DeepRele with real-world characteristics and is guided by the following research question:

How can deep reinforcement learning solve a multi-objective variant of the online order batching and sequencing problem with real-world characteristics?

In this thesis project, the focus is on solving the online order batching and sequencing problem in a DeepRele system whereas additional processes as replenishment of goods, labeling and closing are left out of scope. Furthermore, only the batching decision (pick-by-batch / pick-by-order) is performed by the DRL approach, the subsequent batching operation is performed by a heuristic. Lastly, designing a routing strategy is not included in this thesis project to solely focus on the online order batching and sequencing problem.

Methods
In order to solve the introduced problem, this project contains two solution methods. In the first method, a DRL approach is proposed that solves the OOBSP in a Semi Markov Decision Process (SMDP). This SMDP can be considered as the "environment" in which a DRL approach can

interact and learn by trial and error. The DRL approach has been designed in such a way that this can cope with real-world characteristics. The DRL approach uses a Proximal Policy Optimization model which strikes a favorable balance between sample efficiency, simplicity and computational efficiency. Within the SMDP, an efficient simulation model is developed that mimics the dynamics of a DeepRele warehousing system. This simulation model is used as an approximation to the reality as current physical concepts are not available and it is not desirable to test policies in this way. In the second method, the DRL approach is complemented with a Bayesian optimization technique for reward shaping. This reward shaping process is considered as a hyper-optimization problem that is addressed by Bayesian Optimization. It is identified that shaping a reward function in a multi-objective real-world problem setting is considered difficult and the Bayesian optimization method tries to improve this process. In this reward function, each component attributes to a certain objective that the DRL approach tries to optimize. Specifically, setting these components proportionally to each other is difficult, and the Bayesian optimization method tries to find a set of weights that can be applied to the different components in this reward function.

Results

The two proposed solution methods have been compared in a benchmark study against more traditional batching and sequencing heuristics as the LST heuristic, BOC heuristic by Li et al. (2017) and the GVNS heuristic by Bustillo et al. (2015). In the first solution method, the DRL approach is combined with the LST sequencing heuristic and outperforms the traditional heuristics significantly in terms of tardy orders but not in terms of order picking costs which demonstrates the complexity of solving multi-objective optimization problems. Using a policy analysis, the learned DRL policy decreases its pick-by-batch ratio just before cutoff moments in order to make sure that orders with imminent cutoff times are picked-by-order and therefore have a short lead time to leave the system in time. These DRL approaches have been trained on specific instances and a robustness analysis has been performed to observe generalization of these approaches. These approaches are robust to small changes (10%) in throughput and resource capacities but find it hard generalize when a complete different experiment is presented. As mentioned before, the second solution method presents a Bayesian optimization technique for reward shaping in order to complement the DRL approach. The DRL approach with reward shaping outperforms the DRL approach without reward shaping both in terms of tardy orders and order picking costs. In this reward shaping process, a Bayesian optimization technique finds a set of weights that represent the priority for each objective within the reward function. These obtained weights confirm a prior expectation where the DRL approach focuses during the day more on minimizing order picking costs whereas as during the end of a day, focuses more on minimizing tardy orders. Additionally, this work provides additional analysis on the learned DRL approaches including approximating the learned DRL policies using decision trees to infer logic and provide explainable decision rules.

Conclusions and recommendations

During this project, a DRL approach with reward shaping presented itself as a novel and exiting technique to solve the online order batching and sequencing problem of DeepRele with real-world characteristics. This approach addresses these real-world characteristics by taking into account a multi-objective variant of the OOBSP, a large problem instance size and a dynamic environment with uncertainty in order arrivals in an online setting. The proposed DRL approaches with reward shaping learn a policy that outperforms more traditional heuristics in the OOBSP. These DRL approaches with reward shaping provide a better trade-off to the multi-objective problem whereby prioritizing a certain objective at the right point in time is key. Although the scope of this thesis project includes fewer components than can be implemented, it can be concluded that this research shows great potential when solving real-world scenarios of the OOBSP with a DRL approach using Bayesian optimization for reward shaping. With this, it can be recommended to extend this work by using an even more detailed simulation model, to use exact storage locations of SKUs and optimize a routing strategy, further industrializing the DRL approaches with reward shaping by translating these into explainable heuristics using Explainable AI (XAI) and lastly, provide a plan for bringing a reinforcement learning approach in production.

Preface

This report concludes my thesis project and thereby the master Operations Management and Logistics at the Eindhoven University of Technology. I am grateful for this journey and some of the most important things that I learned are captured in this quote by Michael Jordan: "Talent wins games, but teamwork and intelligence win championships."

With this, I would like to take the opportunity to express my gratitude towards a few people who have guided me throughout this master thesis project. First of all, I would like to thank my university supervisor Yingqian Zhang for her guidance and support. During our valuable meetings, we discussed and mapped out directions for this research that helped me a lot in understanding the field of deep reinforcement learning. Furthermore, my second university supervisor Remco Dijkman helped me by providing a critical perspective to the assumptions that I had taken and definitely helped me to improve the master thesis project.

Also on the side of Vanderlande, I would like to thank Claudy van Dorst for her extensive expertise and knowledge on the warehousing concepts and she provided me with a lot of insights that improved my work. Periodically, we met to discuss the results and to verify that the thing we were doing was still the correct one. Lastly, I would to thank Caglar Seneras and Stijn de Looijer that both supervised me from the New Technology team by providing valuable feedback and support.

Martijn Beeks

Contents

List of Figures

List of Tables

Abbreviations

DNN Deep Neural Networks. 1, 70

DRL Deep Reinforcement Learning. 1–4, 13, 15, 18–22, 24–26, 30, 31, 33, 34, 36, 37, 39–42, 50, 51, 53–70

GtP Goods-to-person. 1, 27, 31, 35, 36, 38, 49, 51, 54, 61, 66

MOO Multi Objective Optimization. 1, 23

MORL Multi Objective Reinforcement Learning. 1, 23

OOBSP Online Order Batching and Sequencing Problem. 1–7, 9–11, 13, 18, 21, 23, 24, 26, 27, 29–31, 33, 34, 37, 39–46, 48–51, 57, 58, 60, 63, 64, 67, 69, 70

PtG Person-to-Goods. 1, 27, 31, 35, 36, 38, 41, 46, 49, 51, 54, 58, 60, 61, 65, 66

RL Reinforcement Learning. 1, 2, 13–15, 19, 22, 24, 25, 70

VI Vanderlande Industries. 1, 3, 37, 38, 48, 49, 54

XAI Explainable A.I.. 1, 70

1. Introduction

1.1 Problem context

This thesis project takes place at Vanderlande industries and presents a deep reinforcement learning approach to solve a multi-objective online order batching problem. Vanderlande Industries (VI) is the global market leader for value-added logistic process automation at airports, the parcel market and the warehousing market. In the warehousing market, VI designs warehouse fulfilment solutions which incorporate receiving, storing, order picking and the consolidation of goods. In order to tackle arising challenges in the e-commerce market, VI has developed a new fulfillment concept called DeepRele. This new warehousing concept consists of a storage area for manual picking with order pickers, a storage area for automated picking with shuttles and some subsequent consolidation processes. DeepRele has been designed to tackle challenges in the e-commerce market: strong expected order growth (Statista, 2020), next-day or same-day deliveries (Yaman et al., 2012) and labour scarcity within the warehousing industry (Hamberg and Verriet, 2012). The most important challenge that DeepRele tries to tackle within the e-commerce market, are the varying workloads. Sale events such as Single's Day, Black Friday and Thanksgiving Weekend cause highly volatile demand that need to be aligned with current warehouse capacity. Peak days can have demand up to 3 times higher than average demand seen at Vanderlande's customers.

Traditional warehouse concepts as storage areas for manual picking (Person-to-Goods (PtG)) and storage areas for automated picking (Goods-to-person (GtP)) both have different perspectives in operational performance. PtG systems provide flexibility in order picking capacity but operate with relatively high lead times per order. On the other hand, GtP systems provide low lead times per order and reduces the requirement of manual labour. However, it cannot provide flexibility in order picking capacity and requires a significant higher investment than PtG systems. This trade-off between the GtP and PtG systems is leveraged in the new warehousing concept called DeepRele. DeepRele provides a combination of PtG and GtP picking areas and considers the flexibility of picking capacity, lead times, manual labour and initial investment (Figure 3.1). According to literature, three planning issues can be distinguished in warehouse operations (Caron et al., 1998): assignment of articles to storage locations (article location), transformation of orders into picking orders (order batching) and routing of pickers through the warehouse (picker routing). This project focuses on the online order batching and sequencing part where for each order, either a picked-by-batch or picked-by-order decision needs to be made and sequenced in real time. The subsequent batching operation is done using a heuristic. This problem is in literature related to the online order batching and sequencing problem (OOBSP) and both picking options have different consequences in terms of operational performance. According to Bartholdi III and Hackman (2018), cost of order picking is estimated to form 55% of the total warehousing costs and together with minimizing the number of 'late' orders are the main concerns in this project.

Although the OOBSP already has received some attention since its acknowledgement, recent developments in the field of deep reinforcement learning (DRL) gave rise to the question whether a DRL approach can also gain expertise in solving these kinds of optimization problems. Based on previous work of Vanderlande and Cals et al. (2021), the decision whether to pick-by-batch or pick-by-order, can be made using deep reinforcement learning. In the work of Cals et al. (2021), the authors demonstrated superior performance of a DRL model for only a single scenario. Within this single scenario, a DeepRele system was used that had a throughput of only 500 orders per hour. However, the complexity of the proposed solution did not allow the authors to solve a system with larger, more complex and more realistic through-puts. Furthermore, the current problem only includes a single objective: minimizing the number of tardy (or late) orders. Ideally, DeepRele would trade-off the number of tardy orders and the order picking time per order. Many optimization problems that are solved with DRL are restricted to single-objective problems (Peer

et al., 2018; Waschneck et al., 2018; Hubbs et al., 2020; Shi et al., 2020; Che et al., 2020; Cals et al., 2021). Whereas many (sequential) decision making problems in the real world try to optimize multiple objectives (Nguyen et al., 2020). In order to increase applicability of DRL models for optimization problems, this master thesis project addresses the following challenge: design a DRL approach to solve a multi-objective variant of the online order batching and sequencing problem with real-world characteristics.

1.2 Real-world operational decision problems

These real-world characteristics, have a multi-objective problem setting, have orders arriving into the system in real time (online problem setting), have large instance size problems arise and have environments that change dynamically. The proposed model should account for these challenges. This section introduces optimization problems solved by DRL, discusses the complexity of multi-objective problems, presents shortcomings of the work of Cals et al. (2021) and thereby, provides academic motivation for this project.

In the field of Operations Research and Computer Science, several optimization problems have been addressed with Deep Reinforcement Learning. Waschneck et al. (2018) demonstrates DRL as a superior method for Job Shop scheduling problem (JSP) that incorporates several manufacturing dynamics. Hubbs et al. (2020) provided a DRL-based scheduling algorithm that efficiently schedules batches for a chemical production site. Shi et al. (2020) designed a reinforcement learning algorithm that solves the problem of lot scheduling for discrete automated production lines with stochastic processing times. Of course finally, the work of Cals et al. (2021) solved the online order batching and sequencing problem for DeepRele application. All these findings proved that DRL can outperform simpler scheduling and allocation heuristics. However, the DRL models in all findings have been used to solve a single-objective optimization problem, address a relatively small instance size and some consider a somewhat static environment. As many decision making problems in the real world require the consideration of more than one objective, addressing this challenge could enhance the applicability of DRL models significantly. Multi-objective optimization problems are often conflicting so that the minimization often leads to maximization of another. According to Nguyen et al. (2020), an aligned reward function is of high importance when solving multi-objective optimization problems and problems with a dynamic environment using deep reinforcement learning. Additionally, Sutton et al. (1998) state that the performance of RL in general is heavily influenced by the way the problem is represented in terms of reward function. Finding an aligned reward function for DRL model is considered difficult according to Marom and Rosman (2018) and so, improving the process of finding an aligned reward function would enhance DRL approaches solving real-world operational decision problems. However, solving these problems pose additional challenges. In the work of Cals et al. (2021), orders do not enter the system in real time, small instances of the OOBSP were used and a fairly static environment was considered. In this static environment, a specific part of the day is selected to use for modeling. Selecting an entire day with changing dynamics in terms of order arrivals and resource capacity was not possible. And so, solving real-world operational decision problems includes a multi-objective aspect, a large instance size and a dynamic environment that is subjected to change.

1.3 Research questions

This thesis project aims to solve the OOBSP with real-world characteristics by using recent advancements in the machine learning domain. In order to guide this process, the following main research question is defined:

How can deep reinforcement learning solve a multi-objective variant of the online order batching and sequencing problem with real-world characteristics?

Answering this research question is structured by answering the following sub questions:

1. What are the existing methods that solve the OOBSP and where do they excel or fall short?
2. What is DRL and what are the applications for optimization problems?
3. In what ways can DRL be complemented so it can handle multi-objective problems?
4. What are the characteristics of the OOBSP present in the DeepRele warehousing concept?
5. How can the OOBSP be formulated in a way that enables DRL to solve a multi-objective variant of the problem with real-world characteristics?
6. What is the performance of such a method compared to existing techniques and how can these methods be explained?

1.4 Research design

The first three sub questions are answered in Chapter 2 by conducting a literature review. Question 1 provides an overview of existing methods that solve the OOBSP and identifies shared dilemma's. The second question considers literature from DRL and DRL applications in the domain of operations research. This obtains an overview of optimization problems being addressed with DRL, what algorithms have been used and what the academic contributions are. The third question identifies techniques that can solve multi-objective optimization problems using DRL to eventually improve applicability of DRL models.

The fourth question is answered in Chapter 3 and identifies the OOBSP for the DeepRele warehousing concept and provides a formal problem definition for this project.

In order to answer question 5, techniques and principles are presented in Chapter 4 for a DRL framework that is able to solve multi-objective optimization problems with real-world characteristics. Subsequently, Chapter 5 provides an outline of the experiment setup used to assess the performance of the DRL application, after which in Chapter 6 the results are discussed which can be used to answer question 6. These results include a policy analysis and robustness analysis.

Finally, Chapter 7 concludes findings from all of the above and provides limitations of this research together with directions for future research and recommendations to VI. An overview of this research design is presented in Figure 1.1.

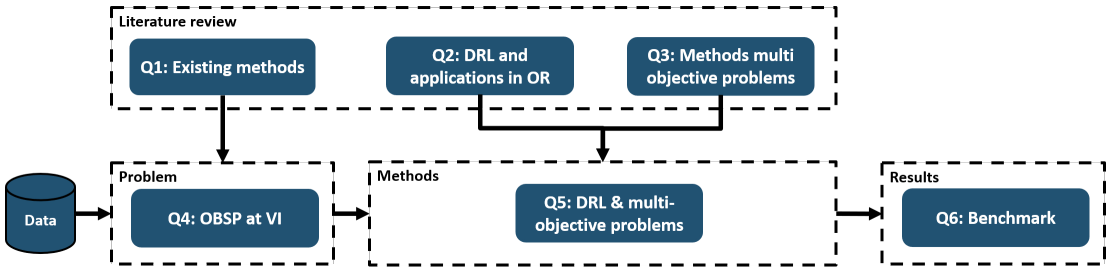


Figure 1.1: Research design and dependencies

1.5 Contributions

This work contributes to literature by providing insights on the following aspects:

- Solving the OOBSP using a DRL approach. Using the approach presented in the work of Cals et al. (2021) as a base, this has been improved to cope with real-world characteristics as a multi-objective variant of the OOBSP, handle large instance sizes and cope with an online and dynamic problem setting. In order to solve these real-world characteristics, the improved DRL approach have been complemented with a reward shaping process. This process uses a Bayesian optimization technique that assigns a priority to each objective within the reward function of the DRL approach.
- Analytic approach on the learned DRL policies with reward shaping to understand the operational impact and modeling behaviour. These insights can be used to help implement these kinds of projects by translating the learned DRL policies into more simpler heuristic rules.
- Explainable AI technique where the DRL approach has been distilled into a decision tree. From this decision tree, explainable decision rules and logic have been inferred. Understanding the learned DRL policies is an important step in the process of implementing DRL techniques in real-world scenarios.

With these contributions, this work differentiates itself and provides a motivation for further research in understanding and applying DRL policies for sequential decison making problems.

2. Literature review

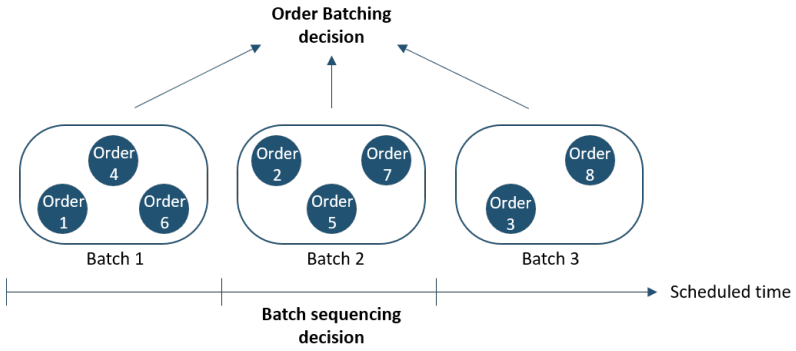
The objective of this literature review is to provide an academic motivation for this thesis project. This is done by introducing a review of current solutions methods for solving the OOBSP and by providing an overview of current (deep) reinforcement learning techniques together with applications to the operational research domain. Additionally, this literature review presents several hyper-parameter optimization techniques that could enable a better integration between the two aforementioned domains.

2.1 Solving the order batching and sequencing problem

This section is divided into two parts. The first part elaborates on the definition of the OBSP whereas the second parts discusses traditional solution methods for the OBSP.

2.1.1 Definition of the OBSP

In the literature, the OBSP is defined as grouping customer orders into picker orders (batches) and subsequently sequencing these batches to optimize a certain single or multiple objectives (Henn et al., 2012). The majority of existing work on OBSP assumes an offline (or static) setting, where all the orders are known in advance and the system provides a solution, which is assumed to be valid for the entire day. However, this thesis project considers an online setting, the OOBSP. The OOBSP is closely related to problems as the order batching and sequencing problem (OBSP), the order batching problem (OBP), the order batching, sequencing and picker routing problem (OBSPRP) and the order batching and picker routing problem (OBPRP). The routing problem tries to address finding an optimal route according to a certain objective. Figure 2.1 presents the order batching and sequencing problem in its traditional format.



	Items	Cutoff time
Order 1	A	11:00am
Order 2	B	4:00pm
Order 3	C	11:00am
Order 4	D	4:00pm
Order 5	E	11:00am
Order 6	A	4:00pm
Order 7	B	11:00am
Order 8	C	4:00pm

Figure 2.1: The combination of the order batching and sequencing operations (Chen et al., 2015)

All OBP and related variants mainly focus on the optimization of three objectives: minimizing order tardiness, minimizing travel distance and minimizing processing times. The OOBSP as described above is known to be $\mathcal{NP}$-hard in strong sense if the number of orders per batch is larger than two and is found as solvable in polynomial time (Gademann and Velde, 2005). The majority of existing work on OOBSP assumes an offline (or static) setting, where all the orders are known in advance and the system provides a solution for a restricted period of time. In the coming subsections, an overview is given on both online as offline solution methods and this provides context on the complexity, and guidance in solving the OOBSP.

2.1.2 Exact solution methods

Most of the exact solutions for the OBP available in present literature rely on applications with small instances. This can be derived from results in Table 2.1. Besides this, all exact solution approaches that have been discussed have the single objective to minimize total processing time and compute a solution in an offline setting.

Armstrong et al. (1979) proposes a mixed integer linear programming approach for the OBP of a semi-automated liquor distribution centre case. An experimental study is performed with a 30 order / 12 aisle example. The authors found optimality in their solution and outperformed the most recent solution at the time: the benders approach.

Van Den Berg et al. (2001) defines two criteria for order batching: firstly, proximity batching minimizes the travel time and thereby increases throughput and secondly, time window batching optimizes due date performance. For this project, the OBP is evaluated as proximity batching. The authors present an exact branch-and-bound algorithm that uses a 2-opt heuristic to compute the initial upper bound of the solution.

Gademann and Velde (2005) conducted an extensive numerical study on their exact solution and found that their algorithm is able to provide optimal solutions for smaller problem instances. The authors devised a branch-and-price algorithm that can find the optimum solution of test instances with up to 32 customer orders. However, the computing time increases significantly and the guarantee to solve the instance optimally disappears when increasing the number of orders.

Bozer and Kile (2008) proposes a mixed integer programming model to obtain near-exact solutions to the problem. This model provides good lower and upper bounds to the problem, which prove to be useful for heuristics to solve the actual problem. The objective is to minimize the total time to pick all orders in a 'wave', given finite-capacity order pickers. A wave is defined as a pre-defined period of time where order pickers simultaneously pick batches. A batch is related to only one specific order picker. This model provides near-exact solutions because the mixed integer linear program contains a heuristic. Unfortunately, the best batching heuristic requires substantial computational time for large waves.

In the work of Muter and Öncan (2015), the OBP is solved using a column generation method. The authors considered different routing strategies in the experiments, and use 90 data sets which include up to 100 orders. These results were obtained from a case study with low-level picker-to-parts picking systems. The column generation method is a well-known algorithm for large-scale LP problems. In this case, it is used to solve the LP relaxation of mixed integer linear programming with exponentially many variables.

Valle et al. (2017) considers the case of online grocery shopping, where orders may be composed of dozens of items. The authors frame the problem as a order batching and picker routing Problem where the objective is to find minimum-cost closed walks where each picker visits all locations required to pick all products from their assigned orders. An integer programming method is proposed and the authors show that given an assignment of orders to pickers, an optimal routing can be computed very quickly. Instances involving up to 20 orders were solved to proven optimality. Instances involving up to 5000 orders were considered where order batching was done heuristically, but picker routing done optimally.

2.1.3 Heuristic solution methods

Existing exact solution approaches to the OBP have only a limited applicability due to $\mathcal{NP}$-hard property, real-world problems have to be solved by means of heuristic approaches (Henn et al., 2012). A variety of heuristic approaches has been suggested for the OOBSP. These approaches can be partitioned into priority rule-based algorithms, seed algorithms, savings algorithms, and meta-

Work	System	Algorithm	Order instance	Items / order	Objective
Armstrong et al. (1979)	PtG	MIP	30 orders	200 - 6000	Min. processing time
Van Den Berg et al. (2001)	PtG	Branch-and-bound	30 orders	4 - 20	Min. tardiness
Gademann and Velde (2005)	PtG	Branch-and-price	30 orders	6 - 20	Min. travel time
Bozer and Kile (2008)	PtG	MIP	25 orders	2 - 7	Min. picking costs
Muter and Öncan (2015)	PtG	Column generation	100 orders	2 - 10	Min. travel time
Valle et al. (2017)	PtG	IP	20 orders	4 - 5	Min. picking costs

Table 2.1: Discussed work with research objective and experiment instance sizes

heuristics (Henn et al., 2012). The first three groups contain approaches which are constructive by nature, whereas meta-heuristics focus on improving given solutions (De Koster et al., 2007).

Constructive approaches
Elsayed (1991) uses due date information for orders when solving the OOBSP in an online setting. Orders arrive at the system in a static fashion and are classified as storage-orders (s orders) and retrieval-orders (r orders) and processing is performed in three different kind of cycles: (i) storage-only cycle (the SIR machine can perform storage functions only), (ii) retrieval-only cycle (the SIR machine can perform retrieval functions only), and (iii) interleaving cycle (the SIR machine can store and retrieve items in the same cycle). In this study, an Automated Storage and Retrieval system is used which is similar to the Goods-to-person system of DeepRele. Elsayed (1991) presented three rules in his work:

- Sequencing s-orders and r-orders using Maximum Longest Travel Time rule and Earliest Due Data rule
- Batching r-orders using the Nearest Schedule rule, the Shortest Processing Time rule and the Most Common Location rule.
- Nearest Location rule of including s-orders in picking tour

The above rules are classified as seed and savings algorithms and are evaluated using a simulation model. Different combination of rules have been considered but eventually, this work concluded that for batching s-orders the Nearest schedule rule provides superior results in terms of shortest travel time. Furthermore, the Earliest Due Date rule is used for sequencing and the Nearest Location rule has been used for including s-orders in a pick tour.

In subsequent work, Elsayed et al. (1993) present an online setting of the OOBSP that is solved with a dual objective: minimizing the tardiness of orders and minimizing the earliness of orders. The OOBSP investigates the performance of order retrievals in AS/RS operating under a Just-In-Time control policy where every order has a due date requirement. A priority rule-based algorithm is used that contains the following steps: (1) sequencing the orders, (2) forming order batches and (3) determining the optimal release time of batches for retrieval. Algorithms have been assigned to each individual step in order to optimize the objectives. The sequencing step (1) consists of an priority index for each order which is a weighted sum of the order tardiness and the order earliness. Based on this sequence, orders are batched into groups while considering capacity of the system. This iterative process creates seeds of first orders and continues to add orders to the formed batch until the system's is violated or the value of the objective function increase. In case of the two latter scenarios, the specific order is skipped. This iterative process continues until either no more orders can be added to the current batch due to capacity constraints or no more remaining orders need to be assigned. This yields the first batch and subsequently, the next order is selected as seed and the process iterates thru the remaining orders, creating batches until all orders have been assigned to a batch. Determining the optimal release time for batches (3) is done by inserting idle

time between the determined batches and to calculate the change in the objective function when a batch is delayed by one unit of time and include if this provides a more optimal solution.

Lee and Kim (1995) published work as an extension on the work of Elsayed et al. (1993) where they focus more on obtaining an optimal schedule under dual commands. They assume that the number of s-orders are almost equal to the number of r-orders and consider a manufacturing environment where a unit-load AS/RS is used for storing work-in-progress and tools from a production line. Furthermore, the papers acknowledges the fact that the proposed solution need to be able to solve large problems. In order to do this, Lee and Kim (1995) proposed 4 types of batching heuristics all according to a 2-step approach.

1. S and r-orders are batched using one of 4 batching heuristics:
    - Minimum Total Processing time (MTP) rule
    - Shortest Processing Time (SPT) rule
    - Shortest Processing Time Nearest Location (SPT-NL) rule
    - Longest Processing Time (LPT) rule
2. During several dual-command cycles, the batched orders are sequenced to minimize total earliness and total tardiness using:
    - The first algorithm sequences formed batches in a non-increasing manner based on processing times and selects the first batch. If the the processing time of the current batch is smaller than the processing time of the next batch in sequence, that specific batch is added to set A: set of batches that are finished before coming due date. If this is not the case, the batch is added to set B in SPT order: set of batches that are finished after coming due date (Bagchi et al., 1987).

Several experiments with different combinations of the above heuristics were computed. For smaller instance sizes, the combinations of heuristics where compared with a mixed integer linear programming model that can generate optimal solutions. For larger instance sizes, the mixed integer linear programming model could not be used, performance among individual heuristics is compared. MTP and SPT outperform the other heuristics in both scenarios. In the small instance size scenario, SPT is superior but if the instance size increases, MTP appears to be superior.

Elsayed and Lee (1996) proposes a solution approach for the generation and sequencing of batches when only the total tardiness has to be minimized. Firstly, the customer order are sequenced according to their due dates and processing times when orders would be processed as a single-order batch. A first bound on the optimal total tardiness can be obtained by assuming that the items of each customer order would be collected on a single tour and sequencing the tours according to the order's position in the sequence. Three decision rules for the generation of batches are compiled and evaluated. (1) The Nearest Schedule Rule assigns the first customer order as a seed to the initial batch. Subsequent customer orders are assigned to the similar batch if the inclusion does not increase the total tardiness and does not violate capacity restrictions. If this is the case, the last remaining customer order in the sequence is used as seed order for the next batch. (2) The Shortest Service Time rule adds orders to the seed which would be possess the shortest processing time, if the order was processed in a separate tour. (3) The Most Common Location Rule, an Order is added to the seed order which has the largest number of pick locations in common with the seed order. Conducting an experiment with these methods yields that the Nearest Schedule rule outperforms the other rules and achieves results which are close to optimality. A MIP model is proposed to compute optimal results and to compare performance of defined heuristics.

Won and Olafsson (2005) presents an integrated solution for the order batching and sequencing problem. The authors propose two heuristics for this problem which minimize the weighted average of the total picking time of batches and the holding time of orders in the batch. The first heuristic

is a sequential decision mechanism, sequential order batching and picking algorithm (SBP), that at first solves the batching problem and subsequently, addresses the sequencing problem. The basic idea is to minimize the processing time, combining the order picking time that represents the warehouse efficiency and the order holding time that represents the responsiveness to customers or supply chain partners. The second method, joint order batching and picking problem (JBP), has similar goals but attempts to achieve them in a parallel manner by simultaneously constructing batches and sequences. Besides this, this heuristic relaxes a constraint made by SBP, namely that each picking vehicle only leaves when full capacity is reached. The numerical results demonstrate considerable benefits when combining both heuristics.

In the work of Zhang et al. (2017), a hybrid rule-based algorithm, referred to as FTWB, is proposed in order to address the OOBSP in an online scenario. The hybrid rule-based algorithm is proposed in order to form batches and assign them to appropriate pickers without any information on the arrival times of future orders. The algorithm identifies three scenarios based on quantity of the items: off-peak (A), normal(B) and peak(C). Based on these scenarios, different rules are applied. The algorithm is divided into two parts. In the first part, orders come in at a certain time and a scenario is selected. Based on this scenario and the related batching rule, orders are batched. In the off-peak scenario, no batching is allowed as there is ample picking capacity. In the normal scenario, incoming orders are batched until some capacity violation. Lastly, the peak scenario requires a set of batching rules in order to perform the batching operation: a seed algorithm. The cumulative-seed algorithm is used that starts off with an empty batch. A seed order is selected from the order pool using the Smallest Arrive Time rule (SAT) and is added to the batch and removed from the order pool. Subsequently, the aisle-time-based rule (ATB) is used to select orders from an set O that all can be included to the batch without violating capacity constraints. This process is an iterative process and stops until no more orders can be added to the specific batch. As the batches have been formed, these are transferred to the batch operating system where these are assigned to a picker using a certain assignment rule: list processing algorithm. This algorithm minimizes the turnover time and balances the workload among the order pickers. The proposed model is compared to a individual time-based batching rules and the quantity-based batching rules. The extensive numerical experiments in comparing Hybrid with the Time and Quantity algorithm, led to the following finding: Hybrid provides better performance than the Time and Quantity algorithm.

Meta-heuristics

In the work of Tsai et al. (2008), an OOBSP is considered where the total travel costs (depending on the total travel time) have to be minimized and earliness and tardiness are penalized. Unlike approaches that were previously discussed, this project allows orders to be split up. In all other approaches, orders could be included in a batch but it was not possible to split up an existing order and divide parts of the specific order to different batches. The problem is solved using a two genetic algorithms (GE) where a solution is represented as a set of integers. A GE is an evolutionary algorithm that is based on natural selection, the process that drives biological evolution. The set of integers is considered as an initial solution to the OOBSP and is referred to as a 'chromosome' and can be randomly generated. The first GE solves the order batching problem by minimizing the processing times together with the earliness and tardiness penalties. The second GE searches for an optimal travel within a batch by minimizing the travel distance. From the comparison they have made, their solution outperforms two other state-of-the-art methods on small instance size (1 picker, 1 vehicle). Furthermore, Tsai et al. (2008) mention that GEs perform less than MIP/MILP formulation for small problem instances but when increasing the instance size, the GE outperforms these exact methods. It appears that Genetic Algorithms provide a good approach in solving real-world OOBSP because of the allowed instance size.

