Integrated production & distribution scheduling at premix feed producers using deep reinforcement learning

van Dijk, M.J.W.

Award date:
2022

Department of Industrial Engineering & Innovation Sciences
Information Systems Research Group

Integrated production & distribution scheduling at premix feed producers using deep reinforcement learning

Master's Thesis

Mart van Dijk
0956773

Supervisors:
Dr. Ir. Zaharah Bukhsh - TU/e
Prof. Dr. Ir. Remco Dijkman - TU/e
Lukas Bruijnel - KSE Process Technology

Eindhoven, June 2022

Abstract

Current scheduling techniques fail to provide suitable solutions for the integrated production and distribution scheduling problem with sequence-dependent setup times, dynamic order arrival and uncertainty. Therefore, real-life production and distribution scheduling is done manually and separately, which leads to hiccups in the operation. We present a novel deep reinforcement learning approach to solve the given problem automatically. First, the problem was formulated as a Semi-Markov Decision Process and modelled in a simulation model. Then, a Proximal Policy Optimization algorithm was trained by interacting with the simulation model. Due to the sequential decision making in the production and distribution process, multi-discrete actions were included. Furthermore, action masking was applied to accelerate the learning process. To evaluate the performance of the algorithm, two greedy algorithms were developed. Different experiments showed that the deep reinforcement learning algorithm outperformed the greedy algorithms when uncertainty was included. The deep reinforcement learning algorithm was able to reduce the makespan by 25.63% when low uncertainty was included, which resulted in 15.13% more on time delivered orders compared to the best performing benchmark. For the high uncertainty case, the deep reinforcement learning algorithm reduced the makespan by 53.70%. However, it failed to deliver more orders on time, which indicates that the distribution part could be improved. In addition, the deep reinforcement learning algorithm appeared to be a robust solution that can be applied to comparable situations as its training environment.

Keywords: Deep Reinforcement Learning, Flow-shop Scheduling, Semi-Markov Decision Process (SMDP), Premix Manufacturers

Executive summary

Problem context

This thesis concerns the integration and automation of production and distribution scheduling at premix feed manufacturers and is conducted at KSE Process Technology (KSE). A novel deep reinforcement learning (DRL) approach is designed to solve the problem. Premix feed manufacturers produce a crucial feed additive for animal compound feed. Nowadays, livestock food is more than solely residual product from corn or grain. Nutrition is carefully selected and composed of different ingredients to enhance animal welfare. Adequate nutrition requires feed composition consisting of macro-ingredients (such as corn or grain) and micro-ingredients (such as vitamins, minerals and amino acids). To ensure homogeneous distribution of micro-ingredients with macro-ingredients, the micro-ingredients are premixed by special premix factories. Premix production is mostly done in bulk, which requires different material handling than regular cargo.

KSE provides machines and software for the production of premix feed. Their machines are used to dose, weight and transport bulk material. Besides, the software package automatically manages and monitors the whole production process. KSE noticed that the production and distribution schedules of their customers (premix feed producers) are not aligned. The mismatch leads to hiccups in the operation as trucks have to wait for production to be finished and production waiting for trucks to arrive. One of the reasons for the mismatch is the fact that both schedules are made separately. Current planning software packages are incompetent to deal with the dynamism of the premix production and distribution environment. Moreover, most premix producers manually schedule both operations.

Premix manufacturers have asked KSE for a solution, i.e. to have an automated and combined production and distribution planning. As mentioned before, current solutions are inadequate since the premix production environment is highly dynamic. Schedules are made on a daily basis while orders arrive throughout the day. Furthermore, schedules are adapted based on the most recent information. Additionally, unexpected events occur such as equipment downtime, lack of raw materials and truck delay. Moreover, contamination of raw materials makes the production schedule challenging as this constrains production sequences.

Research approach

In recent years, studies have shown that DRL is well suited to solve comparable problems. Especially dynamic problems were information becomes available over time and where uncertainty occurs (Zhou et al., 2020), (L. Wang et al., 2021) & (Kardos et al., 2021). DRL algorithms train themselves by interacting with an environment. The algorithm chooses an action and observes the subsequent state of the environment and feedback in the form of a reward. Based on these interactions, it defines a policy. To study the applicability of DRL on the given problem, the following research question is defined:

How can the production and distribution schedule of premix feed producers be automated and integrated by means of deep reinforcement learning, while accounting for unforeseen events?

Method

DRL algorithms require a specific formulation of the environment to interact with it. Based on the work of Rummukainen and Nurminen (2019), the decision was made to model the environment as a Semi-Markov Decision Process (SMDP). SMDP's have the advantage over MDP's that they transition to the next state based on an event, instead of a fixed time interval. Therefore, transitions only occur when an action from the DRL algorithm is required. Consequently, the learning process is accelerated as it eliminates the transitions where the DRL algorithm learns nothing. The environment is represented by a simulation model that allows interaction with the DRL algorithm. The simulation model is based on an actual premix feed factory.

During the production and distribution process, four discrete actions are required. (i) Order from queue to mixing machine allocation, (ii) order from mixing machine to sacking machine allocation, (iii) order from sacking machine to storage space allocation and (iv) order from storage space to truck allocation. The state space includes information about orders in the system, resources and delivered orders. Furthermore, the reward function rewards on time delivered orders and penalizes the makespan of each order. The relative weight of each reward component is optimized with a Tree-structured Parzen Estimator optimizer. The Proximal Policy Optimization (PPO) algorithm (Schulman et al., 2017) is implemented, which is one of the most advanced DRL algorithms available. Besides, the PPO algorithm is one of the few DRL algorithms that handles multi-discrete actions. In addition, action masking is applied because many actions are invalid at each decision moment which is a consequence of the sequential action design and the choice for the SMDP framework. At each transition, an action is required at a specific part of the process, i.e. one of the four actions. For the other parts of the process, no action is required at that moment in time, making those actions invalid.

Two greedy algorithms are developed to benchmark the PPO algorithm on various experiments, one which accounts for contamination and one which did not account for contamination. The experiments include a situation with overcapacity, undercapacity, low uncertainty and high uncertainty. Uncertainty is modelled through machine breakdown and truck delay. Besides, a case study is conducted which is based on real data. For each setting, the DRL algorithm is trained separately. Therefore, a robustness analysis is included to asses the general applicability of the DRL algorithm. Moreover, the offline reinforcement learning algorithms Batch-Constrained Q-learning and Conservative Q-learning were trained on transition datasets acquired by the overcapacity DRL agent.

Results

The results show that the DRL algorithm is able to learn a stable policy for all experiments. Besides, the DRL algorithms learn the contamination rules in all experiments. Contamination is an important aspect of the manufacturing process in premix feed production and therefore crucial to be learned by a scheduling algorithm. The results show that the agent outperforms the greedy algorithms when uncertainty is included. Especially for the low uncertainty case, where the makespan is reduced by 25.63% and the on time delivered orders increased by 15.13%. In the high uncertainty case, the number of delivered orders are comparable, while the makespan is 53.70% lower for the DRL algorithm. The same results are observed for the case study. The DRL agent schedules the production part significantly better than the benchmarks (58.83% lower makespan), yet fails to deliver more orders on time (71.75% more tardy orders). Furthermore, the robustness experiments show that the DRL algorithm can handle comparable cases as its training environment. Besides, the robustness analysis shows that the low uncertainty trained DRL algorithm performs better than the other DRL algorithms, including the DRL algorithms that were trained on the environment. The results of the offline reinforcement learning algorithms show that these algorithms were not able to find a good policy.

Conclusion & recommendation

This thesis presents a novel automated solution for the integrated production and distribution scheduling problem. The results show that the DRL algorithm learns a good policy and outperforms the benchmark algorithms on the low uncertainty case. Furthermore, the DRL algorithm outperforms the benchmark algorithms on the production part of the high uncertainty case. However, this advantage is not reflected in the number of on time delivered orders for the high uncertainty case. Therefore, it can be concluded that the distribution part of the DRL framework should be adapted before the DRL agent can be used in real-life scheduling situations. Improving the distribution part of the DRL framework can be done by postponing the order - truck allocation or allowing rescheduling after unforeseen events. Furthermore, certain assumptions were made during the study, which should be relaxed in future research. Nevertheless, the fundamental properties of DRL show promising applicability to the scheduling environment of premix manufacturers. Especially through its ability to deal with dynamic order arrival and uncertainty, it is well suited for scheduling on a daily basis. Therefore, we recommend KSE to continue with research on DRL. Specifically with expanding the distribution part and extending the contamination modelling. In addition, enabling rescheduling and making the DRL algorithm's policy explainable are directions for future research.

Preface

This report is the result of my graduation research for the Master's programme Operations Management & Logistics at the TU/e. Before you continue to read my work, I would like to take the opportunity to express my gratitude towards a few people who have guided me throughout this master thesis and my studies as a whole.

First of all, I want to thank my first university supervisor Zaharah Bukhsh for the thorough support. Our weekly meetings, which continued during her maternity leave, helped me to understand the field of deep reinforcement learning, even though this was new to me. Furthermore, I want to thank my second university supervisor Remco Dijkman for his critical view on my work. This definitely improved the overall outcome of the thesis. Moreover, I want to thank Michiel Willemse, my first supervisor at KSE who initiated this project. Our meetings in the early stage of the project helped to understand the premix manufacturing environment. Besides, I want to thank Lukas Bruijnel, who took over the supervising role without hesitation from Michiel when he left KSE.

To conclude, I want to thank my friends, family and my girlfriend Paulien for the distraction during stressful times throughout the project. Without your love and support this would not have been possible.

Enjoy!

Mart van Dijk

Contents

List of Figures

List of Tables

1 Introduction

This report is the Master's thesis for the study Operations Management & Logistics at the TU/e and was conducted at KSE Process Technology. In the study, deep reinforcement learning was used to schedule the production and distribution process at premix feed manufacturers. The first section provides an introduction to the industry where KSE Process Technology is active in. After that, the stakeholders are discussed and the problem description is provided. Based on the problem description, the research questions and the outline of the thesis are given.

1.1 Industry

KSE Process Technology is, among others, active in the premix feed industry. The premix feed industry is the facilitator of premix for the production of livestock food. Nowadays, livestock food is more than just residual product from corn or grain. Nutrition is carefully selected and composed of different ingredients to enhance animal welfare. Besides nutritional composition, price and animal characteristics are the main criteria to compile premix feed (Zahari & Alimon, 2005). Often nutritional value comes from micro-ingredients such as vitamins, minerals and amino acids. To ensure equal distribution of these ingredients in the feed, they are premixed by special premix factories. The premix can be added to premix feed by compound feed producers or directly by farmers. Nutritional composition is matched to animal type and growth stage. Therefore, up to 300 different types of feed are produced by premix manufacturers. Profit margins for premix manufacturers are small. Consequently, production of premix is done in large quantities of multiple tonnes and production plants operate around the clock to be profitable. Raw material costs account for the largest part of production costs (Ness & Walker, 1995). Therefore, the production process is highly automated and optimized to a great extent. Due to the variety of products in the market and relatively small lead time, most products are made to order (MTO) although fast selling products are made to stock (MTS). The ratio MTO versus MTS differs per producer. Producers which make more specialized premix feed tend to be MTO focused, while producers of general premix feed are MTS focused.

Production, shipment and storage of most raw materials are done in bulk due to large volumes. This requires different material handling than normal cargo. Transportation of bulk material inside a production plant is done with conveyor belts, bucket elevators and pneumatic conveying (transportation based on gasses). Raw material are stored in silos. These silos are filled at the top and emptied by gravity at the bottom. Due to different handling of bulk materials, special equipment is required for the production and transportation. Shipment of finished product can be done in small bags (15 kg up to 25 kg) or big bags (1000 kg) as well in bulk.

1.2 KSE Process Technology

KSE Process Technology (hereafter named KSE) is solely active in animal feed industries such as premix, compound feed, pet food and aquafeed. It serves the premix industry with two types of products. First of all, KSE makes specialized machines under the name Alfra. Alfra is machinery used for precisely dosing and weighing bulk material and is used at multiple stages in the production process of premix. Dosing and weighing is a process in which bulk material is retrieved from a silo or machine and carefully dosed and weighed to obtain the exact required amount. This is important because finished products should contain the same ratio of ingredients as the recipe. Besides, premix producers want to deliver the exact ordered amount to customers due to the small profit margins. Furthermore, Alfra makes machines which are used to transport bulk material inside the production plant from silos to machines or from silos to trucks. Secondly, KSE makes

the factory automation software Promas ST. Promas ST is specialized for animal feed production facilities. The software is used to automatically manage and monitor the whole production process from manufacturing execution system (MES) to real time control. It can control both Alfra and third party machinery. Since KSE provides machines and software for the premix producers, they are not directly involved in the production of premix although KSE works closely with producers to innovate the industry.

1.3 Stakeholders

Stakeholders in the industry are customers, premix producers and suppliers. Customers are premix feed producers or farmers who order premix at the producers. Premix producers produce the product at production plants. Suppliers can be divided into suppliers of raw material and suppliers of equipment or services for the production of premix. KSE is a supplier of both equipment and services as it provides dosing and weighing machines and factory automation software.

1.4 Problem statement

KSE noticed that the production and distribution schedules of their customers (premix feed producers) are not aligned. This leads to hiccups in the operation as trucks have to wait for production to be finished and production waiting for trucks to arrive. One of the reasons for the mismatch between production and distribution is the fact that both schedules are made separately. Current planning software packages are incompetent to deal with the dynamism of the premix production and distribution environment. Moreover, most premix producers manually schedule both operations. This leads to another problem; premix feed manufacturers heavily rely on a few employees who know the scheduling rules by heart. Due to the complexity of the scheduling process, no documentation is available about the scheduling procedures. Thus, sickness or retirement of one of those employees could have drastic consequences for the premix manufacturers.

Premix manufacturers have asked KSE for a solution, i.e. to have an automated and combined production and distribution planning. As mentioned before, current solutions are inadequate since the premix production environment is highly dynamic. Schedules are made on a daily basis while orders arrive throughout the day. Furthermore, schedules are adapted based on the most recent information. Additionally, unexpected events occur such as equipment downtime, lack of raw materials and truck delay. During the early literature review, the machine learning technique deep reinforcement learning (DRL) showed promising results in comparable problems. Especially in dynamic problems were information becomes available over time and where uncertainty occurs. Therefore, this thesis will investigate if and how DRL could be used to solve the integrated production and distribution problem of premix feed manufacturers.

1.5 Research questions

Based on the problem description, the main research question is formulated as follows:

How can the production and distribution schedule of premix feed producers be automated and integrated by means of deep reinforcement learning, while accounting for unforeseen events?

Before the main research question can be answered, the following sub-questions should be answered first:

1. *How to define the integrated production and distribution process?*
   A mathematical formulation is used to define the scope of the study.
2. *Which existing solution methods are used to solve given problem?*
   A literature review is conducted to investigate which current solutions are used to solve the given problem. This could help with defining a novel solution.
3. *Which deep reinforcement learning techniques are used to solve the given problem?*
   A literature review is done to get an understanding of the capabilities of deep reinforcement learning and how this can be applied to the given problem.
4. *How to formulate the integrated and automated production and distribution scheduling problem at premix feed producers as a Markov Decision Process?*
   This sub-question helps to define the problem as a MDP, which is the basis of a DRL model.
5. *Which of the deep reinforcement learning algorithms is most suitable for the solving the scheduling problem?*
   Before designing a DRL model, different DRL algorithms are compared to find the most suitable algorithm for solving the given problem.
6. *What is the performance of the deep reinforcement model compared to a heuristic scheduling techniques?*
   This sub-question helps to evaluate the proposed DRL model. By comparing the model with a heuristic scheduling technique, the quality can be assessed.