In the work of Henn and Schmid (2013), the authors propose an Iterated Local Search algorithm (ILS) and the Attribute-Based Hill Climber algorithm (ABHC) meta-heuristics for the OOBSP. These meta-heuristics have been designed to minimize the total tardiness of all customer orders. The ILS has been successfully implemented to a variety of different optimization problems, including the OBP of Henn et al. (2010). The initial definition of the ILS algorithm is presented next. Within a set of feasible solutions, every individual solution from this set has a related sub set of neighboring solutions. The heuristic consists of two alternating phases: a local search phase and a perturbation phase. In the local search space, the heuristic starts from an initial solution and determines a sequence of feasible solutions where each element is a neighbor of its predecessor and is an improvement of the objective function compared to its predecessor. This provides a local optimum. During the perturbation phase, the incumbent solution is partially modified (perturbed) and a further local search phase is applied to this modified solution. In order to adapt the algorithm to fit the OOBSP, several principles have been modified. The initial solution set is generated using the Earliest Due Date (EDD) rule wherein orders are sorted according to their due dates in a non-increasing order and orders are batched using a simple heuristic. In the local search phase, two orders from different batches are interchanged (swap move) or by assigning one order to a different batch (shift move). The iterative process start with swap moves, if no improvements can be made according to the objective function, ILS performs a shift move that leads to an improved solution. These steps are repeated until no further improvements can be found. Subsequently in the perturbation phase, 50% of the orders of a certain batch are randomly selected and exchanged with 50% of the orders of batch 1. Should this violate any capacity constraints, then all remaining orders will be assigned to a new batch. If the objective function decreases, the specific solution will be accepted as incumbent solution. Next to this, the ABHC is an almost parameter-free heuristic based on a simple tabular search principle. Henn and Wäscher (2012) have applied ABHC to the OBP and obtained solutions with improved tour lengths compared to other local search-based heuristics. For each problem, the tabu search proceeds the search in solutions until every path has been investigated. In this local search, a solution can be accepted if and only if it represents the best solution found so far and memorizes the visited solutions so far. This guarantees that every move made is measurably improving the objective function and that the search can never return to a solution it has visited previously. Henn and Schmid (2013) adapt this meta-heuristic to fit the OOBSP by creating the initial solution using the Earliest Due Date rule. New solutions can be obtained by conducting swap or shift moves. Regarding the computation structure, two sets are created: a pair of orders that are assigned to a batch and the actual assignment of pairs of orders to a specific batch. Henn and Schmid (2013) indicated that the problem representations grow exponentially with the number of orders. For a small-sized problem instance with 20 orders and in which five orders can be assigned to a single batch, the number of feasible batches is 21,699. This concludes that the use of heuristics for larger problem instances becomes unavoidable. In an 80 order experiment, Henn and Schmid (2013) compare the presented heuristics with only an EDD approach. They present an average improvement of 46% on different problems and both heuristics provide short computation times.

Henn (2015) studies the OOBSP and proposes a Variable Neighborhood descent (VND) and a Variable Neighborhood Search (VNS) to minimize the total tardiness of customer orders. VND is a deterministic local search algorithm that starts with an incumbent solution and iteratively goes thru sequences of multiple neighborhood structures. In each iteration the algorithm identifies the element of the current neighborhood structure with the (local) optimal objective function value. If a solution with an improved objective function value can be identified, this solution becomes the next incumbent solution and the algorithm continues to explore the next neighborhood structure. In addition to the VND, the VNS explores different neighborhoods in a stochastic and deterministic way (Hansen and Mladenović, 2001). It is composed of two phases, a stochastic shaking and a deterministic local search phase. VNS starts with an initial solution that is shuffled using a stochastic method and is improved by a local search phase. The solution stemming from the local search phase becomes the incumbent solution. The results conclude that VND outperforms VNS especially by means of computation times.

Li et al. (2017) have studied a method for joint optimization of order batching and picker routing in the online retailer's warehouse in China. The authors propose an online order batching strategy that uses a similarity coefficient that represents the similarity between two orders. The main idea is to combine similar orders as one in order to obtain a high order picking efficiency. Furthermore, the authors use a s-shaped routing heuristic to compute the order picking route. A numerical experiment based on actual sales data is given to test the efficiency of heuristics and results indicate that the proposed heuristic can shorten the order picking distance by optimising order batching and picker routing, generating orders into less batches which increases the order picking efficiency. These results were obtained based on experiments with a maximum of 10,000 orders. In practice, problem size could be very large and is still increasing. Online retailing in China is growing at a high speed and competition is becoming more furious.

Scholz et al. (2017) propose an integrated approach that addresses three sub problems: (1) how the customer orders are grouped into picking orders (order batching problem), (2) how the picking orders are assigned to and sequenced by the order pickers (Batch Assignment and Sequencing Problem), and (3) how each order picker is routed in order to collect the items of each picking order (Picker Routing Problem). These sub problems all try to minimize the total tardiness. For small instances, the authors use Mixed Integer Programming and for large instances, the authors propose a variant on the variable neighborhood descent algorithm. Compared to a standard VND algorithm, the authors adjusted their algorithm whereby after finding an improvement, the algorithm determines directly a local optimum regarding the current neighborhood structure. For experiments, the work of Henn (2015) is used as base. Compared to solutions obtained by applying the earliest start date rule-based algorithm that was also used by Henn (2015), it is demonstrated that the initial solution leads to a reduction of the total tardiness by up to 63%.

Bustillo et al. (2015) present a General Variable Neighborhood Search (GVNS) that performs an online batching operation and include a comparison with the work of Henn and Schmid (2013). A case is presented where a single order picker addresses 4 different order scenario's: 20, 40, 60, 80. The number of items per order is uniformly distributed between 5 and 25. The GNVS of Bustillo et al. (2015) outperforms the proposed ILS and ABHC methods of Henn and Schmid (2013) based on minimizing total tardiness. Furthermore, the GVNS is computationally more efficient than the ILS method and most computation times are below 10 seconds. These computation times are however based on a maximum instance size of 80 orders. Lastly, the authors state that in order to implement an integrated method, their work should be complemented by a routing strategy to boost performance.

In the work of Ardjmand et al. (2018), order assignment, order batching and picker routing problems with multiple pickers in a wave picking warehouse of a major US third party logistics company is studied and modeled mathematically. Since the exact algorithm suffers from long CPU time when solving large instance problems, a hybrid parallel simulated annealing and ant colony optimization (PSA-ACO) is presented. Simulated annealing is a metaheuristic that starts from a random solution x, and in next subsequent steps, neighbor solutions y are explored. If y is better in terms of solution quality, it will be accepted as the new solution, otherwise it will be accepted based on a acceptance probability. ACO mimics collective behavior of ants in nature where the objective is to find the shortest path from their nest to the food source. In the proposed algorithm, this metaheuristic is used for picker routing. In an experiment, the results of the proposed algorithm are compared against the minimum makespan impact (MMI) heuristic that is currently being used in the warehouse and a state-of-the-art variable neighborhood descent (VND). While PSA-ACO slightly outperforms VND, for picking large waves, PSA-ACO and VND can improve the makespan by approximately 7.8% and 6.9% over MMI respectively. This experiment consisted out of 1000 orders.

Pinto and Nagano (2019) present a set of two genetic algorithms as solution to the OOBSP that better adjusts the trade-of between the level of customer service and the effectiveness of a warehouse. This set of genetic algorithms consists of GE_{Batch} and GE_{TSP} that each have a

different purpose. GE_{Batch} groups orders in multiple batches so that, by complying with capacity constraints and the EDD rule, earliness and tardiness in order fulfillment is avoided and the number of picking travels inside the warehouse is minimized. GE_{TSP} maps out the sequence of orders within a batch so that the routing distance and picking time can be reduced. Experiments with problems of different complexity levels showed that the proposed method produces solutions of satisfactory quality to any problem instance. In those experiments, the authors considered up to 50 orders out of more than 600 SKUs in total. Also, the authors performed a sensitivity analysis on the parameters of the Genetic Algorithms: population size, generations, crossover probability, mutation probability and mutation rate of bits. To conclude: "The proposed solution fills a gap in the literature and makes innovative contributions to advances towards developing picking optimization methods which are more suitable to the reality of the mentioned warehousing operation" (Pinto and Nagano, 2019).

Work	System	Algorithm	Order instance	Items / order	Objective
Elsayed (1991)	GtP	H: Priority and Seed Rule	8 - 12	35 - 65	Min. tardiness
Elsayed et al. (1993)	GtP	H: 3-step priority index algorithm	20	35 - 65	Min. tardiness
Lee and Kim (1995)	GtP	H: MTP, SPT, SPT-NL, LPT rules	3 - 50	N.A.	Min. tardiness
Elsayed and Lee (1996)	GtP	H: Priority and Seed Rule	8 - 12	35 - 65	Min. tardiness
Won and Olafsson (2005)	PtG	H: SBP and JBP	30	1 - 5	Min. travel time
Zhang et al. (2017)	PtG	H: Hybrid rule based algorithm	120 - 480	1 - 5	Min. picking costs
Tsai et al. (2008)	PtG	M: Genetic Algorithm	24 - 250	30 - 400	Min. tardiness & travel time
Henn and Schmid (2013)	PtG	M: ILS and ABHC	20 - 60	5 - 25	Min. tardiness
Henn (2015)	PtG	M: VNS and VND	100 - 200	5 - 25	Min. tardiness
Li et al. (2017)	PtG	M: BOC	100 - 10,000	1 - 5	Min. Travel time
Scholz et al. (2017)	PtG	M: VND	100 - 200	5 - 25	Min. tardiness & Min. travel time
Bustillo et al. (2015)	PtG	M: GVNS	20 - 80	5 - 25	Min. tardiness
Ardjmand et al. (2018)	PtG	M: Simulated Annealing and PSO	1000	N.A.	Min. picking costs
Pinto and Nagano (2019)	PtG	M: Double Genetic algorithm	10 - 50	12	Min. picking cost & processing time
van Gils et al. (2019)	PtG	M: ILS	200	4	Min. tardiness

Table 2.2: Discussed work. (H: Heuristic, M: Meta-heuristic)

The work of van Gils et al. (2019) provides a decision support tool for spare part warehousing operations that also integrates and solves the three main operational order picking planning problems (order batching, picker routing, and picker scheduling). Different from other studies, the objective is to increase order picking efficiency while ensuring a high customer service level. The proposed tool accounts for order due times and limited availability of order picker. The work of van Gils et al. (2019) provides a mathematical formulation for the problem and a metaheuristic algorithm. This meta-heuristic algorithm is based on an iterated local search algorithm (ILS), and is proposed to approximate the global optimal solution. The main components of ILS include a procedure to generate an initial solution, a local search procedure, and a perturbation procedure just as in the work of Henn et al. (2010). A series of numerical experiments are used to assess the performance of the proposed ILS model. During a planning period of 4 hours, 6-8 order pickers pick 200 orders with up to 4 order lines. The proposed ILS model outperforms the current warehouse operations of the case study by reducing the required number of pickers by 25%.

2.1.4 Deficiencies of traditional solution methods

To conclude the discussed work, an overview is presented in Table 2.1 and 2.2 where all solution approaches have been presented. Also the instance sizes of the related experiments have been displayed. Both in problem definition as in the proposed algorithms, a lot of differences can be found among the presented work. The type of storage system that has been included in the problem formulation which either is defined by a Person-to-Goods system or a Goods-to-Person system which has consequences in the design of the proposed algorithms. In terms of system constraints, almost every study considered constraint on picking capacity. A disadvantage of the exact methods is that these cannot cope with uncertainty and assume that all information is known in advance (offline approaches). Furthermore, it stands out that most algorithms solve relatively small order instances and single objective problems. Overall, the literature presents a variety of algorithms to address different variants of the OOBSP. However, an unified design of picking operations is not used.

2.2 Deep reinforcement learning

Reinforcement learning (RL) is the study on how an agent can interact with an environment to learn a policy which maximizes expected cumulative rewards for a task. Recently, RL has experienced major growth in attention and interest due to promising results in areas like: controlling continuous systems in robotics (Lillicrap et al., 2015), playing Go (Silver et al., 2016), Atari (Mnih et al., 2013), and competitive video games (Vinyals et al., 2017; Silva and Chaimowicz, 2017). The work of Mnih et al. (2015) combined reinforcement learning and deep learning and caused a breakthrough in the field: deep reinforcement learning (DRL) was born which demonstrates impressive results for sequential decision making problems. This chapter provides an introduction to RL by a literature review. Secondly, DRL is presented as a continuation of RL and several applications for optimization problems are discussed. Moreover, a conclusion at the end wr In the RL-setup, an *agent*, observes a *state* S_t provided by the *environment* at time step t. The agent interacts with the environment by taking an *action* A_t. After taking this action, the environment transitions to a new state S_{t+1}. Based on the change in states, the agent receives feedback in the form of a *reward* R_t and tries to optimize this reward over time. By choosing actions, the environment moves from state to state and each interaction yields information which the agent uses to update its knowledge. In some cases, this process is iteaps up this section.

2.2.1 Introduction to reinforcement learning

ratively repeated until the environment reaches a *terminal state* and this time window is referred to as an *episode*. For example, a single game play in an Atari game is an episode. After this terminal state, the environment resets and the interaction can start over again independently of the outcome of the previous episode. On the other hand, in many cases the agent–environment interaction does not break naturally into identifiable episodes, but goes on continually without limit. In this case, the additional concept required is that of discounting. According to this approach, the agent tries to select actions so that the sum of the discounted rewards it receives over the future is maximized. In this case, the iterative process is terminated based on some derived termination criteria. According to the work of Sutton et al. (1998), the performance of RL is heavily influenced by the way the problem is represented in terms of states, actions and rewards. For example, the state representation of the system should contain sufficient statistic information of the environment and thereby comprises all the necessary information for the agent to take the best action. RL algorithms are general in their working, representation of the problem is different for each problem. The entire process of RL is displayed in Figure 2.2.

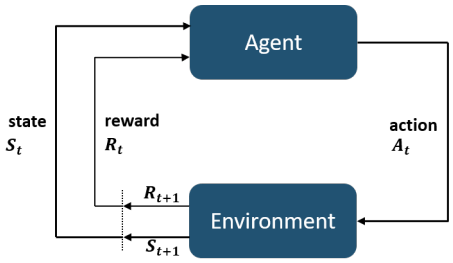


Figure 2.2: The reinforcement learning framework (Sutton et al., 1998)

In order to solve RL problems, the problems are framed as Markov Decision Processes (MDP). MDPs are a mathematically idealized form of the reinforcement learning problem for which precise theoretical statements can be made. These theoretical statements have been incorporated in RL models that try to solve MDPs near optimally. An MDP is defined by a tuple $< S, A, R, T, \gamma >$, that is explained in more detail below (Sigaud and Buffet, 2013).

- Set of states S
- Set of actions A
- Reward function $R(s, a)$ which provides an immediate reward after a transition from s to s' with a certain action a.
- Transition probability function $P(s'|s, a)$ that provides transition probabilities that when in given state s, the action takes action a, the system ends up in state s'.
- Discount factor $\gamma \in [0, 1]$, where lower values place more emphasis on immediate rewards.

A MDP provides a stochastic framework that assumes the Markov property. This property states that the probability distribution of future states only depends on the current state and does not depend on the past. As the transition function of a MDP is generally unknown, it remains a problem to estimate a policy from the MDP that provides us with optimal actions. In order to estimate a policy π, which is a stochastic rule by which the agent selects actions as a function of states, the agent needs to learn the dynamics of the model by interacting with it. If the set of all possible states is assumed to be finite and the set of all possible actions as well, a *finite MDP* need to be solved.

2.2.2 Neural network as function approximators for DRL

Up until this point, this chapter focuses on RL methods that can exhaustively explore all states and obtain accurate value functions due to interaction with the environment. However, in many tasks to which reinforcement learning can be applied, the state space is combinatorial and enormous; the number of possible camera images, for example, is much larger than the number of atoms in the universe. In such cases, it is not expected to find an optimal policy or an optimal value function even in the limit of infinite time and data. Instead, a good approximate solution using limited computational resources should be applied. Memory requirements for large state spaces are not the problem on its own, but rather the computation time to fill this memory. In many target tasks, almost every state encountered will never have been seen before and to make sensible decisions, it is necessary to generalize from previous encounters with different states that are somewhat similar. Fortunately, generalization from examples has already been extensively studied and for instance, one could try to estimate Q-values with $\hat{q}(s, a, w) \approx q_\pi(s, a)$ in which the parameters w can be updated based on experiences of an agent. This generalization technique is often referred

to as function approximation and is an instance of supervised learning, a primary topic in machine learning (Sutton et al., 1998). A neural network is considered as a function approximator for deep reinforcement learning and the following part elaborates on this concept.

A neuron is the basic unit of deep neural networks and is formalized with numerical inputs and outputs. Within a deep neural network, a neuron can have many output neurons in the next layer and a neuron can also have many input neurons in the previous layer. A neuron in one layer aggregates the signals being passed through from its input neurons in the previous layer. This aggregated signal is then passed through an activation function that determines the neuronal behavior. Concretely, if the aggregated signal is strong enough, then the activation function "activates" this neuron and pass forward a high value to the output neurons in the next layer (Dong et al., 2020). The left side of Figure 2.3 displays a signal moving through a single perceptron and a bias is included. On the other side, this principle is displayed in a network with multiple layers (bias operators are not shown in right figure). The activation is typically non-linear, e.g. a Sigmoid function (Sutton et al., 1998). During training of the network, a loss function is defined between desired output and computed output. This loss function is used as objective to minimize by adjusting weight parameters of the neural network. A gradient descent method is used to update these parameters using partial derivatives, and back propagation is used to compute these partial derivatives in the network. As a network is trained iteratively on training data, new unseen testing data can be provided in order to obtain the network's accuracy.

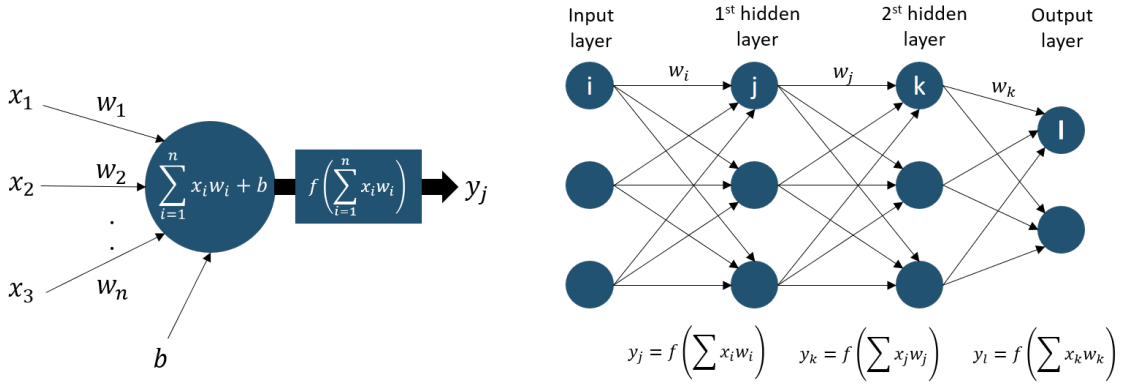


Figure 2.3: Principles of a perceptron and neural network

In combining Deep Neural Networks (e.g. neural networks with multiple hidden layers), with RL, the input layer of the network is shaped as the state representation of the RL problem. With this input, the network tries to predict which action should be performed. Similarly for some DRL methods, the output layers of the network represent the action space of the RL problem. Each output perceptron represents an action in the action space and the network provides a probability for each individual action to execute. Rewards from the environment are used in a loss function to update the network. Similarly to plain reinforcement learning methods, a distinction can be made between methods that use value function, policy iteration or a combined method: actor-critic. Next sub section elaborates further on value function based and actor-critec based methods.

2.2.3 Deep reinforcement learning models

This value-function based algorithm further elaborates on Q-learning principles. Mnih et al. (2015) published a breakthrough paper where a neural network is used to approximate the value function of a Q-learning method, and Deep Q Networks (DQN) were born. In this work, a deep convolutional neural network is used and demonstrated outstanding performance on a variety of Atari games. According to Mnih et al. (2015), approximating the value function with a deep neural

network results in unstable behaviour or even divergence. This instability has several causes: the correlations present in the sequence of observations, the fact that small updates to Q-values may significantly change the policy and therefore change the data distribution, and the correlations between the action-values (Q) and the target values (Mnih et al., 2015). The instable behaviour is counteracted by introducing experience replay, that randomizes a set of experiences whereby it removes correlations. Secondly, an iterative update rule is used that adjusts the action-values (Q) towards target values that are only periodically updated, thereby reducing correlations with the target. Since the DQN was implemented, several additions have been implemented. (1) Double DQN (using two networks to avoid getting overoptimistic estimates Van Hasselt et al. (2015)). (2) Prioritised replay (prioritized experience replay method that samples from a buffer of experiences in a different way (Schaul et al., 2015)). (3) Duelling network (implementing dual output so that there are separate estimates of the value and advantage functions (Wang et al., 2016b)). (4) Distributed agents (These variants could be run on many instances simultaneously, allowing agents to acquire and learn from experience more quickly (Nair et al., 2015; Horgan et al., 2018)). Furthermore, Badia et al. (2020) presents additional improvements to this DQN network that enables it solve all Atari games with superhuman performance.

The PPO algorithm presented by Schulman et al. (2017) is an extension of the Trust Region Policy Optimization (TRPO) algorithm proposed by Schulman et al. (2015) and the Actor Critic with experience replay (ACER) presented by Wang et al. (2016a). In comparison to offline learning methods as DQN, PPO is an online learning method that learns directly from experiences that have been encountered from the agent. The policy gradient loss is estimated and subsequently used to update a neural network according to some update rule. This policy gradient loss is presented in Equation 2.1.

$$L^{PG}(\theta) = \hat{\mathbb{E}}_t \Big[\log \pi_\theta \left(a_t | s_t\right) \hat{A}_t \Big] \tag{2.1}$$

The expectation indicates the average over a finite batch of samples, π_θ refers to a stochastic policy and $\hat{A}_t$ is an estimator of the advantage function at time step t that represents the relative value of action a_t at state s_t. This estimation of the advantage function can be calculated with the sum of all rewards encountered by an agent within one episode multiplied with some discount factor. The value function is subtracted from the discounted reward in order to compute the advantage function. This answers the question: was the specific action that the agent took better or worse than the value function expected? According to Equation 2.1, if the advantage function is positive, the gradient becomes positive and the probabilities of the specific action are increased. Likewise, if the advantage function is negative, the gradient becomes negative and the probabilities of the specific action are decreased. A posing problem with PPO and all other gradient descent methods, if the gradient descent method is applied on a single batch of experiences, the parameters in the neural network are updated outside the range from where this data was collected. This problem is addressed with Trust Region Policy Optimization (TRPO) (Schulman et al., 2015). The objective function of TRPO maximizes an objective function subjected to a constraint on the size of the policy update: KL constraint, and is displayed below in Equations 2.2 and 2.3.

$$\max_{\theta} \hat{\mathbb{E}}_t \left[\frac{\pi_\theta \left(a_t|s_t\right)}{\pi_{\theta old} \left(a_t|s_t\right)} \hat{A}_t \right] \tag{2.2}$$

$$\text{subject to } \hat{\mathbb{E}}_t[KL[\pi_{\theta old} \ (\cdot|s_t), \pi_\theta(\cdot|s_t)]] \ \leq \ \delta \tag{2.3}$$

Here, θ_{old} are the parameters of the neural network of the old policy and the KL constraint makes sure that the updated policy does not move too far from the original policy. As this constraint poses additional computation, PPO incorporates this additional computation as a clipping operation in the objective function and is displayed in Equation 2.4.

$$L^{CLIP}(\theta) = \hat{\mathbb{E}}_t \Big[\min(r_t(\theta)\hat{A}_t, \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t \Big] \tag{2.4}$$

in which $r_t(\theta) = \frac{\pi_\theta \left(a_t|s_t\right)}{\pi_{\theta old} \left(a_t|s_t\right)}$. Furthermore, $\hat{\mathbb{E}}_t$ represents an expected value and this indicates that the objective is calculated over a batch of experiences. Within the expectation operator, a minimum operator selects either the default objective or the clipped version of normal policy gradients. The default objective multiplies $r_t(\theta$ with $\hat{A}_t$ and pushes the policy towards actions that yield a high positive advantage towards the baseline. The clipped version is similar to the default objective but it contains a truncated ratio of the $r_t(\theta)$ ratio by applying a clipping operation between $1-\epsilon$ and $1+\epsilon$. Finally, the minimum of the unclipped and clipped is taken so the final objective is a lower bound on the unclipped objective Schulman et al. (2017). The final loss function that is used when training the agent is presented in equation 2.5.

$$L_t^{PPO}(\theta) \ = \ \hat{\mathbb{E}}_t \Big[L_t^{CLIP}(\theta) \ - \ c_1 L_t^{VF}(\theta) \ + \ c_2 S[\pi_\theta](s_t) \Big] \tag{2.5}$$

Equation 2.5 is the sum of the clipped PPO objective plus two additional terms. The first additional term ensures that the baseline network is updated. The baseline network is in charge of estimating the average amount of discounted reward that is expected from the current state onward. The last term is referred to as the entropy term and this term is in charge of making sure that the agent does enough exploration during training. This entropy term behaves randomly due to its nature but when the other terms start dominating, this effect will be diminished. Lastly, c_1 and c_2 are hyper parameters that define the contributions of the last two loss terms in the loss function. Running the agent and updating the policy network can be decoupled by each other by using thousands of remote agent that interact with the environment using a recent copy of the policy network. In parallel, a GPU cluster runs gradient descent on the network weights using the collected experience from those agents. This exponentially speeds up the entire process.

In the work of Wang et al. (2020), the authors state that PPO still suffers from the risk of performance instability and to cope with this, an enhanced method is presented named Truly PPO. The authors present two critical improvements: 1) Truly PPO adopts a new clipping function to support a rollback behavior to restrict the difference between the new policy and the old one; 2) the triggering condition for clipping is replaced with a trust region-based one, such that optimizing the resulted surrogate objective function provides improvement of the ultimate policy performance. This improves the original algorithm in both sample efficiency and performance.

Besides the DQN and the PPO algorithm, a broader literature review has been conducted which included the following algorithms: Q-learning by Watkins and Dayan (1992), SARSA by Rummery and Niranjan (1994), the REINFORCE algorithm by Arulkumaran et al. (2017) and the Asynchronous Actor Critic model (A3C) by Mnih et al. (2016).

2.2.4 Application of DRL in the Operational Research domain

This section provides an overview of optimization problems that have been addressed with deep reinforcement learning. More specifically based on the OOBSP, specific scheduling problems have been selected to scope the literature review on this topic. In this subsection, several associated papers are discussed and an overview is provided of the instance-sizes of all problems in Table 2.3 to provide motivation for this research project.

Manufacturing systems
The work of Waschneck et al. (2018) focused on the job shop scheduling problem. The objective is to optimize system's throughput taking real time information into account. Their job shop scheduling problem contains several constraints making it a complex problem to solve. The problem contains multiple workstations and each workstation is assigned to an individual agent. All individual agents optimize a set of rules at their specific work station while monitoring the actions of other agents and optimizing a global reward. The agents use a Deep Q-learning Network algorithm to optimize the problem. An experiment based on 4 workstations is created where initially, one agent at a time trains, the other are controlled by heuristics. After this phase, the system is executed with learning ability for all agents simultaneously. In the experiment, the largest scenario contained 50 orders and the DRL algorithm performance equals heuristic approach performance based on a 2-day training session.

Rummukainen et al. (2019) also present an algorithm for job scheduling in a manufacturing environment. The authors consider a stochastic economic lot scheduling problem (SELP) that involves scheduling production of multiple standardized products on a single machine that requires significant time and effort to switch from one product to another. Customer demand is not known in advance, and is assumed to be random and the proposed algorithm decides when to produce which product, in order to optimize a performance indicator such as expected customer waiting time, or expected production and stock holding costs. Rummukainen et al. (2019) model their problem as a Semi-Markov Decision Process (SMDP) and interact with a simulation model. Compared to a normal MDP, the SMDP takes actions every time the system's state has changed instead of after a decision epoch at MDP. The authors use a Proxy Policy Optimization (PPO) algorithm and in an experiment, the performance of this model is outperforms other heuristic models. As mentioned, the experiment consists of a single workstation with three 3 products that have a demand over 400 days of respectively 25, 10 and 10.

The work of Shi et al. (2020) addresses a scheduling problem that includes a discrete automated production line, whose transportation depends on a mechanical robot. This robot needs to transfer orders between several workstations in the production line and this process requires some scheduling operations. When each product is transferred into the next processing unit, a stochastic processing time is sampled where the scheduling system is unaware of. The authors use a DQN to solve a formulated MDP of the above scheduling problem. In an experiment, this DQN algorithm outperformed heuristic methods as FIFO, LPT and SPT. The latter are based on expected processing times. The experiment had an instance size of 7 workstations with a throughput of 6 orders per hour and training took around 24 hours.

Hubbs et al. (2020) examine applying deep reinforcement learning to a chemical production scheduling process to account for uncertainty and achieve online, dynamic scheduling, that benchmarks the results with a mixed-integer linear programming (MILP) model that schedules each time interval on a receding horizon basis. The scheduling process considers a continuous, chemical manufacturing process with a single stage and single production unit operator under stochastic demand. Transitioning the production facility from producing a product to producing a different product causes a lot of change over costs. Furthermore, demand consists out of customer demand and predicted demand, inventory costs occur on a yearly base and late shipments are penalized. These measures are monetized and the goal of the proposed methods is to maximize the profitability. The authors propose an A2C algorithm to address this problem that at its turn, uses a fairly

deep neural network of 12 hidden layers with each layer consisting 256 neurons. This approach is compared to Integer programming approaches in an experiment. This experiment consists out of scheduling 4 products on a single stage reactor for a 90 day period. The proposed approach provides a natural representation for capturing the uncertainty in a system and outperforms the MILP schedulers operating for this reason. DRL provides a viable and promising approach for chemical production scheduling. It is often much easier to incorporate uncertain elements in simulation versus in a mathematical program. This uncertainty can be represented by the DRL agent such that, once the DRL agent is trained, it can produce schedules online, that are superior to more computationally intensive methods. The schedule can be generated almost instantly via a sequence of forward-passes through a deep neural network. DRL has obvious advantages, however there are drawbacks as well. There is no guarantee of optimality with policy gradient methods apart from the REINFORCE algorithm as shown in Sutton et al. (1998).