1.6 Outline

The thesis is structured as follows. First, an extensive literature review is done to get a general understanding of the problem, existing solution methods, DRL and applications of DRL. Then, the problem description is provided. An overview of the production and distribution process is given, followed by a description of the current scheduling process and the desired scheduling process. Based on this information and a mathematical problem formulation, the scope of the problem is defined. Then, the solution method is described. This includes an overview of the states, actions and rewards design, simulation model and the tested algorithms. Subsequently, data from a customer of KSE is analysed to capture the real-life dynamics of the environment. Then, the experimental setup is described, the parameter settings of the simulation environment, training setup, scenario analysis, case study and robustness analysis. Furthermore, the results of the experiments are described, after which the conclusions are drawn. Here, the research questions are answered, as well as a discussion of the thesis and directions for future research.

2 Literature review

A literature review is conducted to find a suitable solution for the problem statement given in the first section. To start, an introduction to the problem in current literature is given. Then, deep reinforcement learning is explained by providing an overview of the terms used in deep reinforcement learning, neural networks and the different deep reinforcement learning algorithms. Last, existing deep reinforcement learning methods are discussed in the field of scheduling and other applications.

2.1 Existing approaches in the literature

This subsection provides an introduction to the integrated production and distribution scheduling problem in the literature. First, the problem is defined based on the terms used in the literature. Afterwards, existing solution approaches are reviewed and elaborated on.

2.1.1 Production & distribution scheduling in the literature

Scheduling premix feed production matches the characteristics of flow shop scheduling. *Flow shop scheduling* is a variant of the classical job scheduling problem. The *job scheduling* problem is about scheduling n jobs J_i $(i = 1, ..., n)$ which have to be processed on m machines M_i $(i = 1, ..., m)$. In job scheduling the following assumptions are made:

1. each job can be processed on one machine at the time;
2. each machine can process one job at a time;
3. processing times are deterministic and schedules are static;
4. jobs may vary in processing time per machine and machines may have different processing power.

For flow shop scheduling, the first assumption is substituted with:

1. each job must be processed on multiple machines and in an identical machine ordering.

The properties of flow shop scheduling are true for premix producers, since jobs must be processed on different machines in identical order. Other variants of the multi-machine job scheduling problem are job shop scheduling and open shop scheduling. In job shop scheduling, jobs can have distinct machine ordering, whereas in open shop scheduling jobs the order of operations is immaterial (Graham et al., 1979). Based on the research by Graham et al. (1979) ample extensions of the classical job shop scheduling problem are investigated in the literature, such as Baker et al. (1983), Baker and Scudder (1990) and Raghavachari (1988).

In this study, the flow shop scheduling problem is combined with a distribution schedule. Combined production and distribution scheduling is studied thoroughly in the literature. Mostly on high level such as strategic and tactical level, whereas operational scheduling is relatively new (Z.-L. Chen, 2009). Since the goal of this thesis is to find a daily planning, the focus will be on operational schedules. Furthermore, combined production and distribution scheduling problems can be divided in three categories based on the used distribution scheduling method (Karaoğlan & Kesen, 2017):

1. Direct shipping of a production batch to one customer immediately after production;
2. Predetermined departure time of the trucks;
3. Routing trucks in a distribution schedule.

This study focuses on the production schedule while including order to truck allocation with predetermined departure times. Finished products are shortly stored before shipment to customers, therefore storage between production and distribution should be considered as well. Integrated production and distribution scheduling problems often consider a storage stage in between production and distribution because intermediate storage connects production and distribution (Z.-L. Chen, 2009). A dynamic setting will be considered due to the dynamic nature of the scheduling environment. To conclude, the problem is defined as a dynamic integrated permutation flow shop and truck allocation problem.

2.2 Existing approaches

In this section, articles are reviewed which have at least some aspect of the integrated production and distribution scheduling problem. The section is divided per component of the problem and a distinguish is made between static and dynamic scheduling due to the different solution approaches. First, integrated production and distribution scheduling studies, which are closely related to the problem, are discussed. Then, permutation flow shop scheduling studies are reviewed because this subcategory of job shop scheduling relates the most to KSE's problem. Followed by general flow shop scheduling problems. After that, fixed vehicle departure times, sequence-dependent setup times and perishability are elaborated on, because those are rare subjects in job shop scheduling literature yet important aspects of the problem. The static section is concluded by studies conducted in other domains with similar characteristics of at least one component of the problem. In the dynamic scheduling section, different sorts of job shop scheduling studies are discussed.

2.2.1 Integrated production & distribution scheduling

The integrated production and distribution scheduling problem is the most overlapping problem with the case of this thesis. Unfortunately, there are only a few articles in current literature who studied this problem with characteristics that match the type of production and distribution scheduling as described in the previous section.

- Hou et al. (2022) studied the integrated production and distribution scheduling problem with a distributed flow shop and delivery time windows. Multiple factories are considered, each with a flow shop production process. Jobs are assigned to trucks and trucks are routed to the customers. A brain storm optimization algorithm is applied to solve the problem, which is a swarm intelligence method inspired by swarm behavior of humans during brainstorm processes. At the production stage, each job is assigned to the factory with the smallest makespan and all jobs are processed in the determined sequence. For the distribution stage, jobs are assigned and loaded to vehicles based on the production sequence. To asses the quality of the algorithm, it was compared to a solver and other meta heuristics. Results showed that the heuristic outperformed the solver for instances from 6 jobs onward and was able to find a better objective value for large instances than the meta heuristics.
- K. Li et al. (2015) studied the integrated production and distribution scheduling problem in a parallel batching setting where profit is maximized. They showed that problems with

identical job sizes can be solved optimally in an efficient way, while nonidentical job sizes cannot be solved exact. Since the problem considered in this thesis does not have identical job sizes this will not be addressed further. A heuristic was used to solve the problem where jobs are scheduled in descending order of their profit versus size. Delivery is done by the first-fit method (Garey & Johnson, 1981), meaning latest produced job j is delivered first. If job j does not fit in the truck next job j is delivered.

- Furthermore, Mortazavi et al. (2015) proposed a chaotic imperialist competitive algorithm (CICA) for the production and distribution scheduling problem with a single machine and air transportation. The CICA solves the proposed problem sequentially, first the transportation sub-problem and then the production sub-problem. Chaos theory has three principles: a simple system exposes a complex behavior, complex systems expose a seemingly random behavior yet it follows a predictable sequence and third the system behavior is related to the initial state. The chaotic version of ICA algorithms prevent premature convergence. Results were compared with a GA and it was shown that the CICA outperformed the GA in both solution quality and robustness.

From these works can be concluded that the integrated production and distribution scheduling problem is a difficult problem and requires advanced meta-heuristics to get a good solution for larger instances.

2.2.2 Permutation flow shop

The permutation flow shop scheduling problem (PFSP) has been studied by several researchers and is the sub-category of job shop scheduling that best matches the characteristics of this problem. The permutation flow shop differs from the standard flow shop scheduling problem because the processing order of all jobs is the same for each subsequent step of processing. This does not have to be the case for flow shop scheduling problems.

- Marsetiya Utama et al. (2020) developed a hybrid antlion optimization algorithm (HALO) for the PFSP. The objective was to minimize the mean tardiness. The HALO algorithm mimics the interaction between antlions and ants. Ants move randomly in the search space (similar to individuals in GA) whereas antlions hunt them and become fitter using traps (Mirjalili, 2015). The HALO algorithm found slightly better solution compared to other (meta)heuristics although it required much more computational time.
- Besides, a discrete artificial bee colony (ABC) algorithm was used to optimize the PFSP by Deng et al. (2016). They considered buffers in between machines and minimized the makespan. According to the authors, their ABC outperformed two earlier developed ABC algorithms and both evolution and greedy algorithms.
- Mishra and Shrivastava (2018) studied this problem while optimizing the inventory holding and batch delay costs. Multiple meta heuristics were compared to one another and it was found that the jaya heuristic performed best for large (500 jobs, 20 machines) instances. Jaya is based on the idea that solutions should move towards the best solution while it goes away from the worst solution. First it initializes the population, followed by the identification of the best and worst solutions. The other solutions are modified based on the best and worst solutions and then compared to each other. If the updated solution is better it replaces the previous solution. These steps are repeated until the termination criteria are met.
- In the work of Rifai et al. (2021), a combination is made of the permutation flow shop and sequence-dependent setup times. An adaptive large neighborhood search algorithm was proposed in order to minimize the makespan, production costs and tardiness. Since this is

a multi-objective optimization problem, there is no unique optimal solution. Instead, the algorithm creates a set of non-dominating solutions.

- Gonzalez-Neira et al. (2021) extended this problem by including stochasticity in the processing times and sequence-dependent setup times. A sim-heuristic was proposed to solve this problem by combining a greedy randomized adaptive search procedure (GRASP), a Monte Carlo simulation, a Pareto archived evolution strategy (PAES) and an analytic hierarchy process (AHP). Four objective functions were optimized which resulted in a Pareto frontier. They showed that there is an interaction effect between the coefficient of variation and the processing times and coefficient of variation of the setup times which underlines the importance of including uncertainties in the optimization process of production scheduling. The permutation flow shop scheduling problem was combined with a vehicle routing problem in the work of Yağmur and Kesen (2020). To formulate the problem a MILP was made, which could not be solved within a reasonable time frame. The authors proposed a memetic algorithm (MA) to find a near-optimal solution. MA is an adaption of a GA and uses local search techniques to prevent premature convergence. The results showed that the MA outperforms the solver for the large instances and converges rapidly.

These articles have, besides the permutation flow shop scheduling, at least one other component of the problem in the thesis. Therefore, they provide some insight in the performance of the proposed solutions if they were applied to the thesis's problem. Unfortunately, none of them included all aspects of the problem.

2.2.3 Flow shop production scheduling

Even though flow shop scheduling problems are a more general category of the problem in the thesis, the dynamics of the environment are closely related. Therefore, these articles are still relevant to get an understanding of possible solutions for KSE.

- Zheng et al. (2021) proposed a hybrid meta heuristic for a flexible flow shop scheduling problem with limited buffers and deteriorating jobs. The heuristic is a combination of heuristic components, with a genetic algorithm (GA) as basis. A Nawaz-Enscore-Ham (NEH) heuristic with bottleneck elimination defines the initial solutions. NEH heuristics suggests that jobs with greater total processing time should be given a greater priority than jobs with a smaller total processing time (Nawaz et al., 1983). Furthermore, local search is enhanced by partially matched crossover and mutation of solutions using various neighborhood search structures. Besides, a VNS with SA avoids local optima and a modified CDS improves quality of non-improved solutions during an iteration. The heuristic is compared with a solver using different instances. On average, the heuristic performs 3.15% worse than the solver and performs better than other heuristics. For instances that cannot be solved with the solver, the heuristic outperforms other heuristics in terms of objective value. The orders arriving at compound feed producers can contain multiple products. Therefore, the jobs in the production process can have common due dates.
- In the work of Z. Li et al. (2021), a hybrid flow shop with common due dates is investigated. A GA was proposed to solve the given problem and was compared to existing GA's and a PSO algorithm. Results showed that the PSO was better for larger instances (5 products, 5 machines and 5 production lines) while the proposed GA performed best for smaller instances (3,3,3).

From these articles can be concluded that more complex solution approaches are likely to produce better results in specific cases, yet fail to deliver good results in multiple instances. This is

something to consider in the algorithm design for the thesis's problem since the solution should handle a variety of cases.

2.2.4 Fixed vehicle departure times

The integrated scheduling problem of production and distribution with fixed vehicle departure times are scarce in the literature. Therefore, articles which studied some form of combined production and distribution via fixed vehicle departure times are included.

- Hajiaghaei-Keshteli et al. (2014) studied an integrated static single machine production and rail transportation problem with fixed delivery departure times. They developed a GA and simulated annealing (SA) to solve the proposed problem. Results showed that the GA outperformed the SA and was more robust in other problem sizes. The GA minimizes the transportation costs and subsequently the production sequence is determined. Sequentially solving this problem does not guaranty a global optimum. Therefore, several schedules are generated and evaluated in an iterative way to find the best solution.
- Stecke and Zhao (2007) developed a MIP model for a static single machine production and transportation MTO setting with fixed delivery departure times. They showed that the problem is NP hard and that in an optimal schedule orders are produced non-preemptively and continuously. To solve the problem, they made a heuristic that gave near optimal production schedules. The heuristic starts with a non-preemptive earliest due date (NEDD) schedule based on non-decreasing order of due dates and largest order first when a tie occurs to satisfy due date and capacity constraints. Then, production order is changed until the quantity of late products no longer reduces.

These articles showed that an integrated production and distribution scheduling problem is often solved in an iterative way. Since the departure times are predetermined in this case, the distribution part is scheduled first.

2.2.5 Sequence-dependent setup times

The described problem in this thesis has sequence-dependent setup times due to contamination of materials. Many research on production scheduling either ignored setup times completely or assumed setup times are independent of job sequence. Therefore, the studies that included sequence-dependent setup times are discussed more in dept in this section.

- Zandieh et al. (2006) proposed an immune algorithm (IA) for a hybrid flow shop scheduling problem, where the objective is to minimize the makespan. IA is based on the natural immune system. Antigens refer the the objective function while antibodies refer to candidate solutions. The steps in an IA are similar to a GA. First, initial antibodies are (randomly) generated. Then, the fitness function for each antibody is determined. After that, the mating pool is generated and random crossover is done. The best known antibodies are retained together with the new offspring. Followed by mutation and fitness evaluation. This is repeated until the termination criteria is met. The algorithm performs better than a compared GA, especially for medium and large problem sizes.
- Furthermore, M. Li and Lei (2021) studied a flexible job shop combined with internal transportation and sequence-dependent setup times. The makespan, tardiness and energy consumption are simultaneously minimized by means of an imperialist competitive algorithm

(ICA). Here, a country represents a solution and the quality is measured by the objective value. The best counties become imperialists and control other countries (colonies). Then, assimilation take place where colonies get closer to the imperialist state. Followed by revolution which is a random change in a countries position. Last, there is an action called competition where imperialists can take control of colonies of the weakest imperialist. Results showed that the ICA outperformed other meta heuristics.

From these articles can be concluded that including sequence-dependent setup times leads to a more complex problem and require a tailored solution approach. This could be the reason why other studied excluded this component although it is not uncommon in real production processes.

2.2.6 Perishable items

Although perishability is not considered in the scope of this thesis, research on integrated production and distribution schedule which included perishability may have interesting findings.