Resource scheduling in in computer systems
In the work of Mao et al. (2016), a data center related problem is addressed with DRL. The problem is defined as scheduling jobs from a job pool to a number of cluster resources. Scheduling is constrained by the CPU and memory requirements for each job. The authors use average job slowdown as the primary system objective. This is defined by the actual completion time of a certain job divided by the standard completion time of that certain job. This problem is represented as a Markov Decision Process and the REINFORCE algorithm is used to train the system. This algorithm is the fundamental policy gradient algorithm on which nearly all the advanced policy gradient algorithms are based on. The algorithm directly updates the policy weights instead of the individual states. The outcomes have been compared with several heuristic results and the authors presented their algorithm as superior in terms of optimizing the average job slowdown. Within this experiment, the agent was given 70 jobs that needed to be scheduled under different resource scenarios.

Che et al. (2020) propose a deep reinforcement learning approach for scheduling tasks to resources in a Data center. The authors state that scheduling computation tasks in a data center is a complex job due to a dynamically changing environment, limited information of tasks that arrive in the near future and dynamically changing resource requirements. The scheduling problem includes a task pool of unscheduled tasks, a machine cluster as a container for a fixed number of virtual machines where tasks are assigned to and a task scheduler that acts as a dispatcher. The authors propose a DRL model with two agents where the first agents takes actions regarding whether to execute a certain task on a certain machine and the second agent controls the resource utilization by turning the machines on and off. A real production dataset published by the Alibaba Cluster Trace Program is used to train an A2C algorithm in the experiment study. This dataset includes the collocation of online services and batch workloads about 1300 machines in a period of 12 hours. The experiment itself used a cluster size of 300 to 500 machines for a period of 12 hours. Training the algorithm took around 24 hours and results have been compared with scheduling heuristics as FCFS, SJF (Shortest Job First) and Fair. The A2C algorithm outperforms other heuristics in terms of resource utilization and average delay time.

Rjoub et al. (2020) considers task scheduling in cloud systems, that daily receive a massive number of tasks that need to be simultaneously and efficiently mapped onto the cloud resources. A task scheduling mechanism should both minimize tasks' execution delay and cloud resources utilization is of prime importance. In this article, the authors capitalize on the concept of cloud automation and propose a deep reinforcement learning-based scheduling approach to automate the process of scheduling large-scale workloads, while reducing both the resource consumption and task waiting time. The problem is defined as a MDP where, the state space of the RL network represents the amounts of available resources on the Virtual Machines (VMs) that would be hosting the tasks in terms of RAM, CPU, disk storage, and bandwidth. The action space represents the scheduling of the set of tasks received. The reward function (cost in our case) represents the cost of executing the tasks on the VMs in terms of RAM, CPU, disk storage, bandwidth, and waiting time. In an experiment where a variable number of VM (10 to 100) are considered and a fixed

number of arriving tasks (78 597), a DQN algorithm with a LSTM-based deep neural network architecture outperformed all heuristics. Traditional heuristics include first-in-first-out, shortest job first (SJF), round-robin (RR), min-min, and max-min. More 'intelligent' heuristics included fuzzy logic, particle swarm optimization (PSO), and genetic algorithm (GA).

Supply Chain Management

The work of Oroojlooyjadid et al. (2020) present an approach to the well known beer game designed by Massachusetts Institute of Technology. The beer game consists of a serial supply chain network with four echelons namely, agents—a retailer, a warehouse, a distributor, and a manufacturer—who must make independent replenishment decisions with limited information. The game is widely used in classroom settings to demonstrate the bullwhip effect, a phenomenon in which order variability increases as one moves upstream in the supply chain, as well as the importance of communication and coordination in the supply chain. Players in the game seek to minimize costs by placing orders at their predecessors. The system uses a single agent per echelon that is trained once for one specific echelon. Transfer learning is used to initiate agents at the other remaining echelons. Oroojlooyjadid et al. (2020) model this problem as a Partially Observable MDP (POMDP). This model captures historic sequence data in each state whereby the state representation grows over time. In practice, the DQN agent only sees the last m periods that were used in a state. The agent can order any amount at the last upstream neighbor and the reward function applies a penalty or reward based on product shortage or inventory holding costs. In several numerical experiments, the authors trained the model on demand scenarios up to 10 orders over multiple products. The proposed model obtains near-optimal solutions when playing alongside agents who follow a base-stock policy and performs much better than a base-stock policy when the other agents use a more realistic model of ordering behavior.

Vehicle scheduling

As opposed to the problems discussed above, Hu et al. (2020) addresses an AGV scheduling problem where the authors aim to find the optimal transport time and equipment for each task, while respecting constraints from the manufacturing location. The scheduling operation directly contributes to the performance of an AGV system and it received extensive academical attention in form of heuristics or meta-heuristics. Although more recently, some academia have used Q-learning for a multi-lead carrier scheduling problem, the authors still identify limited performance and learning efficiency. The AGV scheduling problem is defined with a set of tasks, that each need to be transported to a different location with different transportation times. A Markov Decision Process is defined with a state representation consisting of number of current tasks, remaining transportation time, travel distance and working status of all AGVs. The action space is defined by several scheduling rules (FCFS, STD, EDD, LWT and NVF) and a specific AGV. So, a future action of an agent includes a specific sequencing rule for a specific AGV. A DQN algorithm is used to train on the MDP and is in an experiment compared to the individual heuristics based on the average makespan. This experiment consisted out of 800 jobs with 3 AGV per episode. The DQN model outperformed all heuristics in terms of makespan. Future research will scale the algorithm to handle larger instances, include a navigation task and a multi-objective reward function.

Ying et al. (2020) presents a novel actor-critic model for metro train scheduling with circulation of limited rolling stock. These complex systems require effective schedule control policies in real time and has been addressed in previous literature by both exact and heuristic approaches. In the presented problem, a service loop of metro rides is considered with a set of station nodes each with a set of platforms. Furthermore, after a days service, all trains need to return to a depot. The authors present a actor-critic (A3C) DRL model that is used to solve an experiment wherein 8 stations, 24 carriages and 4 hour of service are used. In this experiment, a benchmark of established heuristics is used that include Differential Evolution (DE), Particle Swarm Optimization (PSO) and Genetic Algorithms (GE). Based on the objective of minimizing both passenger waiting times and train operation expenses, the presented actor-critic model outperformed the benchmark provided by several established heuristics.

Warehousing operations
Finally, this thesis project forms the continuation of the work of Cals et al. (2021). The authors proposed a DRL algorithm for the order batching and sequencing problem (OOBSP) in picking operations within a warehousing concept. This warehousing concept consists of an area where manual picking is performed and an area where automated shuttles retrieve orders (AS/RS). Within an OOBSP, a decision mechanism needs to be designed in order to decide whether orders will be picked with a pick-by-order or a pick-by-batch strategy and what the sequence of those batches is. The authors frame this problem as a SMDP wherein the state of the warehousing system is represented by a number of order categories. These order categories are defined based on whether the order is a single-item-order or a multiple-item-order, in what storage location they need to be picked and based on their coming cut off time. The agent has a pick-by-order / pick-by-batch action for each order category. The system has an objective of minimizing the number of tardy orders (i.e. number of order that are processed before their respective cut off times). A PPO algorithm is used for learning an agent and an experiment is used to compare this trained agents with traditional heuristics as LST, EDD, LPT and SPT. The experiment contains a warehousing concept in which 500 orders need to be picked. The proposed DRL algorithm outperformed heuristics on scenario's wherein the pickers were highly utilized.

Despite the fact that different instance-sizes are hard to compare in Table 2.3, it stands out that current DRL algorithms addressing optimization problems in scheduling operations tend to have difficulties with handling large instance sizes. Besides this, most of the problems presented in this subsection consisted of single-objective problems and fairly static environments. Furthermore, deep reinforcement learning demonstrated promising capabilities in the Operations Research Domain whereas it can provide superior performance in some optimization problems.

Paper	Algorithm	Objective	Instance size experiment
Waschneck et al. (2018)	DQN	Single	Job shop: 4 workstation, 50 orders
Rummukainen et al. (2019)	PPO	Single	Job shop: 1 workstation, 45 orders
Shi et al. (2020)	DQN	Single	Robot transport: 7 workstations, 60 orders
Hubbs et al. (2020)	A2C	Single	Scheduling: 1 workstation, 4 products
Mao et al. (2016)	REINFORCE	Single	Data center: 2 resources, 70 jobs
Che et al. (2020)	A2C	Single	Data center: 500 resources, 12 hour
Rjoub et al. (2020)	DQN	Multiple	Data center: 100 resources, 70.000 tasks
Oroojlooyjadid et al. (2020)	DQN	Single	Supply chain: 4 echelons, 10 orders
Hu et al. (2020)	DQN	Multiple	AGV: 3 AGVs, 800 jobs
Ying et al. (2020)	A3C	Multiple	Metro: 8 stations, 24 carriages, 4h service
Cals et al. (2021)	PPO	Single	Warehouse: 10 pickers, 500 orders

Table 2.3: Overview of optimization problems solved by DRL in OR domain

Conclusion on DRL implementations for optimization problems
Academia apply deep reinforcement learning on optimization problems in Operational Research domain but it appears that there are some challenges in solving large instance-size problems. In most work, proposed models are executed on single objective problems and relatively small instance-size problems that hardly exists in the real-world. These relatively small instance-size experiments require already a considerable amount of computational resources to obtain reasonable performance. This may be acceptable for small problems, but as instances grow bigger, this fact severely limits the applicability of these methods to many real-world instances. Similarly for the OOBSP of DeepRele, it is expected that this problem also requires a considerable amount of computational resources in order to enlarge to problem instance size, solve within a dynamic environment and a multi-objective variant of the problem.

2.3 Solving real-world problems with DRL

This section presents a set of techniques that can be used to enhance the traditional DRL approach in order to solve real-world operational problems. These real-world operational decision problems have a large instance size, have a multi-objective aspect and require to find a policy that address dynamic environment. The first subsection presents a technique called imitation learning what can benefit the sample efficiency of DRL approaches and allow for a large instance size. In the second section, techniques for solving multi-objective problems are addressed. These form of problems are firstly defined after which several solution methods are presented. In the subsequent section, this chapter provides a set of hyper-optimization techniques that could be used concurrently with the DRL approach. This hyper-parameter optimization technique could enhance the DRL approach in solving dynamic environments. All of these techniques can optimize certain parts of the DRL approach. Lastly, a technique is included to distill a policy of DRL approaches. This could be helpful in understanding the learned policy and applicability of DRL approaches in the real-world.

2.3.1 Imitation learning

DRL algorithms usually require a huge amount of interactions with the environment to learn feature representation of complex states, along with a complicated policy. Scaling the instance size of a problem can have extensive consequences in computational complexity. The number of steps per episode increases because the environment requires more time to solve it. Besides this, reward will be even more sparse. As the number of steps per episode increase, the moment between the agent starts and the agent receives a positive reward of the episode becomes larger. It will be harder for an agent to learn a value function of state-action combinations. To cope with the above challenges and to enable DRL to solve real-world operational problems, Yeo et al. (2020) provide an approach that integrates some kind of expert demonstration data with DRL. The demonstrations are used to initialize desired behaviour and there are several solution methods that can be used.

Imitation learning, also known as learning from demonstrations or apprenticeship learning, has benefited from recent progress in core learning techniques, increased availability, fidelity of demonstration data, as well as the computational advancements brought on by deep learning. The objective of 'pure' imitation learning is given some demonstrations, train a policy that eventually mimics these demonstrations (Yisong and Hoang M, 2018). Like RL, imitation learning requires an environment modelled as a MDP and an agent performs different actions in this environment based on its π policy. The simplest form of Imitation Learning is behaviour cloning, which uses supervised learning to learn the expert's policy. Based on state-action based demonstration data of an expert, a supervised learning approach is used to predict an action of a given input state. This approach can be problematic because it assumes that the state-action pairs are independent and identically distributed (i.i.d.) while in a MDP, an action in a given state induces the next state, which breaks the previous assumption. Still, behavioural cloning can work quite well in certain applications because of its simplistic approach (Yisong and Hoang M, 2018). Below, two implementations of this type of solution method are presented.

Deep Q-learning from Demonstrations (DQfD) by Hester et al. (2017) adds a supervised loss to the loss function of a DQN that increases the overall loss values when an agent's action is different from an expert's action. Based on a Deep Q Network presented before, the replay buffer is extended to contain also demonstration trajectories that are stored permanently, whereas older actor data is replaced by incoming actor data. During the learning process, a mini batch is sampled from the replay buffer and is given as input to a neural network. The neural network is back propagated by the sum of all losses. The authors of state that high quality demonstrations are required for obtaining good performance using this solution method.

Gao et al. (2018) propose an unified reinforcement learning algorithm, Normalized Actor-Critic (NAC), that effectively normalizes the Q-function, reducing the Q-values of actions unseen in the demonstration data. NAC tries to alleviate the over-fitting problem of DQfD by using an unified loss function (as opposed to distinct loss functions of DQfD) to process both off-line demonstration data and online experience based on the underlying maximum entropy reinforcement learning framework. The actor-critic method differs from standard methods by the fact that it utilizes a soft policy gradient formulation in which it uses a Q-function gradient that reduces the Q-values of actions that were not observed along the demonstrations. This avoids pushing up the Q-values of actions that are not demonstrated. NAC is setup differently from traditional actor-critic models.

2.3.2 Multi-objective reinforcement learning

Many decision making problems in the real world require the consideration of more than one objective. The process of optimizing multiple objectives systematically and simultaneously is Multi Objective Optimization (MOO). Formally, MOO is the process of simultaneously optimizing multiple objectives which can be complementary, conflicting as well as independent. The application addressed in this literature review, DeepRele, provides a trade-off between two objectives: minimizing the number of tardy orders and minimizing picking time per order. This subsection introduces MOO for reinforcement learning and presents several methods.

Many (sequential) decision making problems in the real world try to optimize multiple objectives. Multi Objective Reinforcement Learning (MORL) extends the conventional single-objective reinforcement learning methods to integrate two or more objectives simultaneously. The objectives of MORL are often conflicting so that maximization of one objective normally leads to minimization of another. This is a more challenging scenario where trade-offs among objectives need to be considered. Evaluation of MORL algorithms can be done on a Pareto front, which represents compromise solutions among the objectives (Nguyen et al., 2020). There is no objectively optimal solution to this type of problems, but there are subjectively optimal solutions that fit the needs of the human decision maker. In terms of actual decision-making, the objective of solving a MOO problem is referred to supporting a human decision-maker in finding the most preferred Pareto optimal solution (Miettinen, 2012). This human decision maker can subsequently decide on one solution to be implemented in practice (Hwang and Masud, 2012). For the specific OOBSP, domain experts have some idea on prioritizing a certain objective during a certain period based on order arrival patterns and resource availability. However, the relative importance of objectives, that are reflected in the reward functions of the DRL approach, are very hard for experts to define. Hence, this could be solved by learning to determine the priority trade-off automatically.

As multi-objective optimization problems using reinforcement learning are not new, several algorithms have been proposed mostly in more general multi-objective optimization problems. Most of these algorithms are based on a scalarization principle to transform the multi-objective problem to a single-objective problem (Vamplew et al., 2008). These scalarization algorithms include linear or non-linear approaches and have not a scalar reward signal but a vector where each element corresponds to an objective. The linear scalarization algorithms compute with a weight vector $w = [w_1, w_2, ..., w_n]$ and a reward vector $r = [r_1, r_2, ..., r_n]$ where n is the number of objectives, the function for the global reward is shown in Equation 2.6.

$$r = \sum_{i=1}^{n} w_i \cdot r_i \tag{2.6}$$

A disadvantage of this method is that it only can find solutions in an convex Pareto set. More advanced scalarization algorithms include for example the Chebyshev scalarization function (Perny and Weng, 2010). This more advanced method uses preference information received from a human decision maker to find a set of efficient solutions. The preference information consists of a weight

vector $w = [w_1, w_2, ..., w_n]$ and an ideal point or utopian vector $z^* = [z_1^*, z_2^*, ..., z_n^*]$. With this, the scalarization can be computed.

$$r = \max_i [w_i | f_i - z_i^* |] \tag{2.7}$$

Within this formula, f_i represents the reward per objective i. The main advantage of this function is that it can find all points in a non-convex Pareto set. There are many more scalarization methods and each has it own advantages and disadvantages. For more of these methods, one can refer to Kasimbeyli et al. (2019). Using these scalarization functions, a human decision maker can decide on the priority it allocates to each objective in solving a optimization problem. So in the case of the OOBSP, a human decision maker can for example allocate more priority for minimizing the number of tardy orders and less priority for minimizing the order picking costs. Some of these scalarization methods have been implemented in approach were deep reinforcement learning is used. Two papers have been presented below.

The work of Nguyen et al. (2020) introduces a framework that can be used for solving multi-objective problems with DRL. This framework requires a set of weights for the multiple objective as input and outputs a Pareto graph with all solutions. This work uses both a linear scalarization function as a more complex scalarization function, thresholded lexicographic ordering (TLO). This latter scalarization method maximizes the performance on one chosen objective subject to meeting a threshold level of the performance on the remaining objectives. This framework is tested on the games Deep Sea Treasure and Mountain Car. Different scalarization methods and single & multi policy agents are compared based on the hypervolume of the Pareto graph. As an additional note, the authors state that an aligned reward function is of high importance when solving multi-objective optimization problems using DRL.

The work of Hayes et al. (2021) provides a guide to the application of multi-objective DRL based methods to difficult problems. In multi-objective problems, the properties of the user's utility can drastically alter the desired solution. This utility is defined as the perceived importance of each objective within the multi-objective problem. The authors present an utility-based approach that aims to derive the optimal solution set from the available knowledge about the utility function of the user, and which types of policies are allowed. This knowledge allows constraints to be placed on the solution set, reducing its size and thereby improving learning efficiency and making it easier for users or systems to select their preferred policy. Furthermore, the authors of Hayes et al. (2021) present several factors that influence the design of multi-objective systems. These factors need to be taken into account when designing such a model.

2.3.3 Hyper-parameter optimization for reward shaping

As for all RL projects, finding an aligned reward function is critical to eventually obtain good performance and is referred to as reward shaping. Additionally, when considering a multi-objective problem, a large instance size and a dynamic environment, this process is even more challenging. As a multi-objective variant of the OOBSP is considered, setting these weight between components in the reward function is considered as a delicate task whereas minimizing one objective often leads to the maximization of another objective. With an aligned reward function, an agent can learn a policy that could possibly outperform other methods. In a RL project, shaping a reward function is a process wherein a domain expert advises on a set of rules that reward/punish an agent. With this set of rules (referred to as a reward function), a policy is learned, the agent' performance is observed and adjustments to the reward function are proposed. This process is repeated until a desired level of performance is obtained. This iterative process is feasible for a small instance size problems but it can become problematic when scaling the instance size and including an additional objective. There are a few techniques presented in this subsection that could be used to improve / automate the process of finding an aligned reward function and help the DRL approach to be able to cope

with multi-objective, large instance, dynamic environments. The idea of improving/automating artificial intelligence algorithms is not new and help to increase the applicability in general (Claesen and De Moor, 2015). A lot of work has been done in the field of AutoML where several tasks in Machine Learning have been automated using hyper-parameter optimization. More recently, work appeared on AutoRL that tries to automate the process of learning a good policy using a DRL agent (Chiang et al., 2019). The authors trained a RL agent to navigate a robot thru some real-world environment based on computer vision data. The authors shaped the reward function for their agent using a Hyper-optimization method considering components in their reward function as hyper-parameters. The idea is constructed by using an optimization layer around a RL algorithm, whereby the reward function can be optimized. With this approach, the process of finding an aligned reward function and subsequently obtaining a better performing agent can be improved. It is however unknown whether this technique works for optimization problems. There are several hyper-optimization approaches that have been developed to find a good set of hyper parameters for a machine learning related problem and may be helpful in obtaining an aligned reward function for a reinforcement learning problem. These optimization approaches broadly fit within two classes: model-based and model-free algorithms.

Grid search
Grid search (GS) is one of the most commonly-used model-free methods to explore hyper-parameter configuration space (Injadat et al., 2020). It can be considered as an exhaustive search that evaluates all hyper-parameter configurations given to the grid. GS works by iteratively evaluating the Cartesian product of a user-specified set of values. The algorithm can be easily implemented and parallelized but is inefficient for high-dimensional hyper-parameter configurations, since the number of evaluations increases exponentially.

Covariance Matrix Adaption evolutionary strategy
One of the best known population-based model-free methods is the Covariance Matrix Adaption evolutionary strategy (CMA-ES) (Hansen, 2006). This simple evolutionary strategy samples configurations from a multivariate Gaussian whose mean and covariance are updated in each generation based on the success of the population's individuals. Each iteration new individuals (candidate solutions, denoted as x) are generated by variation, usually in a stochastic way, of the current parental individuals. Then, some individuals are selected to become the parents in the next generation based on their fitness or objective function value $f(x)$. Like this, over the generation sequence, individuals with better and better f-values are generated (Hansen, 2006). Two main principles are exploited in the CMA-ES algorithm: (1) Increase the probability of successful candidate solutions and (2) Record evolution path so that correlation information between consecutive steps allows for better exploration. CMA-ES is one of the most competitive optimization algorithms, regularly dominating the Black-Box Optimization Bench marking challenge (Elhara et al., 2019).

Hyperspace
Lastly, the authors of the model-based hyper-optimization method Hyperspace, Young et al. (2018), present a method that can learn the dependencies between model hyper-parameters through the optimization process. Hyperspace initially samples a small, predefined number n points of $x_i \subset X$ uniformly randomly, and compute the function values at those locations, $f(x_1), ..., f(x_n)$. Then, f is modelled by fitting a probabilistic model for the function. The authors assume that f is drawn from a Gaussian process (GP). The GP assumes a priori that the probability $p(f(x_1), ..., f(x_n))$ is jointly multivariate Gaussian distributed specified by the mean $\mu(x)$ and the covariance function $k(x_i, x_j)$ where k is a Matérn kernel. The Matérn kernel is commonly used to define the statistical covariance between measurements made at two points (Genton, 2001). Modeling f using a Gaussian process gives a posterior predictive mean function $\mu(x)$ and a posterior predictive marginal variance function $\sigma^2(x)$. Fitting this jointly multivariate Gaussian distribution models the relationship between the hyper-parameters and the objective and becomes more accurate over time. Using this distribution, it must be decided where to sample the next set of hyper-parameters x_{n+1}. The expected improvement algorithm is used to approximate the max-

imum expected improvement of a new point x_{n+1} and is displayed in Equation 2.8.

$$u_{EI}(x) = (f(x^* - \mu(x))\Phi(Z) + \phi(Z) \tag{2.8}$$

Where Φ is the standard normal cumulative distribution function, ϕ its derivative, and $Z = \frac{f(x^* - \mu(x))}{\sigma(x)}$. This expected improvement algorithm balances two competing aims: first the need to explore the domain of the objective function and secondly, to exploit the points which may be the global minimum. To prove the effectiveness of Hyperspace, several machine learning experiments have been conducted: finding clusters in sparse text documents, Gradient boosted regression problems and a convolutional network on the MNIST dataset. Hyperspace is compared with several baselines as Spearmint, Hyperband, Scikit-Optimize and Random Search. In each case, Hyperspace outperforms the different baselines. It can be denoted that no reinforcement learning experiment have been optimized using Hyperspace.

2.3.4 Distilling the policy of a DRL approach

Within deep reinforcement learning, Deep Neural Networks (DNN) perform well in learning policies for control problems. A noteworthy disadvantage of these methods is the lack of clarity as to how specific actions are selected. The root of this difficulty lies in the distributed nature of the representations embedded in the hidden layers of the DNNs (Frosst and Hinton, 2017). This domain is referred to as "Explainable AI". Decision trees offer an alternative, often more legible decision making paradigm where actions can be traced back through sequences of decision based directly on input data. This subsection elaborates on the work of Frosst and Hinton (2017), whom propose a Soft Decision Tree (SDT) to increase the transparency of a learned policy by a DNN. This technique learns a certain set of weights and biases for each node in the decision tree using mini batch gradient descent. This application is particularly useful for image-based input that have latent dependencies. The work of Che et al. (2016) proposes a more applicable method that uses a regular decision tree that splits classes on several features. Using structured data that has been computed by the learned policy, a decision tree is fitted on certain state - action combinations. It should infer a set of rules that separates different actions from each other.

2.4 Position of this research in literature

This literature review provides the academic motivation and foundation for the remainder of this thesis project. Traditional solving methods for the OOBSP provide a good understanding of the problem but lack in some dimensions for solving it completely. Furthermore, based on this review, it appears that some DRL implementations for optimization problems suffer from scalability, ability to solve multi-objective problems and the ability to solve within a dynamic environment. Solving these characteristics pose a significant challenge. These characteristics severely limit the applicability of DRL approaches to many real-world operational decision problems. Luckily, some novel and exiting techniques have been presented that address scalability, multi-objective optimization, dynamic environments of current DRL algorithms and explainable AI that could be useful in tackling these problems. Based on this literature review, this project further designs techniques for scalable DRL approaches, multi-objective optimization approaches and approaches to increase the applicability of DRL models that try to solve optimization problems. As a final disclaimer, the author of this literature review states that this study does not aim to be an exhaustive review of the OOBSP, DRL approaches and techniques that enhance DRL approaches to solve real-world operational decision problems. It does however serve as a broad literature review in order to get a proper idea of the mentioned fields and to provide a base for combining DRL approaches and techniques to optimization problems.

3. Problem Description

This chapter introduces DeepRele and the OOBSP is formalized. The presented problem is scoped in section 3.2 and a research design is given in section 3.4.

3.1 The Online Order Batching and Sequencing Problem

This section includes a system description of DeepRele warehousing concept, principles of the online order batching and sequencing problem within DeepRele and introduces several challenges that can be experienced in real world scenarios.

3.1.1 System description

As introduced, the DeepRele warehousing concept includes a sequential decision making problem wherein an online batching and sequencing operation need to be performed. This warehousing concept, DeepRele, consists out of two processes: picking an order in either a PtG or GtP storage area and processing this order at one of the three subsequent workstations (Figure 3.1). The OOBSP in this project considers the picking process in either a PtG or GtP storage area. The PtG storage area can store goods on racks or pallets. A picker walks with a picker cart through the aisles to pick items in a carton box or a tote. When picking-by-order in PtG area is applied, the pickers collects items directly in a carton box and this order is ready for shipping and does not need to be released to one of the subsequent workstations. When picking-by-batch is applied, the picker collects items in a batch tote and this batch tote is transferred by conveyor to the GtP area where it is temporarily stored until a workstation request the specific batched order. The GtP system is an Automated Storage/Retrieval System (AS/RS) with shuttles that move through the system to pick totes. These shuttles bring one tote per pick tour to one of the lifts which are connected to a conveyor to one of the workstations. When picking-by-order is applied, the shuttles retrieve all items from an order in product totes from the GtP area and sends them to a Direct-to-Order workstation. When picking-by-batch is applied, two options are present. Either the stored batch tote from PtG is retrieved with optionally additional product totes from the GtP area, or multiple product totes are retrieved from the GtP area. Both options are transferred to the Sort-to-Order workstation.

Product and batch totes can be released to three different workstations: Direct-to-Order (DtO), Sort-to-Order (StO) and pack stations. A picker at a DtO workstation removes items from a product tote and collects items of a single order in an order carrier. If an order consists of multiple order lines (multiple SKUs), multiple product totes are provided. All associated product totes arrive in sequence and a picker places the items in an order carrier. This order carrier is client specific and can be a carton box or a product tote. Afterwards, totes with remaining items are stored back in the GtP system. Besides fulfilling orders in carton boxes, the DtO workstation can batch items. In this process, product totes from the GtP area arrive, a picker places all items in a batch tote which is stored again in GtP area. After that, the stored batch tote in the GtP area can be requested by either a pack station or a Sort-to-Order (StO) station.

The StO station includes three processes: sorting, buffering and packing. Within the sorting process, a picker removes items from a batch tote and sorts them in a put wall. This put wall is a rack with shelves where a picker can temporarily allocate a shelve to an unique order. This put wall has a capacity of 50 orders depending on the average order sizes. The put wall is transferred to the buffering area once it is filled. In this area, the put wall waits until a packing operator is allocated. At packing, the operator places the sorted orders into carton boxes and transfers it.

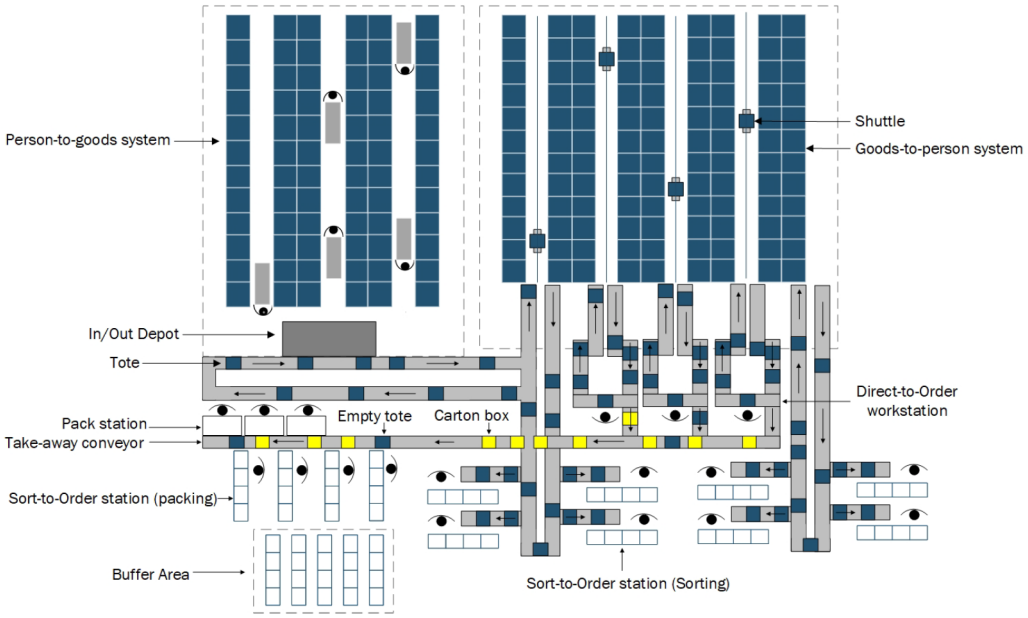


Figure 3.1: DeepRele warehousing concept with a manual- and an automatic picking area

Note that there is a difference between a pack station and the process packing at the StO station.

The pack station processes batch totes that consists out of only Single-item-Orders (SiO). SiO's are defined by orders that consists out of a single order line and a single item, so one SKU per order. A picker removes items from batch totes and places them in carton boxes that are subsequently send for shipping. Multi-item-Orders (MiO) consists out of multiple items from different of similar order lines. An overview of processing options and a graphical representation of these process flows is given in Figure 3.2 and Table 3.1.

Before shipping, two more processes take place: Value added services and closing + labelling. During value added services, specific options are added to the carton box such as customer specific packaging or preparing the order for international shipping. During closing and labeling the carton box is closed and a shipping label is applied. After this step, the bar code is scanned and the carton box is transferred to the correct outbound conveyor for shipping. Supporting processes as replenishment of SKUs and the transportation of empty totes are not considered in this concept.

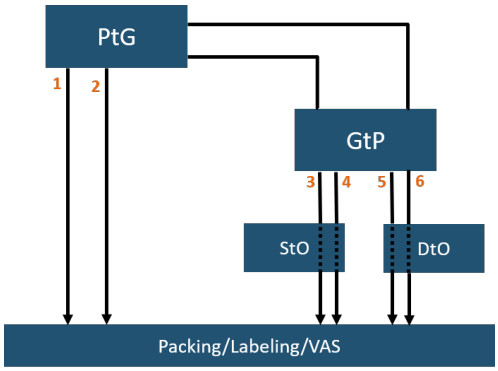


Figure 3.2: Physical process flow variants of DeepRele

Category	Location	Strategy	Route
SIO	PtG	Pick-by-batch	2
SIO	GtP	Pick-by-batch	5
SIO	PtG	Pick-by-order	1
SIO	GtP	Pick-by-order	5
MIO	PtG	Pick-by-batch	3
MIO	GtP	Pick-by-batch	4
MIO	PtG	Pick-by-order	1
MIO	GtP	Pick-by-order	5
MIO	PtG & GtP	Pick-by-batch	3
MIO	PtG & GtP	Pick-by-order	6

Table 3.1: All process flow variants of DeepRele

Table 3.1 depicts the possible variants that an order can go through the DeepRele system. The first distinction between order types is the fact whether an order is a Multi-item-Order (MiO) or a Single-item-Order (SiO). Based on this characteristic, the storage location of the order and the picking strategy of the order, a specific route through the system can be assigned. All these routes have different processing times depending on the number of items in an order and the number of orders in a batch. However, the processing time of an pick-by-order decision is in most of the cases significantly shorter than a pick-by-batch decision. On the contrary, DeepRele can achieve a larger throughput when a pick-by-batch strategy is selected because of a higher picking efficiency.

3.1.2 The online order batching and sequencing problem for DeepRele

As described in the literature review in chapter 2, the OOBSP is defined as grouping customer orders into picker orders (batches) and subsequently sequencing these batches to optimize a certain single or multiple objectives (Henn et al., 2012). More specifically, the batching operation includes a set of orders that need to be assigned to a set of batches. In the case of DeepRele, individual information about orders is available and includes the cutoff time and item ID(s). A model needs to assign these orders to a batch such that the number of orders that become tardy are minimized. A tardy order is defined by an order that is finished after its respected cutoff time provided by the context. Furthermore, the model needs to assign orders to a batch such that the order picking costs are minimized. The order picking costs are defined by the amount of time that a single order requires to be picked. Within a batch with batch size larger than 1, the order picking costs per order is fairly lower than for a batch which contains only a single order. In parallel to this batching operation, the model needs to decide on the sequence of batches in adherence to the aforementioned two objectives. Furthermore, the OOBSP within DeepRele is considered as an online problem and needs to perform batching and sequencing operations in real time.

Within the OOBSP for DeepRele, a model needs to decide on a picking strategy where either an order is picked-by-batch or an order is picked-by-order. Each picking strategy will have different operational consequences. A pick-by-batch picking strategy has relatively short picking times per order (order pickers picks more efficiently) but has a higher lead time (orders within a batch wait on each other). On the other side, when deciding for a pick-by-order strategy, the picking operation has relatively long picking times per order (no synergy of picking several orders simultaneously) but do has short lead times (orders do not have to wait on each other). As mentioned, both strategies have different operations consequences and could be the optimal decision on different scenarios. In a case where a certain order has an imminent cutoff time, ideally this order would be picked with a pick-by-order strategy as this has a low lead time. Differently, in a case where there are no imminent cutoff times for orders, ideally orders should be picked with a pick-by-batch strategy in order to decrease the picking time per order. This trade-off between consequences of the pick-by-batch and pick-by-order action should be included in the proposed model.

3.1.3 Real world characteristics for the OBSP

As mentioned in the introduction, the work of Cals et al. (2021) contains some assumptions whereby the environment used for their experiments becomes fairly theoretical. This sub section identifies several real world aspects of the OOBSP for DeepRele and these are be addressed in the remainder of this project.

First of all, problems with real-world characteristics are often presented as multi-objective problems. Minimizing one objective often leads to the maximizing of another objective. Within Vanderlande, the performance of DeepRele systems is usually evaluated by multiple objectives as the number of tardy orders and order picking costs. This dual objective optimization problem, aims to minimize the number of tardy orders and order picking costs. Between these objectives,

a trade-off can be identified. When performing efficient order picking (i.e. low order picking costs per order when pick in a batch), the lead time of orders increases. If this lead time becomes too large, the specific order can become tardy as it leaves the system after its pre-defined cutoff time. This joint optimization problem is an extension of the work of Cals et al. (2021) where the authors only solved the OOBSP for minimizing the number of tardy orders. For the specific OOBSP, domain experts have some idea on prioritizing a certain objective during a certain period based on order arrival patterns and resource availability. However, the relative importance of objectives, that are reflected in the reward functions of the DRL approach, are very hard for experts to define. Hence, this could be solved by learning to determine the priority trade-off automatically based on states and the environment.

Secondly, solving small theoretical instances can prove effectiveness of solving a OOBSP using a DRL approach, scaling to larger instance was not possible in the work of Cals et al. (2021). DeepRele is designed for a throughput of up to 5000 orders per hour. When scaling the OOBSP, this has several consequences for learning behaviour of the DRL agent and computational complexity. The number of steps per episode increases because the environment requires more time to solve it. Besides this, reward will be even more sparse. As the number of steps per episode increase, the moment between the agent starts and the agent receives a positive reward of the episode becomes larger. It will be harder for an agent to learn a value function of state-action combinations.

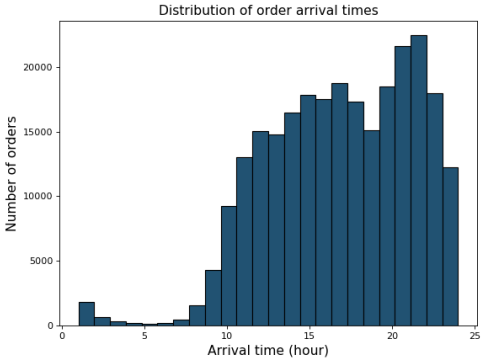


Figure 3.3: Arrival pattern of orders

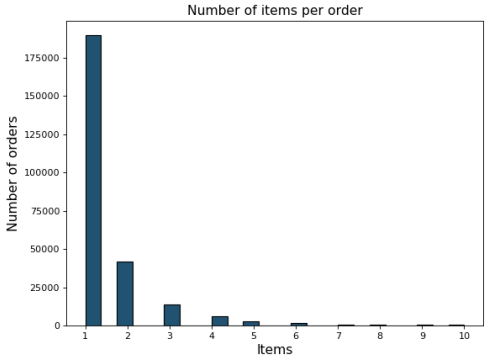


Figure 3.4: Number of items per order

Lastly, real world environments are often subjected to change and dynamics over time. Also in the case of the OOBSP of DeepRele, orders can arrive 24 hours per day and in general, orders that arrive before 23:00 have to be delivered the next day. As DeepRele is designed for e-commerce applications, the data that is be used for this project is from a large e-commerce company in The Netherlands and the arrival pattern is depicted in Figure 3.3. This pattern in Figure 3.3 brings along several operational difficulties in terms of resource planning. Firstly, orders arrive in real time so a proposed solution should be able to perform a batching and sequencing operation in an online setting. According to the literature review in Chapter 2, work on this is sparse and requires specific attention. Secondly, when looking at the arrival pattern itself, a large part of the orders arrives at the end of the day. In order to account for this additional workload, resources need to scheduled accordingly. Furthermore, a dynamic order batching algorithm could greatly benefit operational performance when it adjust its batching strategy according to this order arrival pattern. During a day, when there are little imminent shipping deadlines (cutoff moments), focus more on order picking efficiency and more towards the end of the day, focus more on minimizing tardy orders. Additionally, the histogram in Figure 3.4 depicts the number of items per order for this company. So-called single-item-orders (SIO) have different lead times than multi-item-orders (MIO) and so, have different consequences in terms of operational performance and ideal batching strategy.

3.2 Scope of the problem

With the DeepRele system and the OOBSP described in previous sections, the scope of the problem can be defined. This project addresses a multi-objective optimization problem using deep reinforcement learning where the main challenge is to find a good trade-off between the multiple objectives and apply it in real world environments. In order to focus on solving this challenge solely, the following aspects are of importance to thesis project:

- The batching decision (pick-by-batch / pick-by-order) is be performed by the DRL agent, the subsequent sequencing operation is performed using a heuristic.
- Due to time constraints of this project, designing a routing strategy is not included in this thesis project to solely focus on the batching and sequencing decision problem. This is however considered an important aspect in warehouse operations.
- These real-world characteristics contain a multi-objective problem, a large instance size and is subjected to change in terms of orders arriving into the DeepRele system.

3.3 Problem formulation

Given the OOBSP system description and scope for this problem leaves us with the problem in which the current DRL methods needs to improved in order to optimize both the number of tardy orders as the order picking costs of the OOBSP within the DeepRele system. This improvement needs to be constructed in a way that the OOBSP is solved without violating DeepRele constraints. With this, the problem can be formulated verbally: given a DeepRele setting with a predefined set of resources, the DRL algorithm needs to sequentially determine a picking strategy for each order and sequence the batched orders. This needs to be done such that the system constraints of DeepRele are satisfied and the number of tardy orders and the picking cost are optimized.

An example of this problem formulation is a situation where 10,000 orders need to be processed and where the DRL agent should provide pick and sequence strategies. The DRL agent should make a trade-off between two picking strategies: either pick-by-order or pick-by-batch. When orders are picked with a pick-by-order strategy, the individual orders have short lead times but the system's throughput is low. However, when orders are picked with a pick-by-batch strategy, the system's throughput is high but lead times of individual orders are high as well. If some order has an imminent cut-off time, there is a risk of a tardy order when choosing a pick-by-batch strategy. Picked-by-batch orders have a longer lead time because orders need to wait until all other orders are picked and because batches require additional sorting operations after picking.

Additionally, this problem is represented as an abstract mathematical optimization problem. It is abstract in the sense that it cannot be directly solved by a mathematical programming engine because of the online aspect but it is meant to precisely define the problem. The initial formulation of Cals et al. (2021) is used and changes have been made to include the multi-objective aspect.

Sets. The following sets define the problem domain.

O : the set of orders
B : the set of batches
T : the time
X : the types of resources $X = \{p, g, d, v, d\}$, where p is PtG pickers, g is GtP shuttles, d is DtO work stations, v is StO work stations and b is packing work stations.

Decision variables. The decision variables model the picking strategy per order o at a particular moment in time t.

$PbO_o^t \in \{0,1\}$: pick order $o \in O$ by order at time $t \in T$
$PbB_o^t \in \{0,1\}$: pick order $o \in O$ by batch at time $t \in T$

Parameters. Each order has several properties and each resource has a capacity.

$c_o \in \{SIO, MIO\}$	: $o \in O$ is a single item order (SIO) or a multiple item order (MIO)
$l_o \in \{PtG, GtP, both\}$	: $o \in O$ is available in the Person-to-Goods storage area, Goods-to-Person area or in both
$e_o \in \{e_1, e_2, e_3\}$	: $o \in O$ is in cutoff category 1,2 or 3 where 1 is a cutoff time between 0 and 15 min from now, 2 is a cutoff time between 15 and 40 min from now and 3 is a cutoff time of more than 41 min from now.
$a_o \in \{T\}$	: arrival time of order $o \in O$
$d_o \in \{T\}$	: cutoff time of order $o \in O$
cap_x	: $x \in \{PtG, GtP\}$ is capacity of resources PtG and GtP
N	: Maximum batch size in number of orders

Abstract parameters. Several parameters are left abstract, specifically: the service time of an order (i.e. the time that an orders spends in the system, waiting or being picked or packed). Also the picking time of a picking resource attributed to an order (i.e. when a set of orders are picked-by-batch, a picking resource attributes a specific picking time to each order from this batch. In other words, how much time is a picking resource busy with picking a specific order from a batch.) Also, the batch information to which an order is assigned is left abstract, and the number of resources of a particular type that is occupied at a particular point in time are left abstract. These parameters must be computed, depending on the state of orders that are already being picked and decisions that were made with respect to these orders. In the next chapter, several assumptions are made on how these parameters are computed and show how the problem can be solved as an MDP under these assumptions.

$s_o \in \{T\}$	: the service time of order $o \in O$
$p_{o,x}$	: picking costs: the picking time of picking resource x attributed to order $o \in O$ multiplied by a cost value
$ba_{o,b} \in \{0,1\}$	: order $o \in O$ is part of batch $b \in B$
occ_x^t	: the number of occupied resources of type $x \in X$ at time $t \in T$

Problem. The objective is displayed in the equations below. Let $tardy_o = 1$, if $a_o + s_o > d_o, 0$ otherwise.

$$\min \quad F = \begin{cases} f_1 = \sum_{o \in O} tardy_o \\ f_2 = \sum_{o \in O} p_{o,x} \end{cases} \tag{3.1}$$

$$\text{s.t.} \quad \sum_{t \in T} PbO_o^t + PbB_o^t = 1 \qquad \forall o \in O \tag{3.2}$$

$$PbO_o^t \cdot t + PbB_o^t \cdot t \geq a_o \qquad \forall o \in O, t \in T \tag{3.3}$$

$$occ_x^t \leq cap_x \qquad \forall x \in X, t \in T \tag{3.4}$$

$$\sum_{o \in O} ba_{o,b} \leq N \qquad \forall b \in B \tag{3.5}$$

$$\sum_{t \in T} PbB_o^t = \sum_{b \in B} ba_{o,b} \qquad \forall o \in O \tag{3.6}$$

In objective function 3.1, two functions are minimized, the number of tardy orders and the picking costs. In order to minimize tardy orders, orders need to be processed before their respective cutoff time. In order to minimize picking time per order, orders need to be picked with a pick-by-batch strategy. Constraint 3.2 assures that orders are assigned to one picking strategy. Constraint 3.3

constrains that orders can only be picked after they have arrived. Furthermore, constraint 3.4 ensures that resource abide the predefined capacities. Constraint 3.5 limits the maximum batch size of batches. Lastly, constraint 3.6 makes sure that orders cannot be split up and be allocated to multiple batches.

3.4 Research design

In this section, the setup of the research is discussed to solve the problem formulated above. The current DRL setup of Cals et al. (2021) is discussed in Chapter 4 and several adjustments are proposed. Furthermore, Chapter 4 also contains solutions that enables DRL methods to solve the multi-objective variants of the OOBSP for DeepRele in real world environments. To test these techniques, Chapter 5 presents an experiment where several scenarios of the OOBSP are discussed. Furthermore, since DeepRele is a completely new system and no current solution approach exists, the performance of the DRL cannot be validated with real-world data. Therefore a simulation model is developed that can test different online order batching and sequencing techniques and allows for experimentation. Design of this simulation model is included in Chapter 4. The results for all experiments are presented in Chapter 6 and this chapter additionally performs a policy and robustness analysis. All findings are concluded in Chapter 7.

4. Solution methods

The multi-objective OOBSP addressed by DRL in this work forms the continuation of the single-objective OOBSP presented in Cals et al. (2021). This chapter gives a brief introduction to the approach of Cals et al. (2021) as this forms the starting point. This current DRL approach considers a small instance single-objective OOBSP problem for the DeepRele warehousing concept. This approach proved effectiveness on small instances and a single-objective variant of the OOBSP, but did not demonstrate effectiveness on a multi-objective OOBSP with real-world characteristics. Furthermore, this chapter proposes several adjustments in order to solve a multi-objective variant of the problem and to cope with challenges that are included in a real-wold characteristics. Based on findings in Chapter 2, finding an aligned reward function for DRL approaches that solve multi-objective problems is of high importance. Because of this, a Bayesian optimization technique is proposed that could improve the process of finding an aligned reward function for DRL approaches. Lastly, this chapter introduces two complex heuristics that are used for bench marking studies on the multi-objective OOBSP for DeepRele.

4.1 Current DRL solution method

The literature from chapter 2 states that the essence of RL is learning through interaction. A RL agent interacts with its environment and, upon observing consequences of its actions, can learn to alter its own behaviour in response to rewards received (Figure 4.1). Several components of the DRL based approach of Cals et al. (2021) are outlined below.

4.1.1 Environment

The environment used by the authors Cals et al. (2021), is formulated as a Semi-Markov Decision Process (SMDP) and models the processes of DeepRele. Based on a specific state of DeepRele, an agent can take a certain action that causes the system to go into a different state. The SMDP variant of a Markov Decision Process is defined by τ : the transition time, S : a finite state space, A : a set of actions and R : the reward function. In a SMDP, the transition of time to the next state will depend on changes in the environment. Whereas in a MDP the state does not have to change after each action, the state in a SMDP always changes. In other words, when the agent performs an action and the environment is modeled as a SMDP, there is a change in capacity or orders arriving in the state. Besides this, the time τ it takes wherein a MDP changes from state after taking an action is considered constant whereas in a SMDP, this time period can be variable. All components of the SMDP are outlined below. Additionally, to explain the environment, the notion of episode is introduced. An episode in DRL is a sequence of states, actions and rewards, which ends with a terminal state. In each episode, the simulation model is started and with actions performed by the DRL agent, the DRL agent tries to solve one specific instance. Specifically in the case of DeepRele, an episode terminates when all orders have been processed.

State space S
In order to model a system that captures the dynamics of DeepRele, a state space with relevant information has been defined based on three components. These components are not trivial, many other components can be included in the state. However, there need to be some kind of trade-off between the amount of information and computational resources to learn a policy. With more information, an agent needs more computational resources to train but with too little information, an agent will not converge to a reasonable policy.

1. Orders that need to be processed by DeepRele

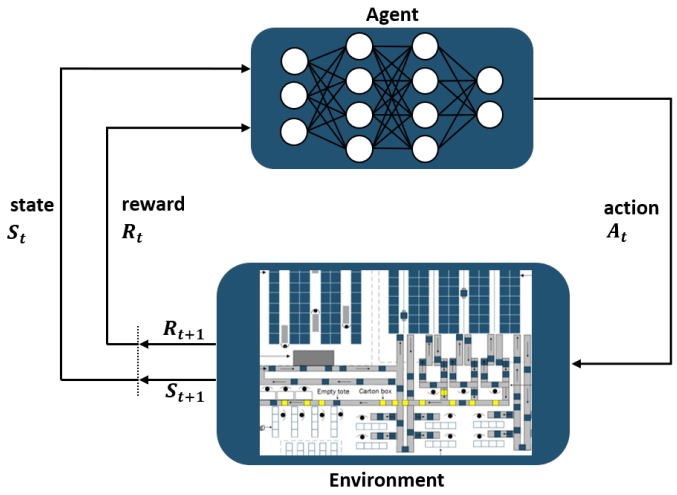


Figure 4.1: Principles of a RL agent making decisions in DeepRele

Within the state space, this component represents the set of orders that need to processed before termination of the specific episode. In this part, order categories O are defined based on order characteristics as composition C (single-item-order c_1 or multi-item-order c_2), storage location l (PtG l_1, GtP l_2 or PtG & GtP l_3) and the time until cut-off moment divided into less than 15 minutes e_1, between 15 and 40 minutes e_2 and more than 40 minutes e_3. With two compositions, three storage possibilities and three categories for duration until cut-off moment, 15 categories of O_{c_i,l_j,e_k} exists.

2. Capacity information of DeepRele's resources
   For processing a certain action, the system is required to have resources available for that certain action. For example, when an order needs to be picked at the PtG storage area, a PtG picker needs to be available to perform that action otherwise orders will accumulate. For this reason, the availability for PtG pickers and GtP shuttles are included in the state representation and are represented by a pipeline variable that includes the current available capacity plus a (virtual) queue variable. This queue variable is included to make sure that the resource does not become idle too quickly between states. There are two pipeline variables *pandg* for both the PtG resources and the GtP resources.

3. Additional information beneficial for learning
   In order to provide the agent with additional information about the general state of the system, variables $t, nandu$ have been added. Variable t represents the number of tardy orders so far. These are the orders that have left the system after their respective cut-off times. n represents the number of processed orders and u represents the time of the simulation. All these variables increase as the system gets closer to the end of the episode. It is assumed that with providing this information, the learning process is enhanced.

Action space A

The work of Cals et al. (2021) maps the action space directly to the order categories in the first component of the state representation (O_{c_i,l_j,e_k}). For all of the 15 available categories, either a pick-by-order or a pick-by-batch decision can be made. This results in 30 different actions where action 1 refers to a pick-by-order action for order category O_{c_1,l_1,e_1}, action 2 to a pick-by-batch action for O_{c_1,l_1,e_1} and so on. The 31th action is a wait action that the agent can take in case there are no resources left to process a certain set of order categories. This wait action 'waits' until the state of the system changes. This can happen either by an arrival of an order or by a resource becoming idle within the DeepRele system. Depending on the available orders O_{c_i,l_j,e_k},

capacities (p, g), the agent can choose to perform a subset of feasible actions. Only if there are no feasible actions to be taken, the agent can perform the wait action. As mentioned, the subsequent batching operation is done using the LST sequencing heuristic.

Reward function R
The reward function included in the project provides a reward at the end of the episode and penalties during the episodes. This reward function is displayed in Equation 4.1 and contains penalties for infeasible actions and tardy order. Furthermore, it contains a reward at the end of an episode based on w, the total amount of tardy orders and N the total amount of orders processed.

$$r(s, a, s') = \begin{cases} -0.005 & \text{if episode tardy order} \\ -0.0075 & \text{if tardy order} \\ (1 - w/N)^2 & \text{if episode terminates} \\ 0 & \text{otherwise} \end{cases} \tag{4.1}$$

To clarify the discussed concept of the state representation and the action space, an example is given below. The state representation is defined by the following string:
$(O_{c_1,l_1,e_1}, O_{c_1,l_1,e_2}, O_{c_1,l_1,e_3}, O_{c_1,l_2,e_1}, O_{c_1,l_2,e_2}, O_{c_1,l_2,e_3}, O_{c_2,l_1,e_1}, O_{c_2,l_1,e_2}, O_{c_2,l_1,e_3}, O_{c_2,l_2,e_1},$
$O_{c_2,l_2,e_2}, O_{c_2,l_2,e_3}, O_{c_2,l_3,e_1}, O_{c_2,l_3,e_2}, O_{c_2,l_3,e_3}, p, g, t, n, u)$

To start the example, the state representation could look like the following at the beginning of an episode:

$$(5, 20, 300, 0, 18, 100, 0, 9, 69, 0, 23, 47, 9, 0, 0, 10, 12, 0, 0, 0)$$

Within the current episode, a total of 600 orders (5+20+300+18+100+9+69+23+47+9=600) need to be processed by the DeepRele warehousing system. As discussed before, the state representation displays the number of orders in each order category, the resource availability of PtG and GtP pickers (10 and 12) and the additional information as tardy orders, processed orders and the simulation time. Currently, there are 14 orders (5+9=14) that are in a category with characteristic e_1. These need to processed and leave the warehouse in under 15 minutes in order to not become tardy. Also, there are 70 (20+18+9+23=70) orders with characteristic e_2 and have a cut-off time in between 15 and 40 minutes from now.

For example, when performing a pick-by-order actions for order category O_{c_1,l_1,e_1}, the action requires one PtG picker. Taking this action results in following new state representation of the system:

$$(4, 20, 300, 0, 18, 100, 0, 9, 69, 0, 23, 47, 9, 0, 0, 9, 12, 0, 0, 0.1)$$

Note that the order category O_{c_1,l_1,e_1} has been decreased with 1 (system processes a pick-by-order action for that order category), available PtG resources has been decreased from 10 to 9 and the simulation time has been increased from 0 to 0.1). The latter change in system can be explained by the fact when assigning orders to resources, the state of the actual system changes. As assigning the order to a resources takes fairly little time, the simulation time changes barely.

4.1.2 Simulation model

The authors of Cals et al. (2021) have used a discrete event scheduling simulation to map the dynamics of DeepRele digitally. As DeepRele is a new warehousing concept, there is no actual implementation of this system and no performance data. Furthermore, using the actual warehousing system to test DRL agent is very costly.

The discrete event simulation consists out of two events: arrival and departure that can take place at each workstation. Furthermore, each workstation has dedicated resources and processing times. An order arrives into the system and an arrival event is scheduled. When this order is in the queue for processing at a certain workstation, the simulation model checks whether there is a dedicated resource available and if so, assigns this resource to this specific order and schedules a departure moment. Once the simulation time surpasses the scheduled departure time, the resource is released and an arrival event is scheduled at the next work station. This logic is used for each route and workstation in the DeepRele system.

The simulation model is built in Automod. Automod is simulation software that can model large and complex manufacturing, distribution, and material handling systems such as our problem. Automod is widely used within VI but causes some computational problems for this project. As Automod runs on a dedicated server, training an agent requires communication with this server each step the agent takes. This process is slow, constrains the learning process and is not scalable.

4.1.3 Agent

To train a certain policy in this environment that tends to minimize the number of tardy orders on a specific resource setting, the authors of Cals et al. (2021) have used a Proximal Policy Optimization (PPO) algorithm. This is algorithm strikes a favorable balance between sample efficiency, simplicity and computational efficiency. This project included a single optimization objective, namely the minimization of the number of tardy orders. An detailed introduction to the PPO algorithm is given in the literature review in section 2.2.3. The specific implementation of Hill et al. (2018) for the PPO algorithm has been used throughout this project.

4.2 Adjustments to the current DRL setup

Solving a multi-objective variant of the OOBSP with real-world characteristics requires several adjustments to the DRL setup of Cals et al. (2021). The authors denote that the current simulation model did not allow to scale the instance size due to the computational complexity. As seen in Chapter 3, instance sizes of up to 5000 orders per hour form real-world characteristics whereas this is currently not possible. In order to address this challenge, a new, more efficient simulation model is designed. Secondly, the work of Cals et al. (2021) did not consider a completely online setting for their problem. In order to form an environment with real-world characteristics, the model is adjusted in order to cope with a real time stream of orders arriving into the system. After this adjustment, this model can be considered as a true online solution method. Thirdly, in order to transform the problem into a multi-objective problem, the reward function is adjusted in order to provide incentive to the agent for minimizing two objectives. Lastly, the action space is changed in order to improve learning behaviour for this multi-objective setting.

4.2.1 Simulation model

A discrete-event based simulation model has been designed that captures the dynamics of DeepRele. In contrary to the work of Cals et al. (2021), this simulation model has been developed using Python. The main advantage is that this aspect enables multi-processing in combination with the DRL agent, wherein the previous model, this was not possible. The simulation model is designed using principles from Boon and Boor (2020). This model uses arrival events, departure events, order entities, queues and workstations to map the flow of orders thru the DeepRele system. The authors introduce Future Event Set as variable-sized arrays where events can be added and removed. These Future Event Sets contain time-sorted scheduled future events and is

simulated in time. The events are either marked as arrival or departure and can be scheduled in the Future Event Set. An event contains several parts of information such as workstation, time and related order entity. Events are scheduled for every order entity on work stations that the specific order is required to visit. Order entities are objects that contain order-specific information and are maintained throughout the DeepRele process to store additional information. At every workstation, information is appended to this object on waiting time, processing time and transporting time. Before every workstation, there is a queue to buffer orders. If a resource at a certain work station is not available, an order waits in this queue. Lastly, the workstations are processing units that have a certain capacity and processing time. These process orders coming in and provide time-related information to the orders. In order to illustrate the principles and explained components of this discrete event simulation model, a short example is given hereafter: After a batching and sequencing operation, an order arrives into this simulation model with an arrival event at one of the two storage areas. If a resource is available to pick this order, this resource is claimed, a departure event is scheduled and the arrival event is removed. If there is no resource available, the related order entity is placed in a queue and waits until a resource becomes idle. As time progresses, the simulation model processes arrival and departure events at all respective workstations. Because this simulation model is developed using a object-oriented method, a lot of information can be maintained on individual orders, workstations and queues to report on. The entire flow chart is displayed in Figure 4.2.

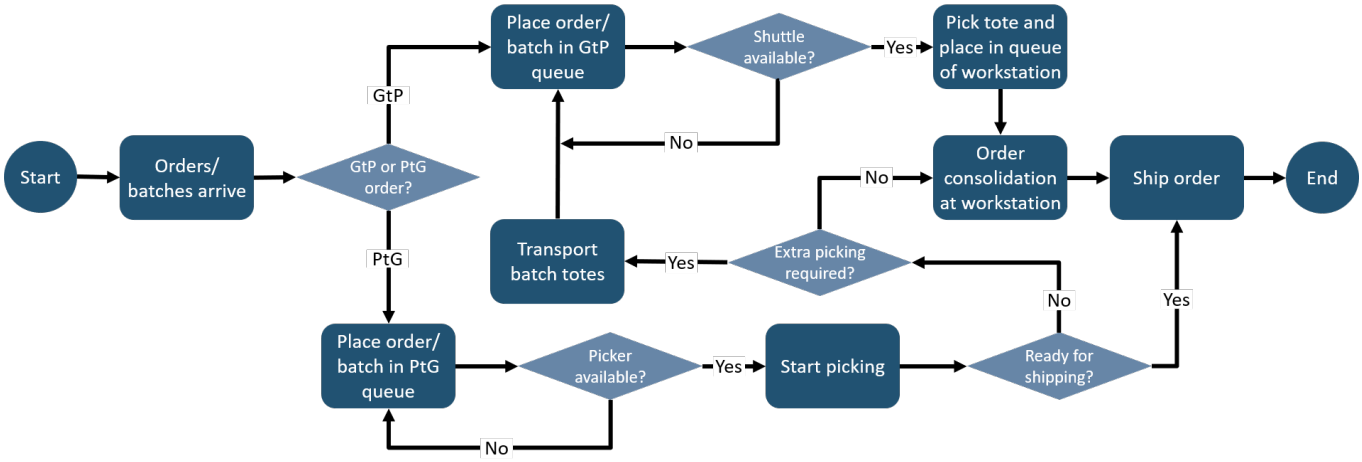


Figure 4.2: Entire simulated process flow of DeepRele

As displayed in Figure 4.2, the process makes a distinction between orders / batches based on storage location. For PtG orders / batches, the orders are placed in a queue and if a resource is available, the picking operation starts. After this picking operation, the model checks whether all items are picked. In some cases, the order contains items that need to be picked in both storage areas and extra picking is required. In other cases the batched order is not ready for shipping because all orders are within a batch tote and require consolidation. In all other cases, orders are ready for shipping and continue. For GtP orders / batches, orders together with orders from PtG that require additional picking are placed within a queue and if there is a resource available, a shuttle performs the picking operation and transports the totes to the workstation where order consolidation assures that all batches are sorted into orders. Subsequently, orders can be shipped. The arrival of orders in the DeepRele system involves uncertainty, where the proposed solution method should deal with. These orders arrive according to a order arrival distribution displayed in Figure 3.3 where a large part of the orders arrive into the system at the end of a day.