- Karaoğlan and Kesen (2017) examined a static combined production and transportation scheduling problem for perishable items without inventory. A branch-and-cut (B&C) algorithm was proposed which minimized the makespan and delivery time. Upper bounds are improved by means of a SA local search heuristic. Their algorithm finds a feasible solution which is 2,4% from the lower bound within an hour and outperforms the compared GA of Geismar et al. (2008).
- Furthermore, Tangour et al. (2021) investigated a static flow shop scheduling problem for perishable items. Their objective function was to minimize the makespan and number of expired products. A GA and ant colony optimization (ACO) was developed to solve the problem. A B&C algorithm was used to obtain the lower bound of the makespan. The GA outperforms the ACO in terms of makespan and number of perished items. Furthermore, it shows similar schedules as found by the exact B&C algorithm.
- The combination of production and distribution for perishable items was examined by Huo et al. (2010) in a static single machine setting with fixed departure times. The objective is to maximize profit, which is earned when a job is delivered at its due date. A pseudo polynomial time algorithm was used to solve the problem exactly. This was possible due to the small instances that were used. Furthermore, they showed that the same algorithm can be used when there are an arbitrary number of parallel machines.
- Solina and Mirabelli (2021) studied perishable items in an integrated production and distribution scheduling problem at a vegetable food supplier, considering identical changeover times and inventory of finished products. Goal was to minimize energy usage, inventory and distribution costs. A rolling horizon scheme is used to schedule the production and distribution, at each iteration a bi-weekly planning is made while demand occurs every week. To find the best optimization solution, a partial rescheduling (current strategy) and complete rescheduling strategies are tested. Due to the complexity of the problem a near-optimal solution with a gap of 3% was accepted. Results showed that complete rescheduling could save the company 4% in terms of total costs and in particular the energy costs. The optimizer found this within one minute, showing its usability for real life cases.
- Although not many articles have been published about production scheduling in the compound feed industry, Toso et al. (2009) did investigate production scheduling at a compound feed producer in combination with lot sizing. This differs from this study since lot sizing is considered out of scope and a static setting was used. They modeled the production process as a single machine problem with a capacity equal to the bottleneck machine. To solve the

proposed problem, three variants of the relax and fit heuristic were developed and tested. The best performing heuristic found a solution within 21 minutes and was just 2.43% worse than the lower bound.

These articles showed that perishability has a great impact on the solution approach since it is often included in the objective function. Thus, perishability should be considered once perishability will be included in the scope of the problem.

2.2.7 Other domains

Studies in other domains may be useful for the thesis when the characteristics of the problem are approximately the same. This is the case for assembly scheduling problems as assembly production often can be modeled as a permutation flow shop. Furthermore, the second article is relevant due to the combination of a production process with fixed vehicle departure times.

- One of the studies that examined a similar problem is the work of K. Li et al. (2006) since they combined an assembly line with air transportation. Air transportation can be modeled as a fixed departure time of delivery. Therefore, this problem is similar to the problem examined in the thesis. A heuristic was proposed to solve a single machine assembly schedule with air transportation. The problem was divided into two sub problems. First, the transportation problem is solved and subsequently the production schedule is made. The assembly schedule accounts for delayed orders due to uncertain events such as equipment downtime. The heuristic efficiently reschedules jobs that were influenced by the downtime. First, affected jobs are scheduled based on the longest processing time (LPT) first rule. Then, affected jobs are inserted into the fixed idle time at the end of a period by the LPT rule. Results show that the heuristic saves on average 2.5%-4% in delivery costs compared to industry practices adopted by logistics provides.
- Sequencing the processing of incoming mail in a distribution center such that it matches the fixed truck departure schedule is relevant for this case as well due to the fixed distribution departure time. This problem was studied by Q. Wang et al. (2005). A revised greedy algorithm (RGA) performs best out of all evaluated heuristics. The biggest mail tray is selected that generates the most mail to the destination with unfilled capacity. As this might lead to over production for some destinations, unfilled destinations receive a higher weight such that in the next iteration more mail is assigned to unfilled destinations. This is repeated until no improvement can be made.

These two articles made it clear that other domains are relevant when studying a scheduling problem since there are many aspects that overlap with the case in the thesis.

2.2.8 Dynamic scheduling

Scheduling dynamic environments differs from static environment since orders arrive over time instead of full knowledge of all orders at time zero. Therefore, this section examines different applications of dynamic scheduling in job shop scheduling environments.

- Rahmani and Ramezanian (2016) examined a flexible flow shop with dynamic order arrivals, meaning not all orders are known in advance. They start with a deterministic MILP model to generate an initial schedule. As new orders arrive, the trade-off is made between adapting

the schedule with a better objective function which may cause disturbance in the system and keeping the stability of the schedule by not changing it. Therefore, the performance measures of the model are the weighted tardiness and stability. The latter is measured as the difference between completion times of the jobs in the initial schedule and modified schedule. The second part of the model is done by means of a variable neighborhood search. Furthermore, Nouiri et al. (2018) focused on the energy efficiency of the dynamic flexible job shop scheduling problem. They considered equipment downtime in their dynamic setting and minimized the energy consumption and makespan. A PSO algorithm was defined to solve the given problem.

- X. Zhang and Van De Velde (2010) studied a dynamic order arrival in a two machine job shop scheduling with time lags between the machines. A greedy algorithm was proposed to solve the problem. They showed that any greedy algorithm is two competitive for their problem. Furthermore, a dynamic two machine flow shop scheduling problem was examined by Liu and Lu (2014). They proposed an algorithm that was able to solve the problem with instances up to 1000 jobs with a mean error of 0.3%. Dynamic order arrivals in a job scheduling problem was investigated by Z. Wang et al. (2019) as well. The objective function and stability of the schedule are measured. A MILP model was made with the objective to minimize the discontinuity rate of new orders, the makespan deviation of the initial schedule and the sequence deviation. An order arrival triggers the rescheduling process. The authors propose a PSO algorithm that deals with the rescheduling, which is benchmarked against other PSO algorithms and three other meta heuristics. The proposed algorithm outperforms other solution methods in terms of average fitness value for all order types and sizes. The optimizer was not able to solve the problem within the time limit for larger instances. Therefore, the performance of the PSO could not be compared to the optimal value.
- Paprocka et al. (2021) examined a job shop production process with uncertainty in equipment downtime. A predictive-reactive ACO algorithm was developed to combine the scheduling of production and maintenance. The ACO made a basic schedule and was adapted with the minimal impact of disrupted operation on the schedule (MIDOS) rule designed by the authors. To do that, maintenance was predicted based on historical data, which led to adequate machinery inspections and an extension of machine uptime. Subsequently, the MIDOS rule assigned the most flexible operation to the bottleneck machine if a disturbance in the job is predicted to enhance the robustness of the schedule. Results showed that the MIDOS rule combined with the ACO algorithm performed better than the ACO algorithm alone. The work of Ghaleb et al. (2020) proposed a real-time scheduling model for dynamic and stochastic flexible job shops. The model accounts for unexpected order arrivals, downtime of machines, stochastic completion times. A MILP is made where closed form expressions are used to model the effect of random downtime. Different rescheduling policies are evaluated and it was found that event-driven rescheduling (ER) and continuous rescheduling (CR) performed equally well in terms of objective function. However, ER is computationally more efficient.
- The permutation flow shop with uncertainty was addressed in the literature as well. Ouchiekh et al. (2021) studied this problem while accounting for dynamic processing times. A hybrid intelligent algorithm based on a discrete eagle strategy combined with a sine-cosine algorithm was developed to solve the problem. Eagle strategy is a two stage optimization which mimics the hunting behavior of eagles. First, the search space is randomly searched globally. Then, the most promising area is searched thoroughly by a local optimizer. This process is repeated until the termination criteria are met. The sine-cosine algorithm is based on random initialization and updating solutions based on sine and cosine functions. The proposed algorithm outperformed a standard sine-cosine algorithm in terms of objective function and standard deviation for a problem instance of 11 jobs on 5 machines.
- Gupta and Maravelias (2020) studied an dynamic production schedule with uncertain processing times, batch yield and equipment downtime. This makes scheduling challenging since

these uncertainties are observed once the schedule is already executed, which can result in infeasibilities. The authors propose a systematic framework that was based on the work of Gupta and Maravelias (2019). First, the key production characteristics are quantified. Second, it is determined how these characteristics affect the choice of rescheduling time-step and horizon length. Then, the role of demand uncertainty is evaluated and how this can be mitigated through tuning of the scheduling algorithm. The rescheduling model is based on re-assigning units, re-timing of production start times and adapting batch sizes. All uncertainties are modeled with appropriate distributions. Task delays are modeled with a Poisson distribution, equipment downtime with a Bernoulli distribution and yield loss with an exponential distribution. Furthermore, they choose a finite rolling horizon because otherwise solving the problem would be computationally too expensive. For each uncertainty the effect on the simulation framework is tested. They found that costs increase with higher production and rescheduling becomes more important as uncertainty increases.