This simulation model has been validated with domain experts from VI. As there is currently no physical DeepRele system implementation, validating the simulation model in other ways was not possible. This simulation model uses a warm-up period of 15 minutes in order to fill the

system with orders. The OOBSP during this warm-up period is solved using the BOC batching heuristic which will be explained in the latter part of this chapter. This warm-up period ensures that the system is occupied with orders before the DRL approach is applied to solve the OOBSP in order to represent a real-world problem. After this warm-up period, the simulation model starts simulating a pre-defined experiment which provides the time window, incoming orders and the number of resources at each workstation. The simulation model stops simulating when all orders have been processed. Furthermore, this simulation model provides information about its state to the DRL approach. Based on this information or so-called state representation, the DRL approach must make a suitable action. The interaction between the simulation model and the DRL approach is depicted in Figure 4.3. At the start of an episode, the agent receives an initial state representation of the DeepRele system and predicts the best action to take. If this action is unfeasible, an penalty is provided to the agent and a new action is predicted on a similar state representation. If this action is feasible, the action is transferred to the simulation model. Within this simulation model, the action is simulated. When for example a picking action is predicted, the DeepRele system simulates a picking order and a new state representation of the system is compiled. When for example a wait action is predicted, the simulation model simulates until a resources becomes idle and subsequently a new state representation is compiled. This newly compiled state representation is transferred back to the agent. The agent checks whether this new state representation is terminal, i.e. when the simulation time is reached and all orders have been processed. If not, the new state representation and reward for the previous action are given to the agent to process. This process iterates until the state representation becomes terminal.

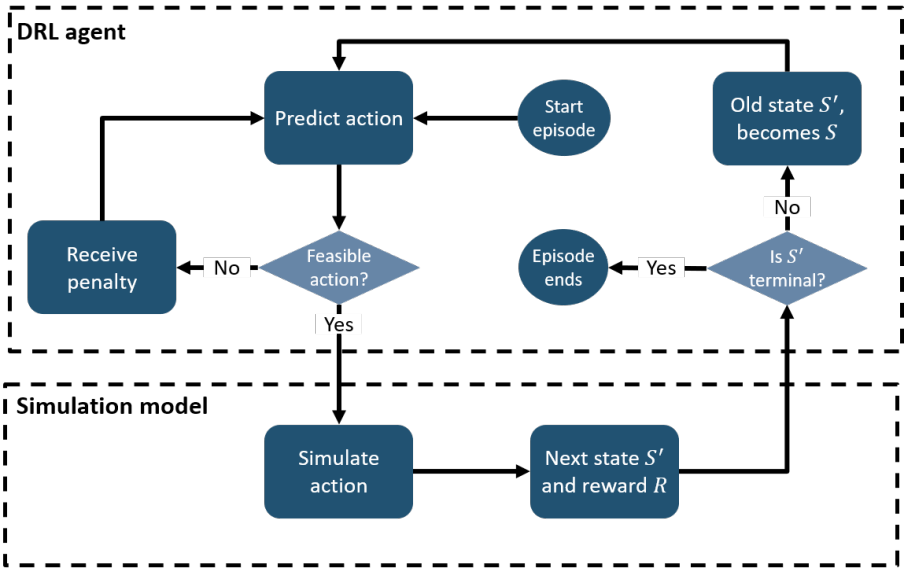


Figure 4.3: Interaction between DRL agent and simulation model

4.2.2 Imitation learning

In order to further improve the computational complexity of solving large instance size problems, this thesis project implemented an Imitation learning approach derived from de Oliveira da Costa and Zhang (2021). DRL requires a considerable amount of data before they reach reasonable performance. This may be acceptable for small problems, but as instances grow bigger, this fact severely limits the applicability of these methods to many real-world instances. The objective of imitation learning is given some demonstrations, train a policy that eventually mimics these demonstrations. Subsequently, the DRL approach can start learning based on interaction with the environment. In the case of the OOBSP for DeepRele, an online order batching and sequencing

heuristic is used to perform imitation learning upon and transfer this 'knowledge' to the DRL approach. However this did not improve performance in terms of computational complexity or system performance. This method is derived from the robotics domain where DRL approaches have large action spaces in contrary to this application. This could have impacted the added value of imitation learning. Furthermore, the sequence in which the actions are predicted have a very large impact in that domain whereas it is doubted if that is the case in the OOBSP for DeepRele.

4.2.3 Online order batching and sequencing problem

As mentioned in the introduction of this section, the newly proposed model tries to solve the OOBSP in an online fashion. This is in line with real-world characteristics where orders come in real time and the proposed model needs to make decisions accordingly. In the work of Cals et al. (2021), orders came into the system as hourly buckets. Based on this data, their model solved the OOBSP. In this project, orders arrive in the system in real time. These orders are collected in an order pool. This order pool is represented in the state representation of the DRL agent. At every time step, this order pool is updated. Once a picking action is provided by the agent, the related orders within this picking action are removed from the order pool. Enabling an online DRL method for solving the OOBSP enhances applicability in the real world.

4.2.4 Reward function of DRL approach

In order to guide an agent to learn a policy, an agent needs to be rewarded for good actions and penalized for bad actions. In order to define good and bad and how much an agent is rewarded/punished, a reward function is used. Based on the reward function introduced in section 4.1.1, a new reward function is compiled in order to reflect the multi-objective variant of the introduced OOBSP and displayed in Equation 4.2.

$$r(s, a, s') = \begin{cases} -1.5 & \text{if tardy order} \\ (1 - {}^{t}/_{N})^2 & \text{If an episode terminates} \\ -0.5 & \text{if infeasible action} \\ -\frac{(1-{}^{a}/_{t})}{50} & \text{If order is picked} \\ 0 & \text{otherwise} \end{cases} \tag{4.2}$$

The first two components in Equation 4.2 account to the first objective, the minimization of the number of tardy orders. This first components gives a negative reward to the agent every time an order leaves the warehouse after its cutoff time. This second component provides a reward when an episode terminates where t is defined the number of tardy orders and N is defined by the total number of order processed. With less tardy orders, this reward grows larger. The third component in this reward function penalizes the agent when it makes an infeasible action. This can happen when it provides a picking action for a certain order category where no resources are available. In this case it ideally provide a wait action until resources become available. The fourth component provides an incentive to the agent to pick large batches and so, minimize the order picking costs. In this equation, a is defined by the actual batch size and t is defined by the maximum batch size. In this case, an agent is not punished when the number of orders within a batch is equal to the maximum batch size. In any other cases, the agent is punished. Lastly, if no other components provide a positive or negative reward, the agent receives reward 0. An important note to make is that within the reward function, the priority for both objectives can be changed according to preference of a human decision maker. The reward function presented in Cals et al. (2021) is used as base and is improved based on trial and error. Different configurations were tested and improvements were made accordingly. Setting the components relatively to each other to solve the

multi-objective problem is a delicate process. A trial-and-error approach for shaping the reward function is used often in literature. Section 4.3 presents a Bayesian optimization technique that learns a set of weights that can be applied to components of the reward function. This will trade-off both objectives within the optimization problem and provide a certain priority to each objective.

4.2.5 Action space of DRL approach

Section 4.1.1 defines the action space for the current setup of a DRL approach of Cals et al. (2021). For the 15 different order categories, the agent can either make a pick-by-order action or a pick-by-batch action what results in 30 different picking actions and a single wait action if there are no resources available. This action space is reduced in order to decrease the computational complexity by proposing an action space of 10 picking actions and 1 wait action and this new action space is displayed in Table 4.1. Compared to the work of Cals et al. (2021), this action space does not have an individual pick-by-order / pick-by-batch action per earliness category anymore. In this new action space, there is an individual action for three earliness categories within an order composition and storage location configuration. This significantly reduces the number of possible actions the DRL agent can take and therefore also reduces the computational complexity. So for example, if the DRL agent predicts action 2, the model checks if there are any orders with composition SIO and storage location PtG within earliness category 1. If not, the model checks whether there are orders within earliness category 2. If so, the agent selects this earliness category and the orders within category SIO-PtG-e_2 are batched into a batch order. The state space still provides information on the amount of order for each individual order category (composition - location - earliness) but only the action space is not defined per earliness category anymore.

Composition	Storage location	Earliness category	Picking strategy	Action number
SIO	PtG	e_1, e_2, e_3	pick-by-order	1
SIO	PtG	e_1, e_2, e_3	pick-by-batch	2
SIO	GtP	e_1, e_2, e_3	pick-by-order	3
SIO	GtP	e_1, e_2, e_3	pick-by-batch	4
MIO	PtG	e_1, e_2, e_3	pick-by-order	5
MIO	PtG	e_1, e_2, e_3	pick-by-batch	6
MIO	GtP	e_1, e_2, e_3	pick-by-order	7
MIO	GtP	e_1, e_2, e_3	pick-by-batch	8
MIO	PtG & GtP	e_1, e_2, e_3	pick-by-order	9
MIO	PtG & GtP	e_1, e_2, e_3	pick-by-batch	10

Table 4.1: Newle defined aciton space for DRL approach

4.3 Bayesian optimization for reward shaping

Solving a multi-objective variant of the OOBSP is fundamentally different than a single-objective variant. Furthermore, the current DRL setup of Cals et al. (2021) was not able to cope with changing environments in terms of order arrival patterns. This is however considered as a real-world characteristic. Experiments only consisted of a particular time window where the environment is considered fairly constant. As seen in the literature review in Chapter 2, it is considered difficult to find an aligned reward function for multi-objective optimization problems. Additionally, these reward functions need to cope with changing environments and provide reward to the agent. The previous section made already several adjustments to the reward function in order to provide incentive to the DRL agent for optimizing both objectives and taking the dynamic environment into account but this is probably not enough to outperform the benchmark. However, it is considered that setting the reward function manually is very hard because of the dependency of the two objectives. More prioritizing a certain objective has most probably a negative impact on the other objective. Because of that, this section introduces a Bayesian optimization framework that could improve the process of finding an aligned reward function for DRL approaches in order to cope

with dynamically changing environments. This process is referred to as reward shaping in the remainder of this project. Bayesian optimization is introduced in Chapter 2 and can be used as a hyper-optimization model as done in several AutoML projects. The idea behind this framework is that basic components of a reward function can be designed by a domain expert, but only the exact weights for these components are hard to derive in a multi-objective, dynamic setting. Using this framework to obtain the weights can improve the process of finding an aligned reward function that allows for accurate learning behaviour of a DRL agent in solving a multi-objective problems with changing environments.

The automatic tuning of an aligned reward function is addressed with the framework of Bayesian optimization. This technique models a learning algorithm's performance as a sample from a Gaussian process (GP). The tractable posterior distribution induced by the GP leads to efficient use of the information gathered by previous experiments, enabling optimal choices about what parameters to try next Snoek et al. (2012). This Bayesian optimization method is part of a special model-based optimization class that work specifically well when there are expensive objective functions that need to be evaluated. In order to solve the multi-objective variant of OOBSP with real-world characteristics using this technique, a framework incorporating the principles of Bayesian Optimization has been developed and displayed in Figure 4.4.

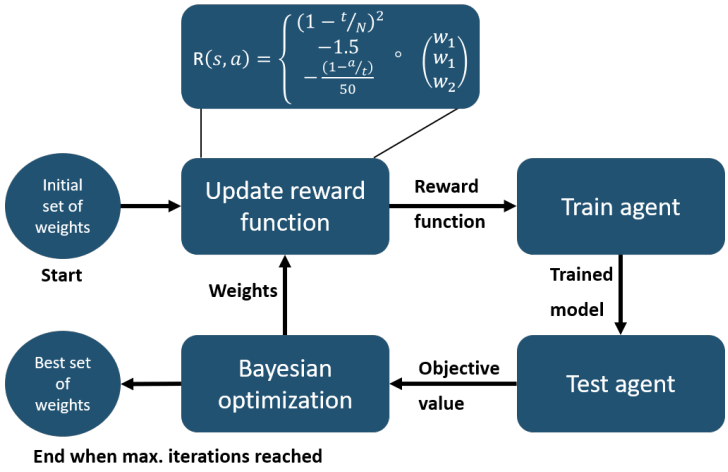


Figure 4.4: Reward shaping for a DRL approach using Bayesian optimization

This framework has been used for hyper-parameter optimization approaches in supervised learning methods. An initial set of hyper-parameters is selected, a model is trained and tested. Based on the results of the testing phase, the Bayesian optimization approach updates the surrogate function and a new set of hyper-parameters is provided. In this case, a Gaussian Process is used as surrogate function. As this project considers finding an aligned reward function for a DRL approach that addresses the OOBSP, a set of two weights are considered as hyper-parameters. Each weight corresponds to a subset of components in the reward function and can put more or less priority on these components. Subsequently, every component in a reward function can be mapped to one of the two objectives: minimizing the number of tardy orders or minimizing the order picking costs. Changing the weights within this reward function alters learning behaviour of the DRL approach.

Within the process of evaluating a certain set of weights, the framework first trains an agent and subsequently tests this agent. The objective value is computed and this acts as an input for the Bayesian optimization approach. This objective value consists initially out of two values, the percentage of tardy orders and the average order picking costs. As the Bayesian optimization approach requires a single scalar value as input, this objective value needs to be transformed. Based on the findings of the literature review in Chapter 2, for transforming the multi-objective problem into a single-objective problem, usually a scalarization technique is used. There are

several scalarization techniques present in the literature, and this framework uses the multiplication scalarization method. This method is preferable when both objectives have different scales as in this case. The number of tardy orders are expressed in a percentage whereas the average picking costs is in a range between 20 - 40. Therefore, a scalarization technique that multiplies both two objectives makes sure that a decrease or increase in the objectives value, attributes proportionally to the single objective value is preferred. With this single objective value, the Bayesian optimization approach can update its Gaussian Process distribution and sample hyper-parameters in the form of weights for the reward function. The design of the Bayesian optimization approach for this project is displayed in Algorithm 1. This algorithm firstly defines the inputs and outputs of the Bayesian optimization approach. The hyper-parameter space X consists of two parameters which have a range between 0.5 and 1.5. These figures have been estimated using several preliminary experiments. Furthermore, it requires an objective function that is depicted in Figure 4.4 that trains and tests the agent. This objective function can evaluate a set of weights in terms of a single objective. Also, Bayesian optimization requires a maximum number of evaluations. This approach outputs the best objective function with the corresponding set of weights. As a first step, the Bayesian optimization approach samples an initial set of weights from the hyper-parameter space and evaluates the objective function with this set of weights. After this initialization phase, the model iterates until the maximum number of evaluation are reached and the following steps occur. First of all, a new hyper-parameter is selected using the expected improvement formula. This expected improvement algorithm balances two competing aims: first the need to explore the domain of the objective function and secondly, to exploit the points which may be the global minimum. More details on this expected improvement formula can be found in section 2.3.3 in the literature review in Chapter 2. Hereafter, this newly sampled set of weights is evaluated using the objective function. The weights and objective values are appended to S and herewith, the Gaussian Process distribution is updated. As a last step within this iteration, it is checked whether, the obtained objective value is the best so far and if so, the parameters x^* and y^* are updated.

Algorithm 1: Pseudo-code of Bayesian optimization for the DRL approach

Input hyper-parameter space X, objective function $f(x)$, max evaluations n_{max}
Output x^*, y^*
Select an initial hyper-parameter configuration $x_0 \in X$
Evaluate the initial score of x_0 by training and testing the DRL approach: $y_0 = f(x_0)$
Set $x^* = x_0$ and $y^* = f(x_0)$
for $n \in \{1, ..., n_{max}\}$ **do**
  Select a new hyper-parameter $x_n \in X$ by maximizing the expected improvement $u(x)$
  $x_n = \arg\max_x u(x)$
  Evaluate f for x_n to obtain a new objective score $y_n = f(x_n)$
  Update the Gaussian Process with (x_n, y_n)
  if $y_n < y^*$ **then**
    $x^* = x_n$ and $y^* = y_n$
  end
end

4.4 Heuristics for bench marking

In addition to designing a deep reinforcement learning approach that solves a multi-objective variant of the OOBSP in real world environments, a set of benchmark methods need to be designed to compute a fair comparison. In the work of Cals et al. (2021) traditional batching and sequencing heuristic rules as Least Slack batching and Earliest Due Date sequencing were used to perform

a benchmark. As the OOBSP is studied extensively, this section presents two more advanced heuristics that are used in experiments in Chapter 5.

4.4.1 General variable neighborhood search for the order batching and sequencing problem (GVNS)

In the work of Bustillo et al. (2015), a General Variable Neighborhood Search (GVNS) is presented that performs an online batching operation and aims to minimize tardiness of orders. The GVNS is a metaheuristic which exploits the idea of neighborhood change in a systematic way. The aim is to descend to a local optimum or, alternatively, to escape from the basin of attraction from that local optimum. The work of Bustillo et al. (2015) is based on the approach displayed in Algorithm 2. All principles of their proposed method have been except for the evaluate function. In this case, the evaluate function has been adapted that computes both expected order pickings costs as the expected tardy orders. The expected order picking costs have been computed by average picking time and the proposed batch size of the GVNS heuristic. Furthermore, the expected tardy orders are computed by comparing the most imminent cutoff of an order within a batch with the expected processing time of this batch. These two objective values are computed for an entire solution and depict the quality of this solution.

Algorithm 2: Pseudo-code for the GVNS algorithm

Input Largest neighborhood to be explored k_{max}, maximum computing time t_{max} and initial solution S
Output Improved solution s

```
while t < t_max do
    k ⟵ 1
    while k < k_max do
        for i ∈ {0, ..., k} do
            S' ⟵ Swap(S)
        end
        k ⟵ 1
        while k < k_max do
            S'' ⟵ Insert(S'), S''' ⟵ Swap(S')
            if evaluate(S') < evaluate(S'') then
                S' = S''
                k ⟵ 1
            else if evaluate(S') < evaluate(S''') and evaluate(S'') < evaluate(S''') then
                S' = S'''
                k ⟵ 1
        end

        NeighborhoodChange(S, S'', k)
    end
    t ⟵ CPUTime()
end
```

As stated, this algorithm searches local neighborhood structures for local optima. It requires parameters k_{max} and t_{max} as input that respectively set the number of neighborhood structures explored and the time that can be used for this operation. Furthermore, it requires an initial solution s. This initial solution is constructed using the Earliest Due Date (EDD) batching heuristic. A solution to the OOBSP is represented as a list of m batches i.e., $S = \{B_1, B_2, ..., B_m\}$, where each batch B_j contains an unfixed number of orders that do not overcome the capacity of a picking cart in terms of total items. Within Algorithm 2, the approach starts with an iteration until some computation time has passed by. The parameter t_{max} is set in order to match the computing time of previous approaches in the state of the art. After this, a k variable is initialized that accounts for the number of neighborhoods explored. The first local search step is the Shaking strategy that is introduced as an effective strategy to escape from a basin of attraction. Given a solution S, the shake procedure performs a swap move that generates a new solution $S^{'}$ in k consecutive times. For the OOBSP, the principles behind a Swap move and Insert move have been depicted in Figure 4.5 where O_i depicts an order i within a batch. It is important to denote that

only feasible insert and swap moves are considered. These feasible moves are listed based on a maximum batch size for each order.

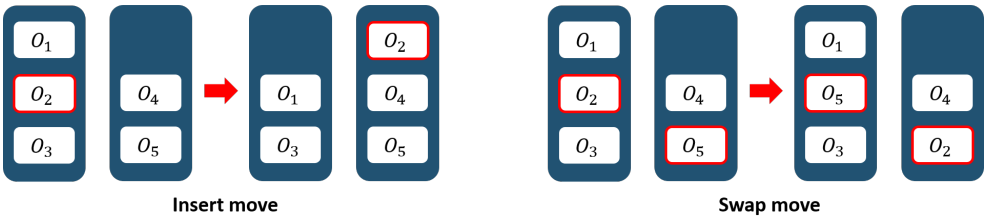


Figure 4.5: Local search moves of the GVNS algorithm

After this shake operation, the VND operation performs both of the local search moves in order to improve the solution. If one of the two moves improves the current solution, it is accepted as the new current solution. The solutions are evaluated using a function that computes average picking time and expected tardy orders. After the VND operation, all solutions are evaluated and a new current solution is selected in the function $NeighborhoodChange$. Within this evaluation function, the solution is simulated and the performance in terms of tardy orders and order picking costs are observed. This process iterates until some time t_{max} has been reached. For the OOBSP in DeepRele, a t_{max} of 0.1 seconds has been used in order to obtain reasonable computational performance in large instance size problems. Furthermore, parameters k_{max} was set at 2, as in the work of Bustillo et al. (2015). This batching heuristic is complemented with a LST sequencing heuristic. According to the work of Cals et al. (2021), this is the best performing sequencing heuristic for all batching heuristics and is therefore be used in this project.

4.4.2 Joint optimisation of order batching and picker routing in the online retailer's warehouse in China (BOC)

As introduced in the literature review in Chapter 2, Li et al. (2017) have studied a method for joint optimization of online order batching and picker routing in the online retailer's warehouse in China. The authors propose an online order batching strategy that uses a similarity coefficient that represents the similarity between two orders and is referred to as the BOC batching heuristic. The main idea is to combine similar orders as one in order to obtain a high order picking efficiency and all principles have been used as described by the authors. Furthermore, the authors use a s-shaped routing heuristic to compute the order picking route and is not considered for this project. As this project does not include specific location information for orders, the similarity coefficient is adjusted. In this case, similarity is computed based on similar items of different orders. This newly defined coefficient is displayed in Equation 4.3.

$$S_{i^*j} = \frac{\text{size of } A_{i^*} \cap A_j}{\text{size of } A_j} \tag{4.3}$$

Let A_i be the items of order i. Then for instance, if order j has 4 items in total and one of them is also included in the seed order i^*, then the similarity coefficient is 25%. This function is used in the adjusted approach of Li et al. (2017). This approach is displayed in Algorithm 3. This algorithm requires a set of unprocessed orders I, capacity constraints about the number of items per batch as input and the maximum simulation time. This maximum simulation time is the time where after no orders arrive anymore and the model should finish the remaining orders. This algorithm outputs a set of batches where all orders have been assigned to. This process iterates until there are no orders left in I and when the simulation time exceeds the max simulation time t_{max}. As this methods concerns an online batching operations, orders can arrive into this unprocessed order

list I in a real time manner. The first step of the batching operation is to select a seed order with the most items. After this step, the model iterates until the capacity constraints for an individual batch are reached. Within this iteration, the similarity coefficient is computed and a suitable order is selected based on this coefficient. This selected order is combined with the seed order and this process iterates until the batch specific capacity constraint has reached. If so, the compiled batch is appended to a list B and the included orders are removed from the unprocessed order list I. As aforementioned, this process iterates until there are no more unprocessed orders in I and the maximum simulation time is reached. Similar to the BOC batching heuristic, the GVNS heuristic is complemented with a LST sequencing heuristic for similar reasons.

Algorithm 3: Pseudo-code of BOC batching heuristic

Input Set of unprocessed orders I, batch capacity constrains, max simulation time t_{max}
Output Set of batched orders B
$B \longleftarrow$ list
$t \longleftarrow 0$
while *I is not empty and $t < t_{max}$* **do**
  Choose the order with the most items as seed order i^*
  while *i^* does not violate capacity constraints* **do**
    Compute $S_{i^*j} = \frac{\text{size of } A_{i^*} \cap A_j}{\text{size of } A_j}$ for all $j \in I$
    Sort orders by S_{i^*j} in descending order
    Select order (i^*j) with the highest S_{i^*j}
    if *Multiple orders have same similarity* **then**
      Choose the order j that arrives first
    end
    Combine order j and i^* as a new order i^*
  end
  Append i^* to B
  Remove all orders within i^* from I
end
update simulation time t

4.4.3 Simple heuristics

In the work of Cals et al. (2021), the authors use several online batching and sequencing heuristics in their benchmark. This subsection provides explanation on a selected set of these heuristics for the benchmark in this thesis project. This selected set contains only the best performing heuristics from the work of Cals et al. (2021). Both the batching as the sequencing heuristic can be combined with each other to provide integrate solutions for the OOBSP.

Least Slack Time batching rule
This heuristic initially sorts orders using an Earliest Due Date sequencing rule and combines orders into batches using a maximum batch size. After this step, this heuristic can dismantle these batches in separate orders. Orders are created once the slack time of a batch becomes negative and picking-by-order should be applied instead of picking-by-batch. Slack time is the amount of time left until the cutoff moment of an order is reached: $s = (d - a) - t$. Where s is the remaining slack, d is the cutoff time, a is the actual time and t the total remaining processing time. For this t only expected processing times are used and no waiting times are considered.

Picking-by-order small batching rule
This POSB rule is applied when picking-by-batch in PtG is applied and the batch size consists of four or fewer orders. Once this is the case, the batch is dismantled into separate orders that all have a pick-by-order picking strategy. This heuristic in combination with the LST batching rule from Cals et al. (2021) presented the best heuristic performance.

The Earliest Due Date sequencing (EDD) rule
This sequencing rule sequences all batches based on critical cutoff times. More imminent batches in terms of order cutoff times receive more priority to be processed first in the DeepRele system.

Least Slack Time sequencing (LST) rule
Lastly, the LST sequencing rule assigns priority to orders based on their slack time to be processed first. This sequencing rule shares the same principles as the LST batching rule but is used for a different operation. Similarly to the LST batching rule, only expected processing times are used and no waiting times are considered in calculated the slack time s.

5. Experiment setup

This chapter introduces the experiments. As mentioned in section 3.1.3, this project assumes real-world characteristics for the OOBSP in DeepRele. In order to test whether the proposed methods can deliver upon this, detailed experiments need to be drafted. Before this can happen, suitable data needs to be in place in order to reflect real-world characteristics. After the drafted experiments, this chapter presents a setup for the current simulation model. Furthermore, this chapter elaborates on the PPO learning algorithm and the Bayesian hyper-optimization approach. Lastly, this chapter provides objectives and performance metrics that are used to evaluate the proposed methods.

5.1 Data description

The order data that is used for the experiments is from a large e-commerce company in The Netherlands that specializes in electronic appliances. This data consists out of the following attributes: orderID, arrival time, SKU numbers of the items within the order and the quantity per item. This dataset has been introduced in section 3.1.3 and distributions for the order arrival time over the day and number of items per order and displayed are given in respectively Figure 3.3 and Figure 3.4. This dataset includes 257,585 orders with a total of 376,522 items. 70% of the orders have only 1 item, 14% of the orders have 2 items, 9% of the orders have 3 items and 4% of the orders have 4 items. According to VI, this pattern is common in the e-commerce business. For this scenario, it is assumed that there are 100,000 SKUs in the warehouse. As the DeepRele warehousing concept is new and there is no physical applications of this concept, several components as the cutoff times and storage locations of SKUs need to be added to the data set.

5.1.1 Generate cutoff times

The current dataset does not include cutoff times, and should be generated based on some assumptions. According to VI, hourly cutoff times from 17:00 until 24:00 need to be allocated to each arriving order. For this scenario, these cutoff times are assigned randomly, with a maximum of three hours later than their respective arrival time. Also, it is ensured that a cutoff moment of an order is planned a minimum of 30 minutes after the arrival time of the order. So orders that arrive before 23:30 are included in the model and have to processed before 24:00. This is in line with most of the delivery requirements of these e-commerce companies where they ensure next-day delivery and this represents real-world characteristics where orders leave the warehouse in specific groups. With this set of rules for determining the cutoff moments, a distribution of cutoff moments per day is depicted in Figure 5.2. Between 23:00 and 24:00, the greatest number of cutoff moments are located and this is representable for real world environments. This poses several operational challenges that should be addressed with the proposed model.

5.1.2 Allocate SKUs to storage locations

In order to derive sensible storage locations to individual SKUs, prior distributions are used. For this thesis project, 30% of the SKUs are stored in the Person-to-Goods storage location, 40% of the SKUs are stored in the Goods-to-Person storage location and lastly, 30% of the SKUs are stored in both storage locations. In literature, better techniques exists to allocate individual SKUs to these storage locations but as this project focuses on the online order batching and sequencing part in warehouse operations, this is left out of scope. In cases where a Multi-item-Order for example

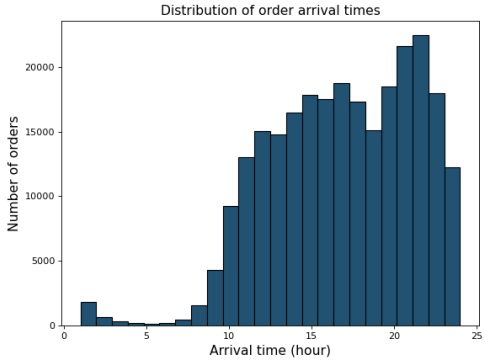


Figure 5.1: Arrival pattern of orders

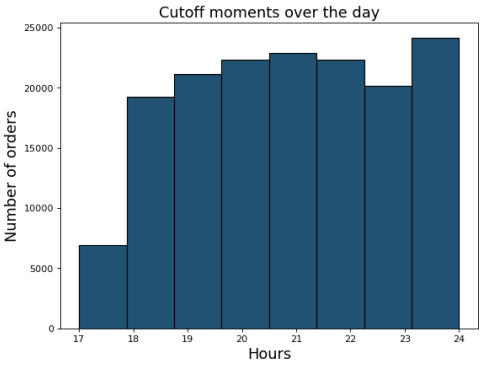


Figure 5.2: Cutoff moments over an entire day

consists of an item stored in the PtG area and an item that is stored in both areas, the model picks the item that is stored in both areas in the PtG area. In another case, when a Multi-item-Order has an item that is stored in the PtG area, and an item that is stored in the GtP areas, this order requires two pick movements (PtG & GtP).

5.2 Experiment description

This section defines the experiment setup which is used to analyze the performance of the proposed methods. These experiments are carefully composed in order to represent real-world characteristics of the OOBSP in the DeepRele system. This setup contains several components where after four concrete experiments are drafted. This first component is the throughput rate. In order to reflect real-world characteristics, a representable order throughput is required. For this project, throughput rates between 2000 and 4000 orders per hour are analyzed and for the order arrival process, the distribution displayed in Figure 5.1 is used. Secondly, based on the selected throughput setting, an appropriate set of resources is required. This set of resources consists of PtG pickers, GtP shuttles, resources for StO work stations and resources for DtO work stations. These settings are adjusted to the desired throughput rate such that the tardy orders and picking costs are at reasonable levels as verified with a domain expert from VI. Determining these baseline resource settings is done in section 6.1 in the next chapter. So for a certain throughput, it is determined using a heuristic what the least amount of resources are in order to perform bench marking on the experiments. Thirdly, simulation time consists out of either two hours or an entire day. In both options, it is interesting to analyze behaviour of the proposed methods and observe whether these methods can cope with a changing characteristics in terms of a changing order arrival pattern. Lastly, orders are released in real time, making this an online problem. At every time step, orders can arrive into the system and the state of the system changes. With these three components, four experiments are drafted.

Setting A: Processing 6000 orders from 15:00 - 17:00
For this first setting A, the OOBSP of DeepRele is addressed with 6000 orders between 15:00 and 17:00. According to Figure 5.1, this time window is fairly representable for the pattern in the afternoon. Furthermore on 17:00, the first cutoff moment is scheduled and this requires a certain policy that trades off minimizing picking costs and the number of tardy orders. The resource settings for this settings as for all other settings are computed in section 6.1.