- In the work of K. Lee et al. (2019), a dynamic flow shop is examined as well where jobs enter the system over time. The authors propose a greedy heuristic to solve the problem while minimizing the makespan. They calculated the competitive ratio of their greedy algorithm for different flow shop settings. The competitive ratio measures the performance of a dynamic setting with a static setting. If the ratio is bounded, the algorithm is competitive (Atallah & Blanton, 2009). It was shown that the ratio is much tighter for permutation flow shop problems than arbitrary flow shops.

Dynamic scheduling is different from static scheduling in the way that the scheduling algorithm should consider uncertainty in job arrivals and unexpected events. This influences the performance of the model and requires more advanced solutions than static environments. Furthermore, there are fewer research articles about dynamic scheduling than static scheduling, providing an opportunity to contribute to the literature by the thesis.

2.3 Limitations of (meta)heuristics

From the literature review of existing solutions can be concluded that complex optimization problems can only be solved exact when very small instances are used. Real life situations with a larger number of jobs, machines and trucks are proven to be NP hard (Garey et al., 1976). To overcome this problem, researchers have developed (meta)heuristics such as GA, PSO and SA algorithms. These methods have to reschedule once the environment state changes. This makes it difficult for (meta)heuristics to deal with dynamic and uncertain events (L. Wang et al., 2021). Deep reinforcement learning (DRL) could solve this problem due to the advantage of real-time sequential decision making. A well trained DRL agent can decide immediately once an (unexpected) event occurs. By employing deep learning, it is able to transform high dimensional input to useful output, which is necessary in complex problems (François-lavet et al., 2018). Besides, DRL has proven to be a good solution for other optimization problems such as bin packing (Kundu et al., 2019), traveling salesman (Fairee et al., 2019) and vehicle routing (Yu et al., 2019). Therefore, in the remainder of this literature review it will be investigated if DRL can be used to solve the given problem.

2.4 Deep reinforcement learning

In this section, the concept of (deep) reinforcement learning is explained based on the works of R. Sutton and Barto (1998) and François-lavet et al. (2018). The essence of reinforcement learning is that a learning agent interacts with its environment to achieve a goal. To achieve this goal,

the agents takes actions that affect the state of the environment, which is illustrated in Figure 1. This is where reinforcement learning differs from other machine learning sub-fields. The agent is not guided which actions to take, it should try actions and find out which one leads to the highest reward by trial-and-error. Therefore, an agent does not need complete knowledge of the environment, instead it should interact with and collect information from the environment. The following three characteristics are considered the most important distinguishing features of RL.

1. RL is an iterative learning process, which makes it well suited for dynamic settings where new information becomes available over time;
2. problems are considered closed-loop since actions from the learner influence later inputs;
3. actions taken by the learner do not solely affect immediate reward, but all subsequent rewards.

This way of leaning brings certain challenges. One of them is the trade-off between exploration and exploitation. To maximize its reward, the agent should prefer the actions it took in the past which turned out to be effective. However, to discover such actions or even better ones it should try new actions. Therefore, a good agent balances exploration and exploitation. Besides the agent and its environment, four sub-elements should be explained.

1. **Policy**. This defines how an agent should behave at any given moment in time. The policy is a mapping of the environment's state with corresponding actions when that state is reached.
2. **Reward**. The objective of the agent is to take actions that ultimately maximize the reward.
3. **Value function**. the value function determines good actions in the long run. The value of a state specifies the total expected and discounted reward an agent receives in the future, while accounting for which states are likely to follow the current state and its corresponding rewards. This is an important concept in RL as this ensures in a policy that lead to the highest value in the long run instead of quick rewards. Since estimating future value is much harder than immediate rewards, finding a method to efficiently predicting values is crucial for a good RL algorithm.
4. **Environment**. A good model mimics the behavior of the environment. This enables planning future actions before they are actually taken. On the other hand, model-free methods do not use planning. Instead trial-and-error is used to consider situations.

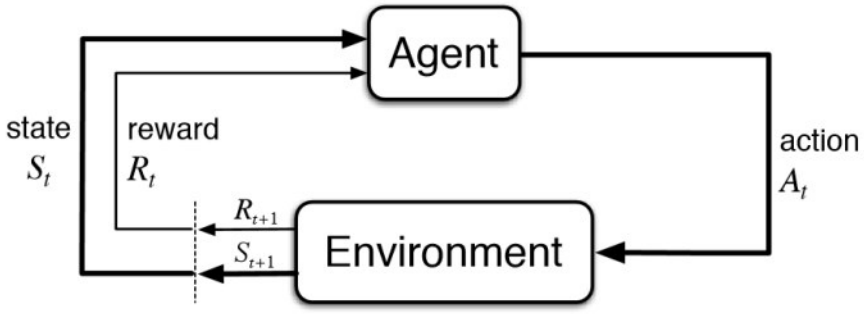

Figure 1: Agent and environment interaction in reinforcement learning (R. Sutton & Barto, 1998)

2.4.1 Reinforcement learning framework

The RL concept is formalized with a Markov Decision Process (MDP), a stochastic framework for which theoretical statements can be made. These statements are used in RL models to solve MDP's near optimally. A MDP is a 5-tuple (S, A, T, R, γ) where (François-lavet et al., 2018):

1. S is the state space;
2. A is the action space;
3. $T : S \times A \times S \rightarrow [0, 1]$ is the transition function (set of conditional transition probabilities between states);
4. $R : S \times A \times S \rightarrow R$ is the reward function, where R is a continuous set of possible rewards in a range $R_{max} \in \mathbb{R}^+$;
5. $\gamma \in [0, 1)$ is the discounter.

Since MDP's are used, the Markov property holds. This means the probability distribution of future states only depends on the current state and therefore does not depend on states in the past. For RL problems, the transition probabilities are often unknown and thus optimal policies cannot be determined based on the MDP. As mentioned before, the agent should estimate the policy based on interaction with the environment.

2.4.2 Why going deep?

The basic Q-learning algorithm (Watkins & Dayan, 1992) which will be explained below, stores a lookup table of state action pairs. To find the optimal Q-value, Bellman equations are used. Problems studied with RL often have such a large state space due to the complexity of the problem, that calculating all value functions is practically impossible. In those cases an approximate solution is more efficient. Deep Q-learning (Mnih et al., 2015) make use of neural networks to approximate Q-values for each state-action pair. Neural networks consist of an input layer, hidden layer and output layer with multiple neutrons in each layer. A deep neural network make use of multiple successive hidden layers, where each layer transforms the input in a non-linear way. The output layer has a loss function such as the mean-squared error to minimize the error. This is interesting for the problem in this thesis since the state and action space will be large.

2.4.3 Neural networks

To explain the concept of neural networks, *neurons* should be understand first. This is done based on the work of Nielsen (2021). One of the most basic forms of artificial neurons are *perceptrons* (left side of Figure 2). A perceptron takes several binary inputs $x_1, x_2, ...$ and produces one binary output. To determine the output, a *weight* $w_1, w_2, ...$ is assigned to each input variable to distinguish importance between input variables for the outcome. The weighted sum of all input variables is then compared to a *threshold value* or bias. When the threshold value is exceeded, the perceptron outputs 1 and 0 otherwise. By varying the weights and thresholds, different decision making models are produced.

Logically, one layer of neurons cannot make complex decisions since it depends solely on the input values. Therefore, multiple layers of neurons are used in a network (right side of Figure 2). In the network, the first layer makes simple decisions by weighting the model's input. Subsequently, the second layer weights the output of the first layer, enabling it to make more complex and abstract

decisions than the first layer. The third layer uses the output of the second layer, which is even more complex and so forth until the output layer translates it to the desired output value.

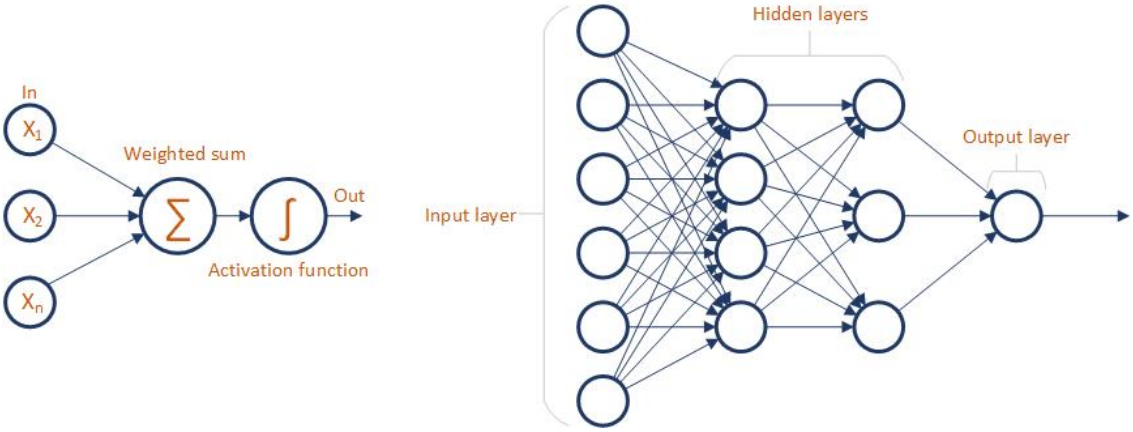

Figure 2: Perceptron & neural network (Nielsen, 2021)

The neural network is trained with data where both the input and output values are known. By knowing the output values beforehand, the neural network can adjust the weights and biases such that it minimizes the difference between the actual output and desired output. This is called the *loss function*, and can be calculated in various ways such as the mean squared error or cross-entropy loss. Commonly, this is optimized with the stochastic gradient descent algorithm, while weights are updated with back-propagation (Goodfellow, 2016). Learning works by repeatably changing the weights and biases such that the network better classifies the output value with the given input. A small change could flip a perceptron from 0 to 1 or vice versa leading to completely different behaviour in the rest of the network (Nielsen, 2021).

To overcome this problem, *nonlinear activation functions* are used such as Sigmoid or ReLu functions. What differs is that the output of a Sigmoid neuron is a float between 0 and 1, while the ReLu function maps the output in a range of 0 to infinity. In general, Sigmoid functions are used when the output is a probability. A downside of the ReLu function is that negative input values are converted to 0 and thus preventing the model from learning of negative inputs. This problem was solved by introducing the Leaky ReLu, which add a gradual negative part to the activation function (Sharma, 2020).

2.4.4 Value functions & policy gradient methods

The objective of a DRL algorithm is to find the optimal policy for the agent. As described before, this is the action it should take given a certain state to maximize the reward. There are roughly two types of algorithms, value-based and policy gradient methods. The first estimates the value of each state-action pair to find the optimal policy, while the latter optimizes the policy directly. Each type will be elaborated below.

Value-based algorithms
Value-based algorithms try to find a value function which can be used to define a policy. This is done by estimating the expected return of being in a state. The first algorithm that was based on this method was the Q-learning model developed by Watkins and Dayan (1992). In this method, a lookup table is stored with the Q-value of each state-action pair. To find the highest Q-value, the Bellman equation is used. This method is useful for small problems, though not suitable for larger state spaces where computing the Q-value for each state-action pair becomes computationally

infeasible. First the fitted Q-learning (Gordon, 1996) and then the Deep Q-Network (Mnih et al., 2015) solved this problem by estimating the Q-value of each state-action pair, thus reducing the computation time.

More specifically, Mnih et al. (2015) used a deep convolutional neural network. The introduction of neural networks brought certain challenges. Correlation in the sequence of observations, changes in data distribution due to small Q-value updates and the correlation between the action values and target values lead to instability of neural networks in RL. DQN tackles this by experience replay. This is a technique which stores all information of the last N steps where the information is obtained with an ϵ-greedy policy (Lin, 1992). This removes the correlation in the observation sequence and smooths over changes in data distribution. Furthermore, an iterative update for the action values towards the target values reduces correlation between them. When learning, Q-learning updates are applied to mini-batches of randomly drawn previous experiences with the following loss function:

$$L_i(\theta_i) = \mathbb{E}_{(s,a,r,s')\sim U(D)}\left[(r + \gamma max_{a'} Q(s^{'}, a^{'}; \theta_i^{-}) - Q(s, a; \theta_i))^2\right] \tag{1}$$

In addition, DQN clips rewards between -1 and +1 to ensure proper learning. Mnih et al. (2015) trained their DQN on several Atari games to test the robustness of the model and results showed that their model was able to learn different environments without prior knowledge and the same settings on each game.

Although the DQN by Mnih et al. (2015) was a revolutionary algorithm for DRL, it had some flaws. Since it used the maximum action value to approximate the maximum expected action value it was more likely to over-optimistically estimated values in noisy cases, resulting in an upward bias. Van Hasselt et al. (2015) tackled this problem by presenting a Double Deep Q-Network (DDQN) with two Q-functions. Each Q-function is updated with a value from the other Q-function for the next state, thus removing the bias in each function. Other variants of the DQN are proposed in the literature as well, such as the dueling network architecture (Z. Wang et al., 2016), distributional reinforcement learning (Bellemare et al., 2017) and prioritized replay (Schaul et al., 2015). Despite the various improvements on the DQN, some limitations remained. For instance, DQN algorithms are not useful for large and continuous action spaces and are unable to learn stochastic policies.

Policy gradient algorithms

Contradictory to value-based algorithms, policy gradient methods do not use value functions to find a good policy. Instead, it optimizes the policy directly by updating the policies parameters via stochastic gradient ascent. The basic form of a stochastic gradient estimator is:

$$\hat{g} = \hat{\mathbb{E}}_t\left[\nabla_\theta \log \pi_\theta(a_t|s_t)\hat{A}_t\right] \tag{2}$$

where π_θ is a stochastic policy with the observed states s_t and suggested actions a_t. $\hat{A}_t$ (the advantage) is the difference between the actual discounted reward and the estimated discounted reward from this state onward. Each iteration the advantage is updated, which enables policy gradient algorithm to update the policy parameters during and episode. The downside of policy gradient methods is that small parameter updates lead to longer convergence time, while larger updates could prevent the algorithm from finding the optimal solution. Therefore, a good policy gradient algorithm balances this trade-off. The simplest form of calculating the policy gradient estimator is by using the likelihood ratio trick (known as the REINFORCE algorithm (Williams, 1992)), which leads to the following gradient:

$$\pi_\theta(a, s) = \pi_\theta(a, s)\nabla_\theta \log(\pi_\theta(a, s)) \tag{3}$$

Another class of policy gradient algorithms are the actor-critic methods (A2C) (Konda & Tsitsiklis, 1999). The actor is specified as the policy and uses the policy gradient estimator to update the policy parameters, while the critic estimates the value function corresponding to the policy.

Therefore, one could define actor-critic algorithms as a combination of policy gradient and value-based methods. An important actor-critic algorithm is the asynchronous advantage actor-critic (A3C) (Mnih et al., 2016). While the standard A2C has one actor-learner, the A3C method combines multiple actor-learners with a CNN that has a distinct output for the policy and the value function, while all non-output layers are shared among the learners.

In the work of Z. Wang et al. (2016) the A2C method was extended by adding experience replay (ACER), making it the off-policy counterpart of the A3C algorithm. Controlling the variance and stability for off-policy estimators can be difficult as the policy gradient is the product of (among others) unbounded weights. To overcome this problem, the authors propose a new importance weight truncation technique. For continuous action spaces, they needed another improvement since the integration of the state-action values cannot be used to derive the value functions. Therefore, stochastic dueling networks are introduced to estimate both values off-policy while ensuring consistency between the two estimates.