Setting B: Processing 7000 orders from 20:00 - 22:00
This setting tries to capture the problem of orders arriving between 20:00 and 22:00. As seen

in Figure 5.1, a lot of orders arrive into the system and this poses a significant challenge to the method in determining a "good" policy. As there is still 2 hours left, the priority is not completely on minimizing tardy orders as opposed to setting C. Within this setting, there are cutoff times between 21:00 and 24:00 according to the cutoff time distribution displayed in Figure 5.2.

Setting C: Processing 6500 orders from 22:00 - 24:00
Setting C addresses the last part of Figure 5.1 where all orders have to be processed before 23:00 and 24:00. In order to achieve this, the proposed method needs to ensure that there is enough throughput by efficient order picking. On the other side, the proposed method needs to ensure that some orders are prioritized to other to ensure that orders with a more imminent cutoff time are processed first.

Setting D: Processing 50,000 orders from 10:00 - 24:00
Lastly, the objective of setting D, is to see whether the proposed method can cope with a changing pattern of orders arriving into the system. It is expected that during the day, the method focuses more on minimizing the order picking costs and during the latter part of the day, the proposed method more focuses on minimizing the number of tardy orders. This final setting captures all real-world characteristics: solving the OOBSP for large instances and the ability to cope with a changing characteristics in terms of orders arriving into the system. Solving this setting may require input from the other settings in terms of priority per objective at a specific time window.

5.3 Simulation setup

In order to guarantee a certain accuracy of the performance metrics of the system during computing the results for the defined experiments, the proposed methods are simulated for multiple runs. In order to compute the number of runs per experiment, there is a theory that provides for this process. The Central Limit Theorem is used (Boon and Boor, 2020). Abbreviated as CLT, the Central Limit Theorem, relates the sample mean of a sufficiently large number of random variables to the normal distribution. The CLT provides a means for determining the number of runs of a simulation model in order to compute an accurate mean of a performance metric. To do so, an idea of the value of σ is required which can be estimated by performing a short initial simulation with a relatively small number of runs. For each experimental setting, an initial simulation run with the heuristic and the DRL approach is computed to obtain this alpha which ranges between 0.4 and 1.1 for all experiment settings. With these computed σ, one can compute the required number of runs to obtain a $(1-\alpha)\%$ confidence interval based on accuracy ϵ with the formula depicted in Equation 5.1. For every heuristic, the following number of runs per setting are required to obtain a 95% reliability (139, 189, 246, 140). For the DRL approaches (with and without reward shaping), the following number of runs per setting are required (228, 554, 168, 172). This is computed at the start of the experiment with an initial guess of σ, an α of 1% and a accuracy of 1 decimal.

$$n > \left(\frac{Z_{\alpha/2} \cdot \sigma}{\epsilon}\right)^2 \tag{5.1}$$

In order to test whether the unknown population means of two models are equal or not, the student t-test is used (Student, 1908). This test requires that the data from both models is independent, sampled from two normal populations and the two independent models have equal variances. If so, the test statistic can be calculated by dividing the difference of model averages by the standard error of difference. This test statistic is compared with with a theoretical t-value from the t-distribution based on a significance level α and the degrees of freedom. If this test statistic is larger than the corresponding t-value, the null hypothesis can be rejected meaning that the two model means are statistically not equal.

DeepRele is a new warehousing concept and no real-life data about the performance of this kind

of concept is available yet. Determining the validity of the simulation model is therefore difficult. Looking at the individual storage locations and workstations within DeepRele, there is however extensive expertise on the dynamics of PtG storage locations, GtP storage locations and consolidation work stations in a stand alone fashion. The dynamics of an integrated system are unknown at this moment, but the functionality of the individual parts have been verified with domain experts. Furthermore, this project aims to analyze the performance of several methods on the OOBSP and all these methods use the same simulation model. Therefore, this makes it valid to compare the results with each other as each algorithm encounters similar challenges of the DeepRele environment.

5.4 Learning algorithm: Proximal Policy Optimization

5.4.1 Selection procedure of PPO

Proximal Policy Optimization, or better referred to as PPO, is a state-of-the-art deep reinforcement learning model that has been used extensively in the literature. Introduced in section 2.2.3 in the literature review, this approach provides the right balance between the ease of implementation, sample complexity and ease of tuning and according to Schulman et al. (2017), outperforms most of the other existing DRL models in several domains, in particularly the work of Cals et al. (2021). Besides this, recent advancements in improving the computational aspects of the algorithm have resulted in a GPU-enabled implementation of PPO. Given these arguments, PPO is selected for solving the multi-objective OOBSP of DeepRele with real-world characteristics.

5.4.2 Training setup for PPO

In order to successfully train the PPO model for solving a multi-objective OOBSP, a training and testing algorithm has been defined in Algorithm 4 in addition to the framework presented in Figure 4.3. The training algorithm starts with initializing the experiment by defining the number of orders, time window and resource settings. The following steps occur iterated until the number of training episode is reached. In this next step, a set of orders from the order arrival distribution is sampled, the simulation model is initialized and a first state is retrieved from the simulation model. After this part, the DRL approach selects action a based on state representation S and its current policy π_θ. The simulation model verifies whether this is a feasible action and if so, the simulation model simulates this action and returns a new state S. This can be a picking action where after an order is picked from its storage location. Furthermore, using the reward function, a new reward r_t is computed. In case of an infeasible action, this reward function provides the agent with a penalty. New state action pairs and the rewards received through an episode are correspondingly used for updating the policy π_θ with the maximization objective of the PPO algorithm. The testing process for the DRL approach consists of similar steps except that it does not change the learned policy anymore. Within an episode, the policy predicts an action based on current state S_t. When this action is not feasible, it will randomly choose a feasible action to continue testing.

The model parameters for training a PPO model for learning the OOBSP present in DeepRele are mostly similar to the original parameters of the work of Schulman et al. (2017). Several adjustments are made that are presented next. The experiments defined in a previous section are trained for 4 - 16 million steps according to the specific experiment. This training is done in an environment where 4 agents train in parallel. The actor and critic network is updated by all the agents simultaneously whereby the agent can learn more state-action pairs at the same time. The discount factor has been set at 0.9999 as the episodes can be fairly long. These discount factors are associated with the length of an episode. Longer episodes have much more variance as they

include more irrelevant information, while short time horizons are biased towards only short-term gains. A discount factor tries to find a good balance between those two challenges. The clipping parameter used for learning was kept at 0.2, ensuring that update policies cannot differ more than 0.2 from the old policy. The neural network initialized for learning is similar to the network used in Schulman et al. (2017), consisting out of fully-connected multi-layer perceptrons with two hidden layers of 64 units and tanh activation layers. PPO has been implemented using the python package Stable-baselines (Hill et al., 2018). This package has been build upon the Tensorflow framework (Abadi et al., 2015). For each experiment defined in section 5.2, the PPO agent is trained separately. Additionally, in Chapter 6, a robustness analysis is included to see whether the agents can generalize over different experiments.

Algorithm 4: Pseudo-code for training and testing process of the DRL approach

Training
Initialize experiment and number of training steps $M = 6,000,0000$
step = 0
while *step* $< M$ **do**
  Sample set of orders from order distribution
  Initialize simulation model
  Initialize first state $S_{t=0}$
  while *S is not terminal* **do**
    Select action a based on S_t and current policy π_θ
    step $\leftarrow$ *step* $+ 1$
    if *action is feasible* **then**
      Simulate action a and obtain new S_t
      Receive reward r_t
    else
      Receive penalty r_t for infeasible action a
    end
  end
  Compute advantage estimates $\hat{A}_t, ..., \hat{A}_T$
  Optimize L_t, via mini batch gradient descent
  $\pi_\theta = \theta_{old}$
end

Testing
Initialize experiment, load policy π_θ and number of testing episodes $T = 200$
episode = 0
while *episode* $< T$ **do**
  Sample set of orders from order distribution
  Initialize simulation model
  Initialize first state $S_{t=0}$
  while *S is not terminal* **do**
    Select action a based on S_t and current policy π_θ
    if *action not feasible* **then**
      Select action a randomly from feasible actions
    end
    Simulate action a and obtain new S_t
  end
  episode $\leftarrow$ *episode* $+ 1$
end

5.5 Hyper-parameter tuning: Bayesian Optimization

As introduced in section 4.3, a Bayesian optimization technique is used for reward shaping to complement the introduced DRL approach. This technique tries to find an aligned reward function that captures priorities for both optimization objectives. This technique requires a single objective and is defined as the percentage of tardy orders multiplied with the average picking costs per order. This method starts with a basic reward function that has been shaped only with domain knowledge of the DeepRele system. It is expected that the Bayesian optimization approach derives a set of weight values that can be applied to the reward function in order to learn a better policy. This set of weights are derived for setting A, setting B and setting C. In a first comparison, the results from a DRL approach with reward shaping is compared with the proposed heuristics from section 4.4 and with a DRL approach without reward shaping. Subsequently, a DRL approach is trained for setting D where the reward function is shaped with the three sets of weights derived from experiments A, B and C. As the first three experiments have specified time windows, the computed weights for the reward function are specified on the related time window. In the DRL approach for setting D, these weight are applied to the reward function dynamically based on the simulation time. This changes the priority of the agent during an episode. It is expected that the DRL agent has more focus on minimizing order picking costs during the day and less on minimizing tardy orders. In the evening, it is expected that the agent has more priority on minimizing the number of tardy orders than for minimizing the order picking costs.

5.6 Objectives and performance metrics

At last, this section provides the objectives and performance metrics for the experiments. These are used in the benchmark study to make a fair comparison and are defined as follows:

1. **Percentage of tardy orders -** As introduced by Cals et al. (2021), the percentage of tardy orders are defined by the proportion of orders that leave the DeepRele system after their respective cutoff times. In all settings, all orders are processed and once completed, this metric checks what proportion is tardy.

2. **Order picking costs -** Newly introduced for this thesis project, the order picking costs ensure efficient order picking for DeepRele. The order picking costs are defined by the time of a resource that were spent on a particular order. Once an order is batch picked, only the time where a resource is picking that specific order is attributed to the metric. If a pick-by-batch action is performed, the time that is attributed to each individual is likely to be smaller than the time that is attributed to an order that is picked-by-order. Specifically, this metric is calculated to divide the total picking time by the number of orders in a batch. In order to evaluate this problem as a single-objective problem, a scalarization technique is applied. Multiplying both objectives into a single objective is used in this case and this method has been concluded from Chapter 2. If necessary, a certain priority can be given to each objective depending on the use case, this has not been done in this thesis project.

3. **Solutions quality: robustness -** This performance metric is only applicable for the DRL approaches and provides insights on the robustness of the solution. When a DRL agent is trained on a specific setting with a fixed throughput and resource setting, it is the question whether it provides similar behaviour when tested on a different setting. If so, this would greatly increase the applicability of the DRL approach because it does not require to be retrained every time the setting changes. The robustness is defined as the difference between percentage of tardy orders and order picking costs of an agent trained on the specific experiment setting and tested on a different experiment setting.

6. Results

This chapter assesses the performance of the proposed solution methods on several experiments. Each experiment exposes different factors of the solution methods which can be helpful in judging the general performance. The results and analyses of these experiments are presented in this chapter and is structured as follows:

- Section 1: Determine the baseline resource settings for all experiments
- Section 2: Experimental results of heuristics
- Section 3: Experimental results and analysis of DRL approach without reward shaping
- Section 4: Experimental results and analysis of DRL approach with reward shaping
- Section 5: Discussion of all findings

In the first section, the baselines resource settings for all experiments are computed that are used throughout this chapter. In the next section, results of all heuristics are presented in a comprehensive way. The third section contains the experimental results of a trained DRL approach without reward shaping. Additionally, an analysis of the learning behaviour is included, a policy analysis is conducted and the robustness of the learned DRL approach is demonstrated. The fourth section presents the experimental results of a trained DRL approach with reward shaping, includes an analysis of the optimization behaviour and an analysis of the learned policy. Reward shaping is in this case done by framing the problem into a hyper-parameter optimization problem and using Bayesian optimization to solve this problem. Lastly, section 5 presents a discussion on the findings of this chapter.

6.1 Baseline resource settings for all experiments

In Chapter 5, four experiment settings have been defined that are used to benchmark the proposed solution methods. In order to perform a benchmark, a set of parameters need to be selected for the DeepRele simulation model. This set of parameters consists of the number of resources at the picking areas and subsequent work stations. As each setting has a fixed throughput and time window, adjusting the amount of resources has several consequences for the operational performance in terms of tardy orders and order picking costs. This section provides these baseline resource settings based on simulation results of the BOC batching heuristic and the LST sequencing heuristic. Based on a preliminary analysis, this BOC batching heuristics provided stable and accurate behaviour and is therefore chosen to compute the baseline resource settings for all experiment settings. Together with the LST sequencing heuristic from Cals et al. (2021), the resource setting of this combined heuristic is displayed in Table 6.1. Results of this heuristic and all other heuristics are introduced in the next section. As displayed in Table 6.1, the resources are displayed in sequence: Number of PtG pickers - number of GtP shuttles - number of StO resources - number of DtO resources. With a certain amount of orders, the performance of each experiment setting is defined by the two objectives: the percentage of tardy orders and the order picking costs. In scenario D, two shifts have been included that vary in the number of resources. According to the order arrival pattern, the end of a day has the most orders coming in and resources need to be scheduled accordingly. Furthermore, the value in between the brackets represent the standard deviation of each value based on 150 simulated episodes. The results for these heuristics are considered reasonable as verified with domain expert from VI. The percentage of tardy orders is fairly high in settings C and D. The end of a day poses challenges for methods and newly proposed methods

should improve upon this. Utilization levels at each work station are considered when selecting the baseline resource settings. These resource settings are used in bench marking the heuristic results, the results of the DRL approach and the results of Bayesian optimization in combination with the DRL approach. The reason why these specific baseline resource settings are selected is because of the fact these provide representable scenarios in real-world warehousing concept. DeepRele has not yet been implemented in a physical systems but these resource settings and throughput have been used in similar fulfillment concepts.

Setting	Heuristic	Resource settings	Number of orders	Tardy orders	Picking costs
A	BOC	20-20-5-9	6000	3.16 % (1.1)	46.46 (0.8)
B	BOC	20-20-5-9	7000	6.35 % (1.2)	48.87 (0.9)
C	BOC	20-20-5-9	6500	11.92 % (1.5)	45.31 (1.0)
D	BOC	S1: 20-20-7-12 S2: 25-25-8-15	50,000	23.98 % (0.4)	35.38 (0.2)

Table 6.1: Baseline resource settings for the experiment

6.2 Heuristic results

Based on the selected baseline settings, several heuristics are used to provide a basis for the benchmark study. These heuristics are introduced in Chapter 4 and results from these heuristics are presented in Table 6.2. These average results are obtained by simulation per setting for respectively 139, 189, 246, 140 runs and these were computed in section 5.3.

Model	Setting A		Setting B		Setting C		Setting D	
	Tardy orders (%)	Picking costs	Tardy orders (%)	Picking costs	Tardy orders (%)	Picking costs	Tardy orders (%)	Picking costs
BOC	**3.17 (1.1)**	**46.80 (0.8)**	**6.24 (1.4)**	49.00 (0.9)	**11.80 (1.8)**	**45.42 (0.9)**	**23.91 (0.3)**	**35.40 (0.2)**
LST	17.6 (0.5)	65.12 (0.7)	18.7 (0.6)	68.83 (0.8)	21.15 (0.9)	67.21 (0.9)	25.76 (0.3)	53.34 (0.3)
GVNS	6.82 (0.2)	47.41 (0.9)	13.05 (0.5)	**48.92 (0.9)**	18.78 (1.5)	45.48 (1.0)	26.46 (0.2)	35.52 (0.2)

Table 6.2: Heuristic results for all settings

As displayed in Table 6.2, the BOC model obtained best results in almost every setting on percentage of tardy orders and picking costs. This BOC model strikes a balance between batching efficiently using a similarity coefficient and determining whether a batch will be processed by the system on time. In all experiment settings, the GRASP model also provides fairly good results. This model is however much more computational expensive due to its local search principles. At every step, neighboring structures are explored and this makes it expensive to compute. Therefore, for the coming comparison with the proposed DRL approach, the BOC model is used.

6.3 DRL results without reward shaping

In this section, the results of the trained DRL approach are presented. This DRL approach is trained on two objectives: percentage of tardy orders and order picking costs. First, the experimental results are presented where a comparison is made with the heuristic results. After this, the learning behaviour of the DRL approach for every experiment setting is analyzed. The third sub section provides an analysis of the learned policies in terms of operational consequences. Lastly, a robustness analysis is included to demonstrate generalization behaviour of the DRL approaches.

6.3.1 Experimental results

The trained DRL approaches without reward shaping have been tested on four settings. In this testing process, the DRL agent greedily predicts an action based on it's trained policy and no exploration occurs anymore. The results of these agents are displayed in Table 6.3 and Figure 6.1 together with the heuristic results from the earlier section. The values between brackets represent the standard deviation. These average results are obtained by simulation per setting for respectively 228, 554, 168, 172 runs and these were computed in section 5.3. The rationale for this experiment is to assess the performance of a DRL approach without reward shaping.

Model	Setting A		Setting B		Setting C		Setting D	
	Tardy orders (%)	Picking costs	Tardy orders (%)	Picking costs	Tardy orders (%)	Picking costs	Tardy orders (%)	Picking costs
DRL	**2.03 (0.8)**	48.65 (0.8)	**4.18 (0.8)**	50.70 (1.0)	**6.65 (0.8)**	**42.22 (0.7)**	**13.39 (0.4)**	**34.15 (0.2)**
BOC	3.17 (1.1)	**46.80 (0.8)**	6.24 (1.4)	49.00 (0.9)	11.80 (1.8)	45.42 (0.9)	23.91 (0.3)	35.40 (0.2)
LST	17.6 (0.5)	65.12 (0.7)	18.7 (0.6)	68.83 (0.8)	21.15 (0.9)	67.21 (0.9)	25.76 (0.3)	53.34 (0.3)
GVNS	6.82 (0.2)	47.41 (0.9)	13.05 (0.5)	**48.92 (0.9)**	18.78 (1.5)	45.48 (1.0)	26.46 (0.2)	35.52 (0.2)

Table 6.3: Experimental results of the best performing heuristic and DRL approach

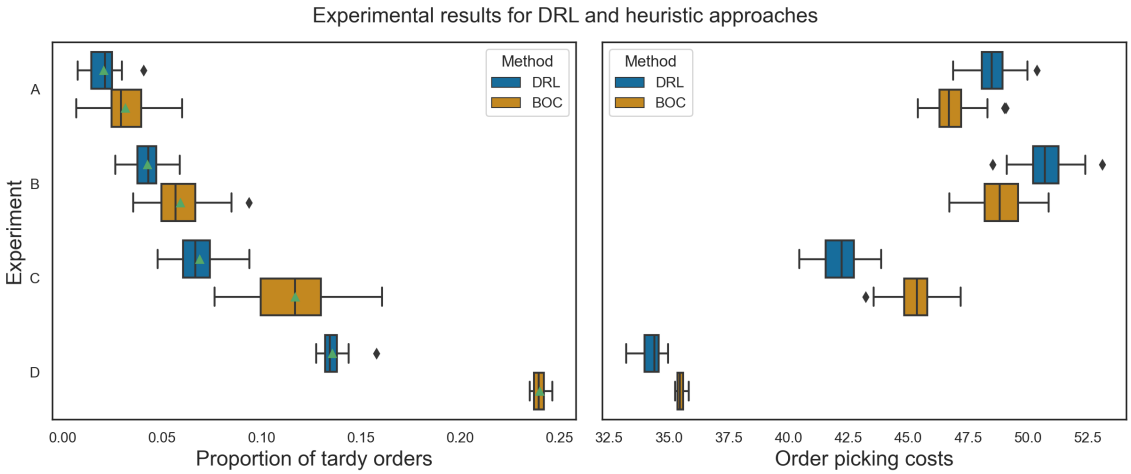


Figure 6.1: Box plots for experimental results of the DRL approach and BOC heuristic

The results of all of the heuristics, have been added to Table 6.3 to make a comparison in results. For every setting, the DRL approach outperforms the best heuristic based on the percentage of tardy orders. An independent sample t-test is used to test whether the unknown populations means of the two methods are equal or not. For these settings, there is statistical evidence that the mean performance of the two methods is not equal in terms of both the percentage of tardy orders as the picking costs. This has been tested with a p-value of 0.01 (Student, 1908). It is assumed that the DRL algorithm performs better as it is able to learn the dependencies between orders and their characteristics. Within setting A, the percentage of tardy orders is lower than for the heuristic method but the order picking costs are however higher. Within this setting, the average pick-by-batch ratio was lower (63%) than the pick-by-batch ratio of the BOC heuristic (71%) which indicates less efficient picking for the DRL approach. This observation also applies to setting B where the percentage of tardy orders is lower but the picking costs are higher (67% vs. 75%). For setting C and D, both the tardy orders and picking costs are lower than the BOC heuristic. Both DRL approaches reduce the number of tardy orders without letting the order picking costs increase. A deeper analysis on the policies is given in the next section 6.3.3.

In order to present the complexity of solving a multi-objective variant of the OOBSP, Table 6.4 displays the results obtained by a DRL approach that has been trained in order to only minimize a single objective: percentage of tardy orders. This Table only displays the results for setting B, whereas the other settings present similar patterns in terms of the relation between optimizing the two objectives. It can be denoted that when this single-objective DRL approach solely focuses on minimizing the percentage of tardy orders, the picking costs increase significantly. The multi-objective DRL approach finds some trade-off in between the optimization boundaries. There seems to be a negative relationship between the two objectives.

Model	Setting B	
	Tardy orders (%)	Picking costs
Single-objective DRL	3.81 (0.3)	58.51 (0.4)
Multi-objective DRL	4.18 (0.7)	50.70 (0.9)

Table 6.4: Results of DRL approaches trained on single and multiple objectives for setting B

6.3.2 Analysis of learning behaviour

The rationale for this analysis is to observe whether the DRL approach demonstrates stable behaviour during learning. Figure 6.2 displays the learning behaviour of the agents for setting A, B and C during the training process. The colored lines represent the mean over all training steps and the translucent lines represent the actual measured values. Setting D is not included in this graph as the episodic reward was fairly different and does not fit with the other settings. The results of these three settings show that the episodic reward displayed in the top left plot stabilizes after approximately 2M steps which implies that the PPO agent does not learn better strategies. The large variance comes from the exploration part that is included during training.

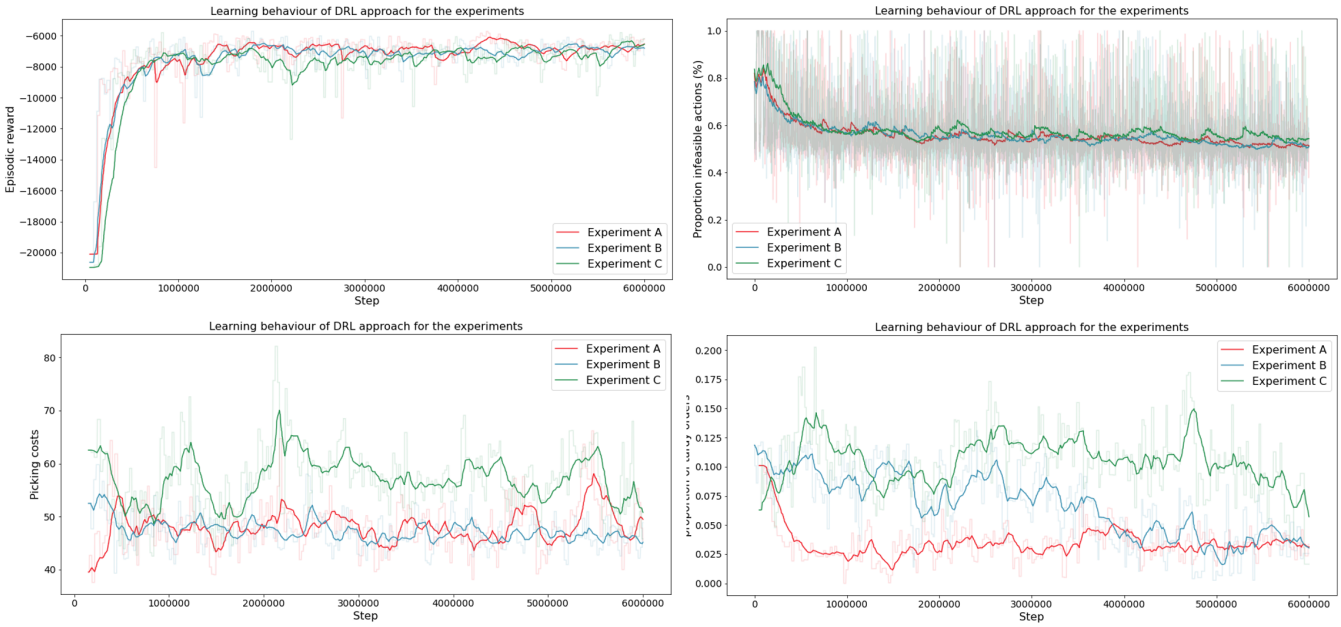


Figure 6.2: Learning behaviour of PPO agent in terms of episodic reward, infeasible actions, picking costs and tardy orders

As shown in the reward function, the reward can be attributed to three components: the number of

tardy orders, the order picking costs and making infeasible actions. Starting with making feasible actions, the top right plot display the infeasible action rate over all training steps. Similarly for all settings, this stabilizes at 55%. The reason this is quite similar for all settings is the fact that the problem of predicting feasible action is for each setting the same. A different throughput, resource setting or time window will not have impact on that. Furthermore, the rate at which the agent stabilizes is quite high. However, during the training process, the DRL agent makes random actions to explore the solution space. When exploiting its policy in the testing process, in 90% of the time, the agent predicts feasible actions. This has been observed when computing the experimental results. This exploration behaviour also explains the huge variance around the mean infeasible action proportion. Next, the distribution of pickings costs are displayed in the lower left plot of Figure 6.2. This displays fairly constant behaviour for setting A and B, and a little more unstable behaviour for setting C. As setting C has a time window of 22:00 - 24:00, a lot of cutoff moments for all orders are located in this latter part of the day, and the agent is figuring out whether to predict a pick-by-batch or pick-by-order action. On the one side, the system requires a certain throughput to process all orders (by performing pick-by-batch action) but also needs to make sure that some individual orders are processed on time (by performing pick-by-order action). The lower right plot of Figure 6.2 displays the distribution of tardy orders over the training process. Here, three different behaviours can be identified. Setting A stabilizes fairly quickly after approximately 2M steps. In contrary, setting B takes a lot longer to present stable behaviour at around 5M steps. Looking at setting C, this distribution of tardy orders over the training process is quit different. First of all, there are more tardy orders (as confirmed by the baseline setting), but secondly, also the distribution is quite uncertain. This may be explained by the fact that setting C assumes a processing window of 22:00 - 24:00 where a lot of cutoff moments are situated because this is the end of a day.

6.3.3 Policy analysis

This section tries to infer trained policies from the DRL approaches. The rationale for this analysis is to observe whether the trained DRL approach without reward shaping makes appropriate actions and what the consequences of these actions are on operational performance. As the results indicate superior performance compared to heuristic results, there should be a different approach to the OOBSP. In order to capture the learned behaviour of the policy, first, the pick-by-batch ratio over time for the experiment settings is displayed in Figure 6.3. Providing a pick-by-batch or pick-by-order action is nevertheless the main means of action of the DRL approach.

Within the top part of Figure 6.3, the average pick-by-batch ratio for PtG storage is given for experiment settings A, B and C. Within this storage area, the online order batching and sequencing operations makes the most impact. Setting A has a time window between 15:00 and 17:00 and has a single cutoff moment at 17:00 indicated by the red vertical lines. The pick-by-batch proportion for this setting displays some volatile behaviour where just before the cutoff moment at 17:00, the pick-by-batch ratio dips indicating a larger proportion of pick-by-order actions in order to process orders with a shorter lead time. For setting B, the time window is defined between 20:00 and 22:00 and the proportion of pick-by-batch actions displays a decreasing trend towards 22:00. With two peak during 20:30 and 21:30, this can indicate efficient order picking (large proportion of pick-by-batch actions) during a hour and close to a cutoff moment, switch to a more pick-by-order based policy. Setting C represents the last part of the day and displays different behaviour. A dip in the pick-by-batch ratio at around 22:30 and after this, a steady increase. As this last part of the day is both busy and a lot of orders have an imminent cutoff time, the agent seeks some trade off between ensuring enough throughput by making pick-by-batch actions (efficient order picking) and minimizing the number of tardy orders by making pick-by-order action that have lower lead times. After 23:00, the pick-by-batch ratio starts decreasing until 24:00. In the middle part of Figure 6.3, setting D includes a time window between 10:00 and 24:00. Initially, the proportion of pick-by-batch actions increases until around 16:00. After this moment, this proportion starts decreasing.

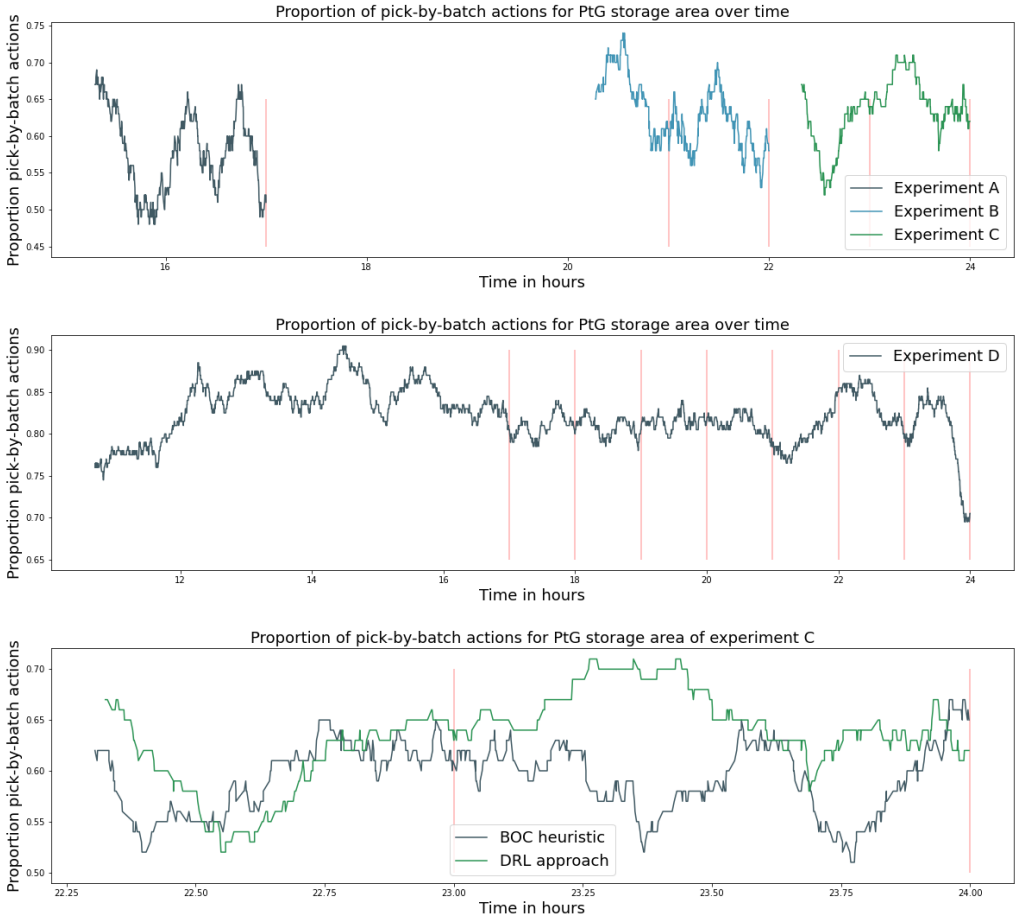


Figure 6.3: Distribution of pick-by-batch actions for the PtG storage area

The end of this time window provides interesting behaviour where after 21:00 the proportion of pick-by-batch actions increases heavily after which near the end of 24:00, it decreases heavily again. This pattern can be recognized from setting C. A possible explanation for this behaviour is the trade off between efficient order picking and short lead times of processed orders. As the end of a day is a relatively busy period, the system needs to guarantee a certain throughput to process all orders. However, some orders cannot be included within a batch as they have imminent cutoff moments that are not satisfied due to the long lead times of batch picking. In order to infer differences in policies between the DRL approach and the BOC heuristic, the pick-by-batch ratio of setting C is displayed in the lower part of Figure 6.3 for both approaches. Based on this ratio over time, one can see that the main difference is in the last hour of the setting where the BOC heuristics demonstrates a low pick-by-batch ratio which increases near 24:00. The DRL approach on the other side displays a higher pick-by-batch ratio until 23:30 and provides a lower ratio in the last half hour of the setting. As the lead times of batches are significantly longer than for single orders, increasing this ratio just before a cutoff moment inevitably leads to an increase in tardy orders. This could also be the explanation for the better performance of the DRL approaches. The increase in picking costs of the BOC heuristic for setting C in table 6.3 could be explained by the pattern of pick-by-batch ratio after 23:00 which is significantly lower than that of the DRL approach. This lower pick-by-batch ratio causes order picking costs to be higher as the order pickers perform less efficient picking.

Secondly, the agent can of course determine the sequence of order categories to pick. This can have

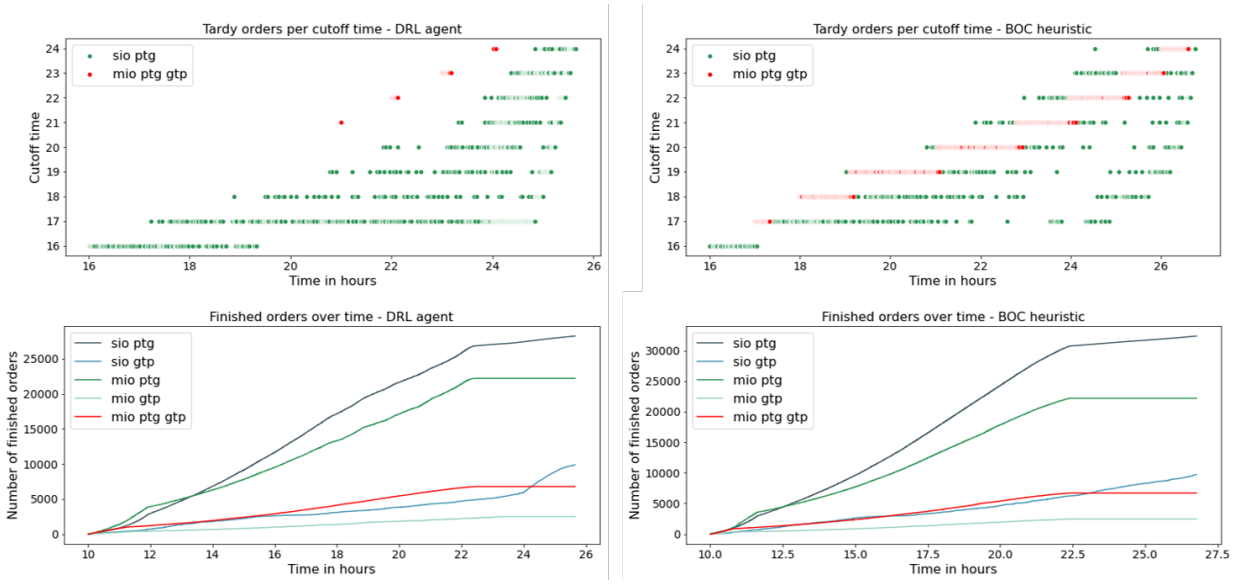


Figure 6.4: Progress of finished orders and the distribution of tardy orders for setting D

influence on the types of tardy orders that are obtained at the end of the episode. The distribution of tardy orders after an episode for experiment setting D is displayed in the top part of figure 6.4 for both the DRL approach as the BOC heuristic. As the DRL outperforms the BOC heuristic, the DRL approach finishes earlier than the BOC heuristic and so, has less tardy orders. Both approaches have two order categories that have tardy orders after finishing an episode: sio-ptg and mio-ptg-gtp. The first category consists of single item orders in storage location Person to Goods. This second order category consists of multi item orders that are stored in both Person-to-Good storage and Goods-to-Person storage. Besides this, the DRL approach has relatively less tardy mio-ptg-gtp orders than the BOC heuristic. An explanation for this behaviour might be that the DRL focuses more on processing mio-ptg-gtp orders while the BOC heuristic focuses more on processing sio-ptg orders. The lower two plots represent the progress of different categories of orders during an episode. Each line represents an order category that is processed and based on this, the sequencing policy can be inferred. The DRL approach has a slightly larger priority for the order category mio-ptg compared to the BOC heuristic as the directional coefficient is larger. On the other side, the BOC heuristic focuses more on mio-gtp order category than the DRL approach. Furthermore, both approaches focus on similarly on processing the largest order category: sio-ptg. Results for the other experiment settings are similar to these described above and are therefore omitted.

Lastly, more or less as a consequence from making either a pick-by-batch action or a pick-by-order action and the learned sequencing policy, Figure 6.5 displays the distribution of PtG picking costs over the time because these are impacted the most when addressing the OOBSP. A pick-by-order action has a higher total picking costs as the order picker is less efficient when performing a pick-by-order action and therefore spending relatively more time on an individual order. On the other side, a pick-by-batch action has lower picking costs as the order picker picks more efficiently and thus, relatively less time is spent on a single order. The average PtG picking costs for setting A behaves quite linearly. This could be related to the characteristics of the setting itself as it is conducted in a time window between 15:00 and 17:00 where the order arrival pattern is steady. For setting B, the average order picking costs for the PtG storage area displays an upward trend near the end of the setting. As these order picking costs is mostly a consequence from the pick-by-order or pick-by-batch decision, the pick-by-batch ratio declines over time and thus, the order picking costs rises (this can be confirmed by figure 6.3). The high order picking cost at 21:00 can

be explained by the cutoff moment at this time where the agent decides to perform 'expensive' pick-by-order action before this cutoff moment in order to process certain orders that have a high probability of becoming tardy. After 21:00, the agent decides to perform 'cheaper' pick-by-batch actions. The order picking costs for the GtP storage area decline near the end of the setting, indicating a larger pick-by-batch ratio. As experiment setting C is situated at the end of day between 22:00 and 24:00, a similar explanation of the behaviour can be used. Again, the average picking costs for the PtG storage area displays an upward trend near the end of the setting and the average picking costs for the GtP storage area displays a downward trend. Lastly, for setting D that represents an entire day of processing, more or less steady average picking costs for the PtG storage area is displayed. However what can bee seen is that the variation intensifies near the end of the setting. This could be explained by the fact that the agent incorporates different behaviour near cutoff moments at the end of the setting.

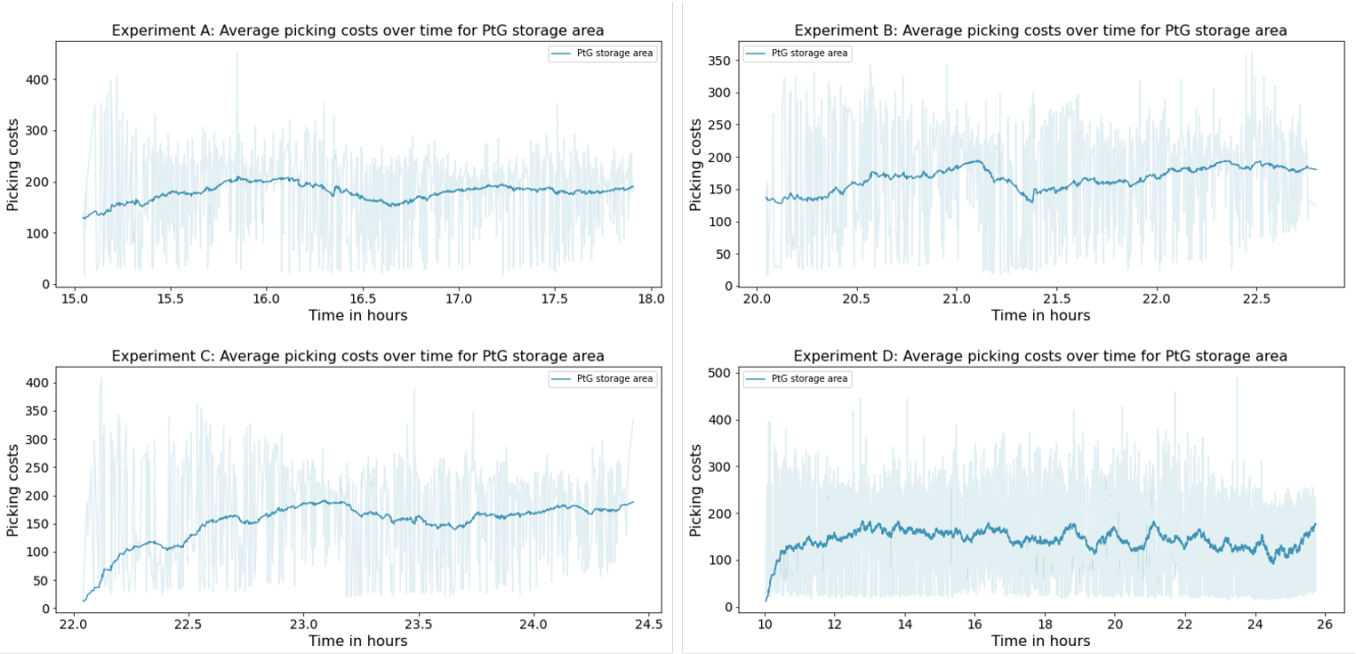


Figure 6.5: Average picking cost during the experiment

6.3.4 Robustness analysis

As mentioned in section 5.6, the proposed solution needs to be evaluated on its robustness to identify generalizing behaviour. This sub section performs a robustness analysis on the pre-trained DRL approaches by two parts.

In the first part, all experiment settings have been adjusted so that 10% more orders require to be processed and a few more resources are added. The argument behind these adjustments is to analyze how the DRL approaches adapt to a slight change in experiment setting. These results have been displayed in Table 6.5 and present similar ratios as in Table 6.3. The objective values have been displayed for the DRL approach and the best performing BOC heuristic. It can be seen that the DRL approach outperforms this BOC heuristic in every setting based on the percentage of tardy orders. For setting A, B and D, the picking costs increased slightly, where it is assumed that this can be attributed to the lower percentage of tardy orders. For DRL approach in setting C, both the percentage of tardy orders and the picking costs is lower than the results of the BOC heuristic. This shows that the trained DRL agents are robust to small throughput and resource changes within their current experiment settings.

Model	Setting A		Setting B		Setting C		Setting D	
	Tardy orders (%)	Picking costs	Tardy orders (%)	Picking costs	Tardy orders (%)	Picking costs	Tardy orders (%)	Picking costs
BOC	1.98 (0.7)	**46.83 (0.5)**	5.91 (1.6)	**48.80 (0.9)**	10.23 (1.8)	45.23 (0.8)	23.83 (0.2)	**35.34 (0.9)**
DRL	**1.23 (0.4)**	48.29 (0.9)	**3.54 (0.4)**	50.65 (0.6)	**6.83 (0.1)**	**41.74 (0.6)**	**15.28 (0.4)**	35.37 (0.5)

Table 6.5: Robustness analysis DRL approach adjusted experiment settings

In the second part, the trained agents are tested on three other experiment settings and the results are captured in Table 6.6. Bold numbers mark the best objective value for a specific setting. For models that were tested on setting A, trained models of setting B and C are fairly close to the result of the model that was actually trained on setting A. The fact that setting A and B are somewhat similar can explain this overlap in performance. However, the other way around, this explanation does not hold as testing setting B using a model that was trained on setting A, yields dramatic results. When testing setting B, it stands out that, model C outperforms model B based on picking costs. This could be explained by the fact that experiment C has a different priority of objectives that works particularly well for the order picking costs when testing for setting B. In the case of testing on setting C, the model that was actually trained on setting C outperformed all other models. The results of testing setting D provides similar results as testing for setting B. Based on the percentage of tardy orders, the model that was trained for this setting specifically outperformed all other models but based on the order picking costs, model C provides better performance. Furthermore, when looking at the results of testing setting D, there is a large difference in performance between model D and the others. Model D captures an entire day, whereas the remaining models capture only a time window of two hours. Because of this difference, it is assumed that there are different dynamics during an entire day of processing than instead a two hour window. And thus, for different settings within a similar order arrival distributions, the DRL approach finds it hard to demonstrate robust behaviour especially in terms of the percentage of tardy orders. Every experiment has different dynamics that are captured in the learned policies and therefore do not generalize that well.

Tested on → / Trained on ↓	Setting A		Setting B		Setting C		Setting D	
	Tardy orders (%)	Picking costs	Tardy orders (%)	Picking costs	Tardy orders (%)	Picking costs	Tardy orders (%)	Picking costs
Setting A	**1.23 (0.4)**	48.29 (0.9)	7.78 (1.3)	47.65 (0.8)	11.03 (1.3)	44.72 (1.0)	23.58 (0.3)	36.06 ((0.2)
Setting B	1.32 (0.5)	47.40 (0.9)	**3.54 (0.4)**	50.56 (0.6)	8.11 (1.2)	46.38 (0.8)	21.55 (0.3)	35.37 (0.2)
Setting C	2.56 (0.7)	**43.70 (0.8)**	6.00 (1.0)	**44.37** (1.0)	**6.83 (0.1)**	**41.74 (0.6)**	20.72 (0.2)	**33.62 (0.1)**
Setting D	4.87 (0.7)	45.57 (0.7)	9.20 (0.8)	46.45 (0.7)	11.30 (0.6)	43.67 (0.4)	**15.28 (0.4)**	35.37 (0.5)

Table 6.6: Robustness analysis DRL approach based on the single objective

6.4 DRL results with reward shaping

This section presents the results of the DRL approaches with reward shaping. This reward shaping process is done by using a Bayesian optimization approach. As mentioned in Chapter 4, this method takes a standard reward function and shapes this dynamically. Each iteration, a new set of weights is applied to this reward function, the agent is trained, tested, and the objective value is observed. With this new information, a new set of weights is applied to the reward function and the iteration continues until some specified maximum of iterations. First, the experimental results are presented and compared to the DRL approaches without reward shaping. Secondly, an analysis is provided on the optimization results. Lastly, a policy analysis is presented that uses a Decision tree to map the learned policy to a set of explainable decision rules. This will help to

explain the learned policy and to eventually get some insights in the dynamics of the OOBSP.

6.4.1 Experimental results

The rationale for this experiment is to assess the performance of a DRL approach with reward shaping compared to a DRL approach without reward shaping. Using the proposed Bayesian optimization technique for reward shaping, an agent was trained and tested for experiment settings A, B and C. Then for setting D, the obtained weights from the previous settings are used for training the agent. During the different time windows of the day, different weight settings are applied to the reward function of the agent for setting D. In this way, the agent's reward function is dynamically shaped according to the moment in time of the day. The results are displayed in Table 6.7 and Figure 6.6. These results are obtained by simulation per setting for respectively 228, 554, 168, 172 runs and these were computed in section 5.3. Within this table, the mean values of both approaches for setting A, B and C are tested using a student t-test. These provide statistical evidence that the means are different from each other with an α of 0.05. As setting D is computational too complex to perform enough iterations and compute the student t-test, statistical evidence is not provided. The DRL approach with reward shaping trained on setting A outperforms the DRL approach without reward shaping on both tardy orders and picking costs. The DRL approach with reward shaping also outperforms the DRL approach without reward shaping in experiment setting B. However for setting C, the DRL approach with reward shaping did not outperform the DRL approach without reward shaping based on order picking costs. Probably, the priority for minimizing the order picking costs of the DRL approach with reward shaping was too low. The results for the DRL approach with reward shaping on setting D outperforms the DRL approach without reward shaping slightly in terms of tardy orders and order picking costs.

Model	Setting A		Setting B		Setting C		Setting D	
	Tardy orders (%)	Picking costs	Tardy orders (%)	Picking costs	Tardy orders (%)	Picking costs	Tardy orders (%)	Picking costs
BOC	3.17 (1.1)	46.8 (0.8)	6.24 (1.4)	49.0 (0.9)	11.8 (1.8)	45.4 (0.9)	23.91 (0.3)	35.40 (0.2)
DRL	2.03 (0.8)	48.65 (0.8)	4.18 (0.8)	50.70 (1.0)	6.65 (0.8)	**42.22 (0.7)**	13.39 (0.4)	34.15 (0.2)
DRL + RS	**1.68 (0.7)**	**43.02 (0.6)**	**2.60 (0.7)**	**46.87 (0.8)**	**5.53 (1.1)**	42.90 (0.6)	**12.40 (0.2)**	**33.96 (0.2)**

Table 6.7: Experimental results of the DRL approach with and without reward shaping (RS)

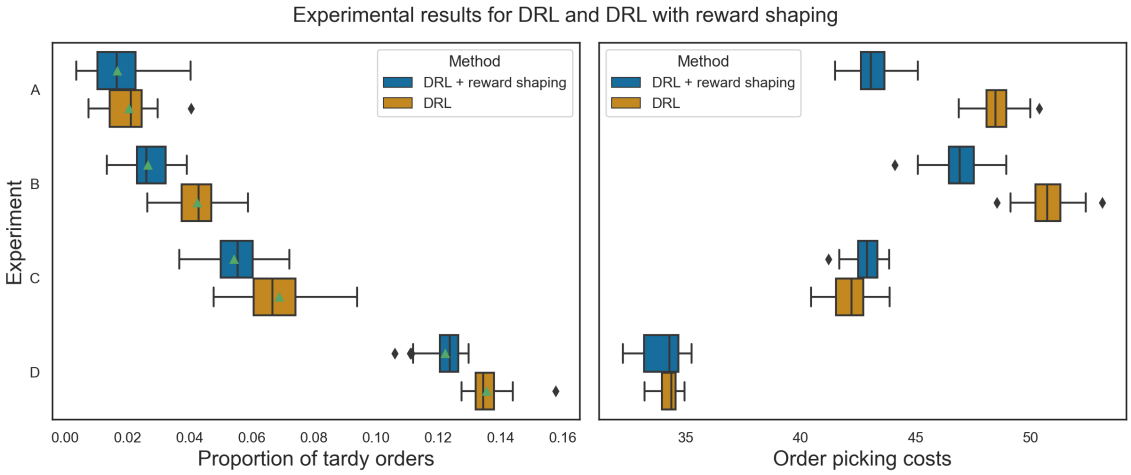


Figure 6.6: Experimental results of the DRL approach with and without reward shaping

6.4.2 Analysis of optimization results

The rationale for this analysis is identify how possible solutions (i.e. sets of weights) are distributed over the solution space. This could yield information about the OOBSP itself. The proposed Bayesian optimization methods iteratively gathers more information about the surrogate function between the weights in the reward function and the resulting objective value of the DRL approach. As this is an iterative process, it is interesting how this surrogate function is fitted and how does that reflect a better performing DRL approach. After 50 iterations, the Bayesian optimization method found the following weights for in the reward functions of all experiments: (0.86, 0.9), (1.34, 0.52), (1.2, 0.45). The weights are computed for each experiment individually and consists out of the weight for tardy orders and the weight for order picking costs respectively. Figure 6.7 displays these findings and provides information about the explored solution space of the two weight parameters and the objective value for setting A, B and C. Setting D is computationally too expensive to perform Bayesian optimization. However, the obtained knowledge from setting A, B and C is used in setting D when re training the DRL approach. For each setting, figure 6.7 indicates objective values within the solution spaces where the goal is to minimize the objective value. Several solutions have been evaluated and are subsequently used to fit a surrogate function and sample new weights. For setting A, the best set of weights are 0.86 for tardy orders and 0.90 for picking costs. For setting B, the best set of weights are 1.34 for tardy orders and 0.52 for picking costs. For setting C, the weight of tardy orders seems to have a slightly larger impact on the objective value than in the other two settings. This could be explained by the fact what within this setting, all orders have their respected cutoff time at the end of the setting and that therefore, the agent focuses on picking these orders in time. For setting C, the best set of weights are 1.20 for tardy orders and a weight of 0.45 for order picking costs. The weight for order picking costs is decreasing over setting A, B, C. This could indicate that order picking costs become less important near the end of a day and that tardy orders becomes more important. For each setting, the Bayesian optimization method converged to some set of weights that provided the best objective value. The trajectories of this was quite different among the experiment settings. From a total of 50 iterations per setting, setting A found the best solution after iteration 29, setting B found the best solution after iteration 8 and setting C found the best solution after 45 iterations.

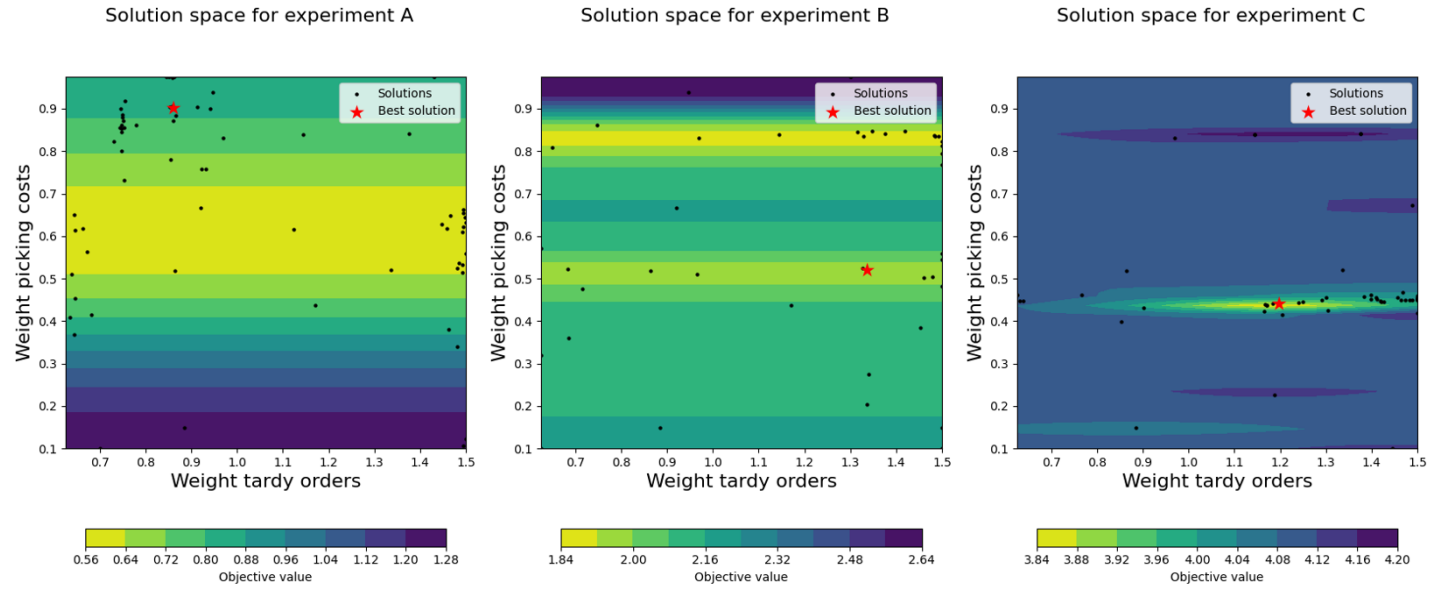


Figure 6.7: The solution spaces for settings A, B and C explored by Bayesian optimization

6.4.3 Policy analysis

This subsection contains a policy analysis wherein the DRL approach with reward shaping is distilled into a decision tree. These decision trees can make learned policies more interpretable and provide explanation for certain predicted actions on given state representations. The work of Che et al. (2016) provides insights in a DRL policy using these decision trees and this is used to provide insights to the DRL approach with reward shaping for settings A and C. The tree for setting A is displayed in Figure 6.8 and the tree for setting C is displayed in Figure 6.9. The rationale for this analysis is to obtain explainable factors that help in understanding the DRL approaches with reward shaping.

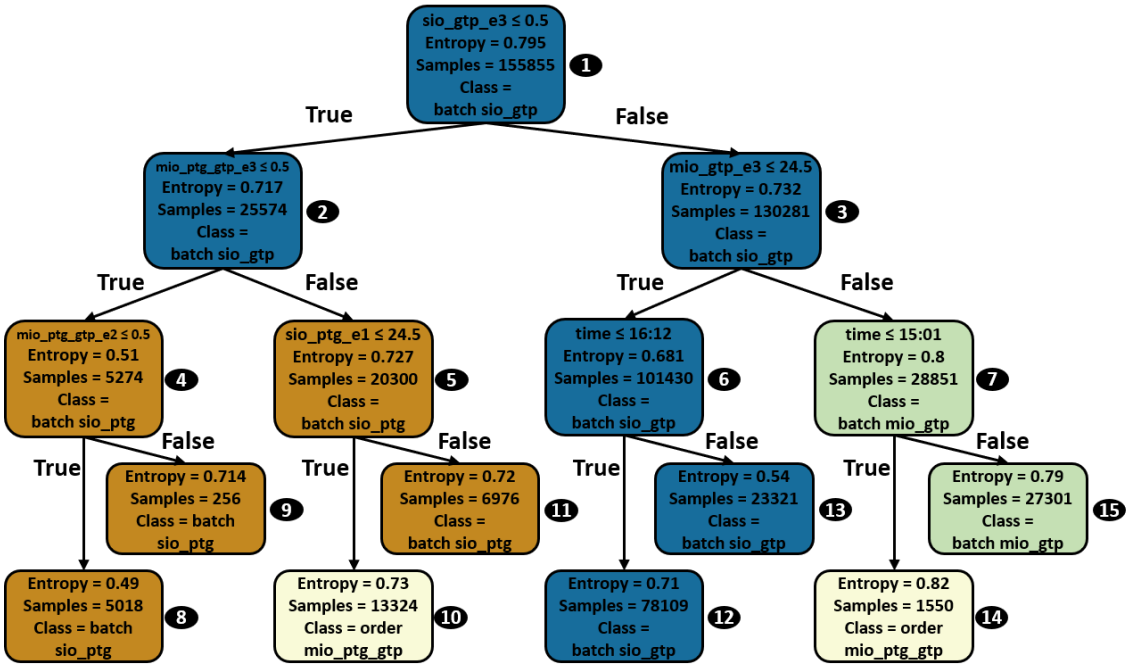


Figure 6.8: Decision Tree trained on DRL agent of setting A

Using the DRL approach with reward shaping trained on experiment setting A, a dataset has been compiled with a state representation of 20 features and 10-class predicted actions. This experiment is conducted between 15:00 and 17:00 and does not contain a lot of cutoff moments. Using this structured dataset, a classification approach is taken and a decision tree is fitted. The dataset is unbalanced where the following list indicates the occurrence per class [7224, 13008, 20441, 59091, 1550, 1778, 9463, 20415, 17601, 5284]. Using a 10-fold cross validation evaluation method, the decision tree obtained an accuracy of 47%. For a 10-class classification problem, this performance is reasonable as the decision tree is only used for inference of the learned policy and a random guess would yield 10% accuracy. Using this decision tree in Figure 6.8, several 'rules' can be inferred. The colors within this figure represent similar predicted actions and the entropy in all nodes refers to the 'disorder' of this node in providing a certain decision rule when classifying the data. First of all, the decision tree makes a split based on whether there are orders within order category sio-ptg-e3. This order category contains orders that are single-item, are stored in the PtG storage area and have a cutoff time that is more than 40 minutes from the current moment. Furthermore in node 2, the decision tree splits the data based on orders within order category mio-ptg-gtp. If there are orders within this order category and the condition in node 5 is true, a pick-by-order action for order category mio-ptg-gtp is predicted. In all other cases, the decision tree predicts a pick-by-batch action for order category sio-ptg (node 4, 8, 9, 5, 11). On the right

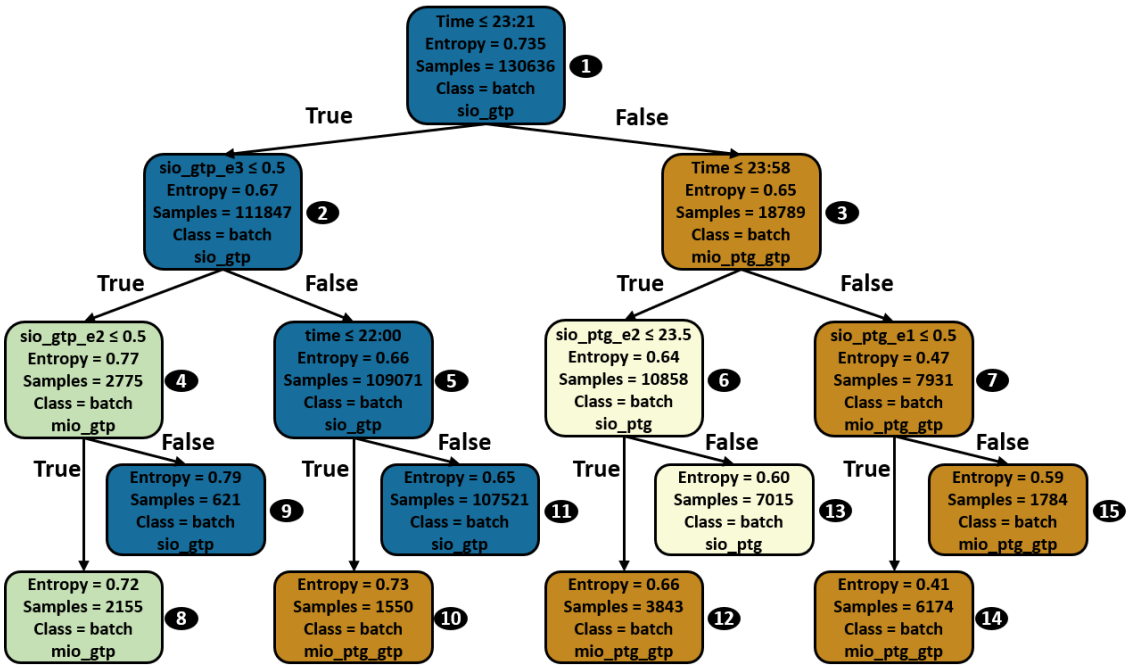


Figure 6.9: Decision Tree trained on DRL agent of setting C

side of figure 6.8, the decision tree makes a split based on order category mio-gtp-e3 where if there are less than 25 orders in this category, the decision tree predicts a pick-by-batch action for order category sio-gtp (node 6, 12, 13). If this is not the case, the model makes a cut in node 7 where based on the simulation time, either a pick-by-batch action for order category mio-gtp is made of a pick-by-order action for order category mio-ptg-gtp.

Experiment C is conducted between 22:00 and 24:00 and contains a lot more cutoff moments for orders arriving within this window. This poses a different dynamic than in experiment A. Using the structured dataset obtained from the policy learned on experiment C, a classification approach is taken and a decision tree is fitted. The dataset is unbalanced where the following list indicates the occurrence per class [2609, 12997, 8097, 60973, 685, 2129, 8644, 11921, 18178, 4403]. Using a 10-fold cross validation evaluation method, the decision tree obtained an accuracy of 56%. For a 10-class classification problem, this performance is reasonable as the decision tree is only used for inference of the learned policy. The decision tree learned on data from experiment C in Figure 6.9, makes a split based on the simulation time in node 1. Setting C has a time window between 22:00 and 24:00 and if a certain state representation has a simulation time that is lower than 23:21, the decision tree guides the prediction to the right side of the tree. This indicates a certain time dependency learned by the DRL approach. After this step, the tree checks whether there are orders within the order category sio-gtp-e3 in node 2. This order category contains orders that are single-item, are stored in the GtP storage area and have a cutoff time that is more than 40 minutes from the current moment. If there aren't any orders within this category, the decision tree prescribes to choose a batching action for mio-gtp orders in node 4. Most of the time, there are orders within this category (node 4) and the tree describes to pick a batch of SiO GtP orders. On the other side of the tree, the simulation time is again used for forming two branches where it is checked whether the current simulation time is lower than 23:58 and higher than 23:21 in node 3. This can infer that the DRL approach has a priority for the left side of the tree as this part is processed first. Furthermore, on this right side of the tree, a split is made on the sio-ptg-e2 order category in node 6, where if there are less than 24 orders within this category, the tree describes to pick a batch of MiO orders with storage areas PtG and GtP in node 12. With some of the decision

of the decision tree explained, some dependency can be identified based on simulation time and the number of orders within order categories. Again, this decision tree only reflects a small part of the learned policy but confirms some of the dependencies. Comparing decision tree A and C with each other, the main difference is that decision tree C has a time dependency that plays an important role in prediction the class actions. The learned policy for experiment C appears to be more sensitive for the time aspect as there are a lot more cutoff moments within the time window.

Both decision trees try to infer a piece of logic from the learned policies of the DRL approaches. These decision trees are subsequently used as a heuristic for solving the OOBSP for both experiment setting A as C and results are displayed in Table 6.8. These results were obtained by using the described decision trees in simulating the OOBSP for 228 and 168 runs respectively. As the decision trees are used as an approximation to the learned DRL approaches with reward shaping, results will not match the performance. However, in terms of the percentage of tardy orders and order picking costs for experiment setting A and C, the distilled decision tree outperforms the BOC heuristic significantly. A student t-test has been used to provide evidence for these statements where a $\alpha = 0.05$ has been used. This indicates that the distilled decision tree has learned some useful and explainable decision rules from the DRL approaches with reward shaping. This last experiment provides useful information on the succeeding heuristics derived from a DRL approach that demonstrate comparable performance with traditional heuristics for the OOBSP. When better approximation algorithms can be developed, the performance of succeeding heuristics will likely increase as well.

	Setting A		Setting C	
Model	Tardy orders (%)	Picking costs	Tardy orders (%)	Picking costs
BOC	3.17 (1.1)	46.8 (0.8)	11.8 (1.8)	45.4 (0.9)
DRL	2.03 (0.8)	48.65 (0.8)	6.65 (0.8)	**42.22 (0.7)**
DRL + RS	**1.68 (0.7)**	**43.02 (0.6)**	**5.53 (1.1)**	42.90 (0.6)
Distilled DT	3.06 (0.9)	45.59 (0.7)	9.24 (1.0)	44.84 (0.7)

Table 6.8: Experimental results of the decision tree heuristic

6.5 Discussion

As this Chapter presents a lot of results from all conducted experiments, this section gives a short recap on the key findings. In the first section, the baseline resource settings are determined using the BOC heuristic. Using these baseline settings, the results for the other heuristics could be computed in order to provide a comparison with the proposed solution methods.

In the second section, the DRL approach without reward shaping is tested, and the results are compared with the heuristics. In every experiment setting, the DRL approach outperformed the heuristics based on the percentage of tardy orders. However, the BOC heuristic outperformed the DRL approach without reward shaping in some settings based on the order picking costs. This confirms the complexity of shaping the reward function to address the multi-objective problem setting. After this, the learning behaviour of the DRL approaches without reward shaping have been examined and after a certain amount of training steps, the DRL approach demonstrates stable behaviour. Additionally, some comparisons have been done with an DRL approach without reward shaping that was trained on a single-objective variant of the OOBSP. This shows that there is a delicate trade-off between optimizing the two objectives. Furthermore, a policy analysis

demonstrates “intelligent” behaviour near cutoff moments where the pick-by-batch ratio decreases. After these moments, this ratio increases again in order to focus on minimizing order picking costs. A robustness analysis for the learned DRL approaches have been included that demonstrate robustness on small changes to the experiment but presents problems when the principles of the experiment settings are changed.

In the third section, the DRL approach with reward shaping is tested, and the results are compared with the heuristics and the DRL approach without reward shaping. For this reward shaping process, a Bayesian Optimization technique is used that addresses this problem as a hyper-parameter optimization problem. This new DRL approach with reward shaping outperforms both the heuristics as the DRL approach without reward shaping. The learned weights that were obtained from the Bayesian optimization technique confirm the expectation where a potential solution method should focus on minimizing order picking during the day and focus on minimizing tardy orders at the end of the day.

Using this DRL approach with reward shaping, a decision tree has been used to infer certain decision rules from this policy. The main findings were that in some state representations, the DRL approach with reward shaping uses the point in time to make a certain picking decision and has a priority for some of the order categories. This time dependency indicates the change in focus from minimizing order picking costs to minimizing tardy orders.

7. Conclusion & recommendation

This thesis project is a study on solving an online order batching and sequencing problem (OOBSP) in a warehousing concept of Vanderlande called DeepRele. The main objective of this work is to investigate to what extend deep reinforcement learning can solve a multi-objective variant of the online order batching and sequencing problem with real-world characteristics. In order to do so, the literature review provided insights in traditional methods, DRL based methods and methods that can optimize DRL approaches. These insights have been used to design a DRL approach that can solve the OOBSP with real-world characteristics. Within these real-world characteristics, there is a multi-objective variant of the OOBSP, large instances (5000 orders per hour) and changing dynamics in terms of order arrival patterns. In this chapter, the conclusion of this thesis project is presented and recommendations on further research are provided.

7.1 Conclusion

Traditional methods find it difficult to cope with an ever changing environment and increasing instance sizes of the OOBSP in DeepRele. Together with the findings of Cals et al. (2021), a DRL approach for this problem is proposed. In order to cope with these aforementioned challenges, the DRL approach has been designed accordingly. A more dynamic reward function to capture the multi-objective aspect of the problem, a smaller action space for the DRL agent to reduce the computational complexity and re-framed to problem into an online order batching and sequencing problem. Furthermore, the DRL approach uses at its base a Proximal Policy Optimization model which strikes a favorable balance between sample efficiency, simplicity and computational efficiency. Besides this, an efficient simulation model is developed to mimic the dynamics of a DeepRele warehousing system. This new DRL approach was used in a benchmark study where it was compared with a set of three heuristics: LST, BOC and GVNS. In this benchmark study, a scalarization technique has been used to transform the multi-objective problem to a single-objective problem. In each of the experiments in this benchmark study, the DRL approach outperformed the heuristics. The percentage of tardy orders was in all experiments significantly lower for the DRL approach but the order picking costs remained similar to the heuristic approach. This pattern confirms the complexity of solving multi-objective problems where objectives have a negative relationship. In order to infer some "intelligence" of the learned policy, a policy analysis have been provided. This concludes that the DRL approach demonstrates different behaviour in front of cutoff moments. Before these moments, the pick-by-batch ratio drops where more pick-by-order actions are made to ensure that picking orders have low lead times. This ratio increases again after such a cutoff moment. As a results, the order picking costs follow this pattern as pick-by-batch actions are less "expensive". These DRL approaches have been trained on specific instances and a robustness analysis have been performed to observe generalization of these approaches. These approaches are robust to small changes (10%) in throughput and resource capacities but find it hard generalize when a complete different experiment is presented.

In order to further address the challenges that come a long with a multi-objective variant of the online order batching and sequencing problem with real-world characteristics, a hyper-parameter optimization methods is proposed to dynamically shape the reward function of the DRL approach. In real-world environments, it is import to have a reward function that is aligned with the objectives and that can cope with system dynamics. A Bayesian optimization method is used to optimize a set of weights that is applied to the reward function of the DRL approach. For every experiment, a new set of weights is obtained. These weights provide evidence that the DRL approaches have a larger priority for minimizing order picking costs during the day and during the end of the day, have a larger priority for minimizing the percentage of tardy orders. With these new optimized

DRL approaches, a benchmark study is performed compared to the "vanilla" DRL approach from the previous section. This optimized DRL approach outperformed the vanilla approach on both objectives: percentage of tardy order and order picking costs. Similar to the previous part, a policy analysis is conducted to infer some kind of "intelligence" using a decision tree. This decision tree maps a part of the learned policy to some feature-based decision rules and gives an indication where the DRL approach bases its decision. This decision tree provides a time dependency for the DRL approaches where it alters its decision upon. This confirms the findings of the Bayesian optimization approach, where different objectives have a different priority over time.

Overall, the DRL approaches presented in this project provide a novel and exiting method for solving the online order batching and sequencing problem with real-world characteristics. These DRL approaches demonstrate intelligent policies that cope with a multi-objective variant of the problem, large instances and dynamic environments. This Bayesian optimization approach provides a technique that is able to deal with multi-objective problems and changing environments and can contribute to current RL projects solving optimization problems. Compared to other OOBSP heuristics, the DRL approach presents better performance and makes better use of the available data. Using this available data, the DRL approach provides a great potential in incorporating this algorithm into Vanderlande's Warehouse Management Systems. Furthermore, the DRL approach allows for a preference in the ratio of objectives when optimizing the problem.

7.2 Limitations and recommendations

This thesis project served as a experiment to investigate in what extend deep reinforcement learning can solve a multi-objective variant of the online order batching and sequencing problem with real-world characteristics. This work provides a proof of concept for a DRL approach but due to the available time frame of this thesis, several components and challenges were not included and can provide directions for further research. With this, Vanderlande is advised the following:

1. First of all, the simulation that has been used in this thesis project tries to reflect the dynamics of the DeepRele warehousing concept at a certain level of detail. Between this current moment and implementing the DRL approach in a physical warehousing concept, there are several steps to take. One of these steps should address the level of detail of the simulation model by providing a more detailed simulation model that nearly replicates real-world systems. The current level of detail provides a good trade-off between computational complexity and the amount of incorporated system dynamics.

2. Secondly, exact storage locations of individual items are not used in computing the picking times. This information might be useful in order to perform the online order batching and sequencing operation. In relation to this, this work does not address any routing strategy when picking orders. This has impact on the eventual operational performance and before implementing a DRL approach and should be taken into account.

3. Thirdly, in order to industrialize this solution, the learned policy should be converted to an explainable heuristic model. This work tried to infer some of the learned logic into an explainable model that subsequently demonstrated good performance compared to traditional heuristics. Recent advancements in Explainable A.I. (XAI) include different techniques to make Deep Neural Networks (DNN) more explainable and improve the findings of this work.

4. Lastly, there are some steps between this moment and implementing the DRL approaches in physical systems. Bringing reinforcement learning to production, requires a significant amount of effort as indicated by the maturity of the field. It is considered difficult to bring DRL approaches in production, monitor these approaches and periodically retrain these approaches. The domain of Machine Learning operations (ML-Ops) could provide tools in supporting this process.

Bibliography

Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G. S., Davis, A., Dean, J., Devin, M., Ghemawat, S., Goodfellow, I., Harp, A., Irving, G., Isard, M., Jia, Y., Jozefowicz, R., Kaiser, L., Kudlur, M., Levenberg, J., Mané, D., Monga, R., Moore, S., Murray, D., Olah, C., Schuster, M., Shlens, J., Steiner, B., Sutskever, I., Talwar, K., Tucker, P., Vanhoucke, V., Vasudevan, V., Viégas, F., Vinyals, O., Warden, P., Wattenberg, M., Wicke, M., Yu, Y., and Zheng, X. (2015). TensorFlow: Large-scale machine learning on heterogeneous systems. Software available from tensorflow.org. 52

Ardjmand, E., Shakeri, H., Singh, M., and Bajgiran, O. S. (2018). Minimizing order picking makespan with multiple pickers in a wave picking warehouse. *International Journal of Production Economics*, 206:169–183. 11, 12

Armstrong, R. D., Cook, W. D., and Saipe, A. L. (1979). Optimal batching in a semi-automated order picking system. *Journal of the Operational Research Society*, 30(8):711–720. 6, 7

Arulkumaran, K., Deisenroth, M. P., Brundage, M., and Bharath, A. A. (2017). A brief survey of deep reinforcement learning. *arXiv preprint arXiv:1708.05866*. 17

Badia, A. P., Piot, B., Kapturowski, S., Sprechmann, P., Vitvitskyi, A., Guo, D., and Blundell, C. (2020). Agent57: Outperforming the atari human benchmark. 16

Bagchi, U., Chang, Y.-L., and Sullivan, R. S. (1987). Minimizing absolute and squared deviations of completion times with different earliness and tardiness penalties and a common due date. *Naval Research Logistics (NRL)*, 34(5):739–751. 8

Bartholdi III, J. and Hackman, S. (2018). Warehouse & distribution science: release 0.96. atlanta, ga: The supply chain and logistics institute, school of industrial and systems engineering, georgia institute of technology. 1

Boon, M. and Boor, M. (2020). Stochastic simulation using python from department of mathematics and computer science, eindhoven university of technology. 37, 50

Bozer, Y. A. and Kile, J. W. (2008). Order batching in walk-and-pick order picking systems. *International Journal of Production Research*, 46(7):1887–1909. 6, 7

Bustillo, M., Menéndez, B., Pardo, E. G., and Duarte, A. (2015). An algorithm for batching, sequencing and picking operations in a warehouse. In *2015 international conference on industrial engineering and systems management (iesm)*, pages 842–849. IEEE. iv, 11, 12, 44, 45

Cals, B., Zhang, Y., Dijkman, R., et al. (2021). Solving the online batching problem using deep reinforcement learning. *Computers & Industrial Engineering*, page 107221. iii, 1, 2, 4, 21, 29, 30, 31, 33, 34, 35, 36, 37, 40, 41, 43, 45, 46, 51, 53, 54, 69

Caron, F., Marchet, G., and Perego, A. (1998). Routing policies and coi-based storage policies in picker-to-part systems. *International journal of production research*, 36(3):713–732. 1

Che, H., Bai, Z., Zuo, R., and Li, H. (2020). A deep reinforcement learning approach to the optimization of data center task scheduling. *Complexity*, 2020. 2, 19, 21

Che, Z., Purushotham, S., Khemani, R., and Liu, Y. (2016). Interpretable deep models for icu outcome prediction. In *AMIA annual symposium proceedings*, volume 2016, page 371. American Medical Informatics Association. 26, 65

Chen, T.-L., Cheng, C.-Y., Chen, Y.-Y., and Chan, L.-K. (2015). An efficient hybrid algorithm for integrated order batching, sequencing and routing problem. *International Journal of Production Economics*, 159:158–167. viii, 5

Chiang, H.-T. L., Faust, A., Fiser, M., and Francis, A. (2019). Learning navigation behaviors end-to-end with autorl. *IEEE Robotics and Automation Letters*, 4(2):2007–2014. 25

Claesen, M. and De Moor, B. (2015). Hyperparameter search in machine learning. *arXiv preprint arXiv:1502.02127*. 25

De Koster, R., Le-Duc, T., and Roodbergen, K. J. (2007). Design and control of warehouse order picking: A literature review. *European journal of operational research*, 182(2):481–501. 7

de Oliveira da Costa, P. and Zhang, Y. (2021). Learning 2-opt local search from heuristics as expert demonstrations. *arXiv preprint*. 39

Dong, H., Ding, Z., and Zhang, S. (2020). *Deep Reinforcement Learning: Fundamentals, Research and Applications*. Springer Nature. 15

Elhara, O., Varelas, K., Nguyen, D., Tusar, T., Brockhoff, D., Hansen, N., and Auger, A. (2019). Coco: the large scale black-box optimization benchmarking (bbob-largescale) test suite. *arXiv preprint arXiv:1903.06396*. 25

Elsayed, E. (1991). Order sequencing in automated storage/retrieval systems with due dates. In *Material Handling'90*, pages 245–267. Springer. 7, 12

Elsayed, E., Lee, M., Kim, S., and Scherer, E. (1993). Sequencing and batching procedures for minimizing earliness and tardiness penalty of order retrievals. *International Journal of Production Research*, 31(3):727–738. 7, 8, 12

Elsayed, E. and Lee, M.-K. (1996). Order processing in automated storage/retrieval systems with due dates. *IIE transactions*, 28(7):567–577. 8, 12

Frosst, N. and Hinton, G. (2017). Distilling a neural network into a soft decision tree. *arXiv preprint arXiv:1711.09784*. 26

Gademann, N. and Velde, S. (2005). Order batching to minimize total travel time in a parallel-aisle warehouse. *IIE transactions*, 37(1):63–75. 5, 6, 7

Gao, Y., Xu, H., Lin, J., Yu, F., Levine, S., and Darrell, T. (2018). Reinforcement learning from imperfect demonstrations. *arXiv preprint arXiv:1802.05313*. 22

Genton, M. G. (2001). Classes of kernels for machine learning: a statistics perspective. *Journal of machine learning research*, 2(Dec):299–312. 25

Hamberg, R. and Verriet, J. (2012). *Automation in warehouse development*. Springer. 1

Hansen, N. (2006). *The CMA Evolution Strategy: A Comparing Review*, pages 75–102. Springer Berlin Heidelberg, Berlin, Heidelberg. 25

Hansen, P. and Mladenović, N. (2001). Variable neighborhood search: Principles and applications. *European journal of operational research*, 130(3):449–467. 10

Hayes, C. F., Rădulescu, R., Bargiacchi, E., Källström, J., Macfarlane, M., Reymond, M., Verstraeten, T., Zintgraf, L. M., Dazeley, R., Heintz, F., Howley, E., Irissappane, A. A., Mannion, P., Nowé, A., Ramos, G., Restelli, M., Vamplew, P., and Roijers, D. M. (2021). A practical guide to multi-objective reinforcement learning and planning. 24

Henn, S. (2015). Order batching and sequencing for the minimization of the total tardiness in picker-to-part warehouses. *Flexible Services and Manufacturing Journal*, 27(1):86–114. 10, 11, 12

Henn, S., Koch, S., Doerner, K. F., Strauss, C., and Wäscher, G. (2010). Metaheuristics for the order batching problem in manual order picking systems. *Business Research*, 3(1):82–105. 10, 12

Henn, S., Koch, S., and Wäscher, G. (2012). Order batching in order picking warehouses: a survey of solution approaches. In *Warehousing in the global supply chain*, pages 105–137. Springer. 5, 6, 7, 29

Henn, S. and Schmid, V. (2013). Metaheuristics for order batching and sequencing in manual order picking systems. *Computers & Industrial Engineering*, 66(2):338–351. 10, 11, 12

Henn, S. and Wäscher, G. (2012). Tabu search heuristics for the order batching problem in manual order picking systems. *European Journal of Operational Research*, 222(3):484–494. 10

Hester, T., Vecerik, M., Pietquin, O., Lanctot, M., Schaul, T., Piot, B., Horgan, D., Quan, J., Sendonaris, A., Dulac-Arnold, G., et al. (2017). Deep q-learning from demonstrations. *arXiv preprint arXiv:1704.03732*. 22

Hill, A., Raffin, A., Ernestus, M., Gleave, A., Kanervisto, A., Traore, R., Dhariwal, P., Hesse, C., Klimov, O., Nichol, A., Plappert, M., Radford, A., Schulman, J., Sidor, S., and Wu, Y. (2018). Stable baselines. `https://github.com/hill-a/stable-baselines`. 37, 52

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*. 16

Hu, H., Jia, X., He, Q., Fu, S., and Liu, K. (2020). Deep reinforcement learning based agvs real-time scheduling with mixed rule for flexible shop floor in industry 4.0. *Computers & Industrial Engineering*, 149:106749. 20, 21

Hubbs, C. D., Li, C., Sahinidis, N. V., Grossmann, I. E., and Wassick, J. M. (2020). A deep reinforcement learning approach for chemical production scheduling. *Computers & Chemical Engineering*, 141:106982. iii, 2, 18, 21

Hwang, C.-L. and Masud, A. S. M. (2012). *Multiple objective decision making—methods and applications: a state-of-the-art survey*, volume 164. Springer Science & Business Media. 23

Injadat, M., Moubayed, A., Nassif, A. B., and Shami, A. (2020). Systematic ensemble model selection approach for educational data mining. *Knowledge-Based Systems*, 200:105992. 25

Kasimbeyli, R., Ozturk, Z. K., Kasimbeyli, N., Yalcin, G. D., and Erdem, B. I. (2019). Comparison of some scalarization methods in multiobjective optimization. *Bulletin of the Malaysian Mathematical Sciences Society*, 42(5):1875–1905. 24

Lee, M.-K. and Kim, S.-Y. (1995). Scheduling of storage/retrieval orders under a just-in-time environment. *International Journal of Production Research*, 33(12):3331–3348. 8, 12

Li, J., Huang, R., and Dai, J. B. (2017). Joint optimisation of order batching and picker routing in the online retailer's warehouse in china. *International Journal of Production Research*, 55(2):447–461. iv, 10, 12, 45

Lillicrap, T. P., Hunt, J. J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D., and Wierstra, D. (2015). Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*. 13

Mao, H., Alizadeh, M., Menache, I., and Kandula, S. (2016). Resource management with deep reinforcement learning. In *Proceedings of the 15th ACM Workshop on Hot Topics in Networks*, pages 50–56. 19, 21

Marom, O. and Rosman, B. (2018). Belief reward shaping in reinforcement learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32. 2

Miettinen, K. (2012). *Nonlinear multiobjective optimization*, volume 12. Springer Science & Business Media. 23

Mnih, V., Badia, A. P., Mirza, M., Graves, A., Lillicrap, T., Harley, T., Silver, D., and Kavukcuoglu, K. (2016). Asynchronous methods for deep reinforcement learning. In *International conference on machine learning*, pages 1928–1937. 17

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*. 13

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., et al. (2015). Human-level control through deep reinforcement learning. *nature*, 518(7540):529–533. 13, 15, 16

Muter, İ. and Öncan, T. (2015). An exact solution approach for the order batching problem. *IIE Transactions*, 47(7):728–738. 6, 7

Nair, A., Srinivasan, P., Blackwell, S., Alcicek, C., Fearon, R., De Maria, A., Panneershelvam, V., Suleyman, M., Beattie, C., Petersen, S., et al. (2015). Massively parallel methods for deep reinforcement learning. *arXiv preprint arXiv:1507.04296*. 16

Nguyen, T. T., Nguyen, N. D., Vamplew, P., Nahavandi, S., Dazeley, R., and Lim, C. P. (2020). A multi-objective deep reinforcement learning framework. *Engineering Applications of Artificial Intelligence*, 96:103915. 2, 23, 24

Oroojlooyjadid, A., Nazari, M., Snyder, L., and Takáč, M. (2020). A deep q-network for the beer game: A deep reinforcement learning algorithm to solve inventory optimization problems. 20, 21

Peer, E., Menkovski, V., Zhang, Y., and Lee, W.-J. (2018). Shunting trains with deep reinforcement learning. In *2018 ieee international conference on systems, man, and cybernetics (smc)*, pages 3063–3068. IEEE. 1

Perny, P. and Weng, P. (2010). On finding compromise solutions in multiobjective markov decision processes. In *ECAI*, volume 215, pages 969–970. 23

Pinto, A. R. F. and Nagano, M. S. (2019). An approach for the solution to order batching and sequencing in picking systems. *Production Engineering*, 13(3-4):325–341. 11, 12

Rjoub, G., Bentahar, J., Abdel Wahab, O., and Saleh Bataineh, A. (2020). Deep and reinforcement learning for automated task scheduling in large-scale cloud computing systems. *Concurrency and Computation: Practice and Experience*. 19, 21

Rummery, G. A. and Niranjan, M. (1994). *On-line Q-learning using connectionist systems*, volume 37. University of Cambridge, Department of Engineering Cambridge, UK. 17

Rummukainen, H., Nurminen, J. K., and Nurminen, J. K. (2019). Practical reinforcement learning-experiences in lot scheduling application. *IFAC-PapersOnLine*, 52(13):1415–1420. 18, 21

Schaul, T., Quan, J., Antonoglou, I., and Silver, D. (2015). Prioritized experience replay. *arXiv preprint arXiv:1511.05952*. 16

Scholz, A., Schubert, D., and Wäscher, G. (2017). Order picking with multiple pickers and due dates–simultaneous solution of order batching, batch assignment and sequencing, and picker routing problems. *European Journal of Operational Research*, 263(2):461–478. 11, 12

Schulman, J., Levine, S., Abbeel, P., Jordan, M., and Moritz, P. (2015). Trust region policy optimization. In *International conference on machine learning*, pages 1889–1897. 16

Schulman, J., Wolski, F., Dhariwal, P., Radford, A., and Klimov, O. (2017). Proximal policy optimization algorithms. 16, 17, 51, 52

Shi, D., Fan, W., Xiao, Y., Lin, T., and Xing, C. (2020). Intelligent scheduling of discrete automated production line via deep reinforcement learning. *International Journal of Production Research*, 58(11):3362–3380. iii, 2, 18, 21

Sigaud, O. and Buffet, O. (2013). *Markov decision processes in artificial intelligence*. John Wiley & Sons. 14

Silva, V. d. N. and Chaimowicz, L. (2017). Moba: a new arena for game ai. *arXiv preprint arXiv:1705.10443*. 13

Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., Van Den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., et al. (2016). Mastering the game of go with deep neural networks and tree search. *nature*, 529(7587):484–489. 13

Snoek, J., Larochelle, H., and Adams, R. P. (2012). Practical bayesian optimization of machine learning algorithms. 42

Statista (2020). Retail e-commerce sales worldwide from 2014 to 2023. 1

Student (1908). The probable error of a mean. *Biometrika*, pages 1–25. 50, 56

Sutton, R. S., Barto, A. G., et al. (1998). *Introduction to reinforcement learning*, volume 135. MIT press Cambridge. iii, viii, 2, 13, 14, 15, 19

Tsai, C.-Y., Liou, J. J., and Huang, T.-M. (2008). Using a multiple-ga method to solve the batch picking problem: considering travel distance and order due time. *International Journal of Production Research*, 46(22):6533–6555. 9, 12

Valle, C. A., Beasley, J. E., and da Cunha, A. S. (2017). Optimally solving the joint order batching and picker routing problem. *European Journal of Operational Research*, 262(3):817–834. 6, 7

Vamplew, P., Yearwood, J., Dazeley, R., and Berry, A. (2008). On the limitations of scalarisation for multi-objective reinforcement learning of pareto fronts. In *Australasian joint conference on artificial intelligence*, pages 372–378. Springer. 23

Van Den Berg, J. P., Van Der Hoff, H. H., et al. (2001). An order batching algorithm for wave picking in a parallel-aisle warehouse. *IIE transactions*, 33(5):385–398. 6, 7

van Gils, T., Caris, A., Ramaekers, K., and Braekers, K. (2019). Formulating and solving the integrated batching, routing, and picker scheduling problem in a real-life spare parts warehouse. *European Journal of Operational Research*, 277(3):814–830. 12

Van Hasselt, H., Guez, A., and Silver, D. (2015). Deep reinforcement learning with double q-learning. *arXiv preprint arXiv:1509.06461*. 16

Vinyals, O., Ewalds, T., Bartunov, S., Georgiev, P., Vezhnevets, A. S., Yeo, M., Makhzani, A., Küttler, H., Agapiou, J., Schrittwieser, J., et al. (2017). Starcraft ii: A new challenge for reinforcement learning. *arXiv preprint arXiv:1708.04782*. 13

Wang, Y., He, H., Wen, C., and Tan, X. (2020). Truly proximal policy optimization. 17

Wang, Z., Bapst, V., Heess, N., Mnih, V., Munos, R., Kavukcuoglu, K., and de Freitas, N. (2016a). Sample efficient actor-critic with experience replay. *arXiv preprint arXiv:1611.01224*. 16

Wang, Z., Schaul, T., Hessel, M., Hasselt, H., Lanctot, M., and Freitas, N. (2016b). Dueling network architectures for deep reinforcement learning. In *International conference on machine learning*, pages 1995–2003. 16

Waschneck, B., Reichstaller, A., Belzner, L., Altenmüller, T., Bauernhansl, T., Knapp, A., and Kyek, A. (2018). Optimization of global production scheduling with deep reinforcement learning. *Procedia CIRP*, 72(1):1264–1269. iii, 2, 18, 21

Watkins, C. J. and Dayan, P. (1992). Q-learning. *Machine learning*, 8(3-4):279–292. 17

Won, J. and Olafsson, S. (2005). Joint order batching and order picking in warehouse operations. *International Journal of Production Research*, 43(7):1427–1442. 8, 12

Yaman, H., Karasan, O. E., and Kara, B. Y. (2012). Release time scheduling and hub location for next-day delivery. *Operations Research*, 60(4):906–917. 1

Yeo, S., Oh, S., and Lee, M. (2020). Accelerated deep reinforcement learning with efficient demonstration utilization techniques. *World Wide Web*, pages 1–23. 22

Ying, C.-s., Chow, A. H., and Chin, K.-S. (2020). An actor-critic deep reinforcement learning approach for metro train scheduling with rolling stock circulation under stochastic demand. *Transportation Research Part B: Methodological*, 140:210–235. 20, 21

Yisong, Y. and Hoang M, L. (2018). Imitation learning tutorial. In *International conference on machine learning*, pages 1889–1897. 22

Young, M. T., Hinkle, J., Ramanathan, A., and Kannan, R. (2018). Hyperspace: Distributed bayesian hyperparameter optimization. In *2018 30th International Symposium on Computer Architecture and High Performance Computing (SBAC-PAD)*, pages 339–347. IEEE. 25

Zhang, J., Wang, X., Chan, F. T., and Ruan, J. (2017). On-line order batching and sequencing problem with multiple pickers: A hybrid rule-based algorithm. *Applied Mathematical Modelling*, 45:271–284. 9, 12