Additionally, there are natural policy gradients algorithms, introduced in reinforcement learning by Kakade (2002). The main difference with policy gradient algorithms is that natural gradients use the steepest direction of the Fisher metric instead of the steepest direction in the parameter space to get to the optimal solution. This prevents the policy from remaining in local optima although it is much harder to compute, making it unsuitable for practical usage. To overcome this problem, Schulman et al. (2015) developed the Trust Region Policy Optimization (TRPO). TRPO constraints the size of a policy update with the idea that the updated policy should be close to the current policy where results are known to be acceptable. Schulman et al. (2015) proves that the maximum update size is a term that depends on the KL divergence. The KL divergence measures the difference between two probability distributions (Kullback & Leibler, 1951). Therefore, this term is subtracted from the gradient estimator:

$$max_\theta \quad \hat{\mathbb{E}}_t \left[\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)} \hat{A}_t - \beta KL[\pi_{\theta_{old}}(\cdot|s_t), \pi_\theta(\cdot|s_t] \right] \tag{4}$$

Note that the first term has changed to the policy divided by the old policy, instead of the logarithm of the policy. Although TRPO coverts faster to an optimal solution, it is still quite hard to implement as β cannot be fixed (making it a multi optimization problem) and the constraint leads to extra computation time (Schulman et al., 2017). The Proximal Policy Optimization (PPO) (Schulman et al., 2017) aims to solve this problem. Instead of penalizing the size of the policy update, it clips the probability ratio $r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)}$ within the interval $[1-\epsilon, 1+\epsilon]$. Leading to the objective:

$$L^{CLIP}(\theta) = \hat{\mathbb{E}}_t \left[min(r_t(\theta)\hat{A}_t, clip(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t) \right] \tag{5}$$

Then, the minimum of the clipped and unclipped reward is used as lower bound of the final objective. Since the neural network uses parameters from the policy and value function, an extra loss function which combines the policy surrogate and value function error is subtracted. Last, an entropy term is added to ensure exploration resulting in the following objective:

$$L^{CLIP+VF+S}(\theta) = \hat{\mathbb{E}}_t \left[L_t^{CLIP}(\theta) - c_1 L_t^{VF}(\theta) + c_2 S[\pi_\theta](s_t) \right] \tag{6}$$

The PPO algorithm was able to outperform other online policy gradient methods on Atari games and other benchmark tasks. Still, the algorithm is not perfect and criticisers note flaws in design choices. Hsu et al. (2020) argued that general design choices do not lead to the best results in every setting and proposed two adaptions for specific cases. For discrete action spaces KL divergence objectives make PPO more robust, while for continuous action spaces beta distribution lead to better results than Gaussian policy parameterization.

2.4.5 Training a deep reinforcement learning agent

Once a deep reinforcement learning algorithm is chosen, it should be trained on a specific task. To guide the agent towards the optimal policy, several techniques can be applied. First, action masking is explained, followed by hyperparameter tuning.

Action masking
A disadvantage of reinforcement learning is the relatively long training phase. Especially in more complex environments with a large action space and multiple actions, such as the flow shop scheduling problem in this study. In these cases, the major part of the action space is not possible in a given state. For instance in flow shop scheduling, in the state where machine m is idle and all other machines are occupied, all actions that assign job j to any machine except machine m are invalid. When the agent chooses an invalid action, the state does not change and the environment proceeds to the next step. In these settings the agent does not learn when an invalid action is chosen because the state does not change. This is done in the field of scheduling by Brammer et al. (2021), Mao et al. (2016) & W. Chen et al. (2017). From their results can be concluded that these RL algorithms take relatively longer to convergence than algorithms with some form of smart action selection. Another option is to design the action space in such a way that all actions are valid in every state. For instance by choosing between scheduling heuristics (Ren et al., 2021), (Marchesano et al., 2021) & (H. Wang et al., 2021).

Nevertheless, when one wants to implement an algorithm that actually assigns jobs to machines instead of choosing the heuristic there will be invalid actions in some states. One of the methods for guiding the agent towards the optimal policy is penalizing invalid actions (Cals et al., 2021). Here, a (small) penalty is included in the reward function of the algorithm. Once the agent chooses an invalid action it receives negative feedback by means of the penalty. In this way, the agent learns which actions are invalid in a particular state. A drawback of this method is that it can take a long time before the agent has learned all invalid actions per state. A relatively new approach for this problem is the masking of invalid actions. With this method the invalid actions in a given state will be 'invisible' for the agent. The agent can only select visible actions and thus the chosen action will always be valid. To use action masking, the agent and environment have to be modified. The environment should indicate which actions are masked for each state, this can be hand-crafted or estimated by an algorithm. The agent has to understand which actions are masked. Often, this is done by a binary value for each action where 0 corresponds to invalid and 1 to valid.

Action masking can be used for both value-based and policy gradient algorithms. (Zahavy et al., 2018) used action masking for DQN in a game environment. They implemented contextual multi-armed bandits which estimates if an action is valid in a given state. The masked algorithm found an optimal policy faster than unmasked comparable algorithms. Furthermore, Wu and Rasmussen (2019) developed a trainable-action-mask (TAM) to predict if an action should be masked. Their model predicted the similarity of consecutive states and classified the corresponding action as invalid if the similarity exceeded a threshold. Policy gradient algorithms can be masked as well.

In the work of Tang et al. (2020), action masking is implemented in a PPO algorithm. Before masking can be used in PPO algorithms, it needs two modifications. First, only valid action should be included in the observation space. Second, during the stochastic ascent only valid actions should be used in the search. Furthermore, the probability of valid actions need to be normalized again in the output layer of the neural network. To test the performance of the maskable PPO algorithm it was compared to the original PPO algorithm. Results showed that action masking lead to better results in terms of reward function and computational time. Since the production and distribution planning at premix producers exist out of multiple actions, the action space will be relatively large. Consequently, there will be many invalid actions in some states and thus action masking could be convenient in this case.

Hyperparameter tuning
Hyperparameter tuning is important for DRL algorithms and can be time consuming. It is not only important for the algorithm itself but for comparison with other algorithms as well. Especially the neural network architecture, learning rate, reward scale, training discount value have a great impact on the results (François-lavet et al., 2018). Henderson et al. (2018) discussed the role of each aspect on the outcome of a DRL algorithm. For the neural network design, ReLu and Leaky ReLu are the best activation functions overall, although there are differences across DRL algorithms. This shows how the algorithm is interconnected with the neural network design. For instance, changing the neural network of a PPO algorithm may require changes in the trust region clipping or learning rate to compensate for architectural changes. Furthermore, reward scaling affects the results as well. In particular layer normalization influences the result of reward rescaling, which indicates that neural networks and gradient-based methods are the cause of this phenomena. The authors showed that the source of the algorithm has a great impact on the outcome as well.

The actual tuning of hyperparameters itself can be done in many different ways. A grid search is one of the easiest ways but depends on luck and experience of the designer (Rijsdijk et al., 2021). Other options include automatic hyperparameter optimization (HPO) (B. Zhang et al., 2021), Bayesian Optimization (BO) (Snoek et al., 2012) or another DRL algorithm (Rijsdijk et al., 2021). In the work of B. Zhang et al. (2021) the importance of hyperparameter tuning was once again underlined. Furthermore, they found that their HPO performed much better than manual hyperparameter selection. In the Bayesian Optimization, the performance of an algorithm is modeled as a sample from a Gaussian process. This leads to efficient parameter selection for the next experiments based on the results of previous experiments (Snoek et al., 2012). Again, this way of hyperparameter tuning outperformed manual selection.

Another method of tuning the algorithm is by means of reward shaping. The reward function is an excellent way to include domain knowledge into the algorithm. However, expressing knowledge in numbers is a difficult task due to cognitive biases (Hu et al., 2020). Basic reward shaping modifies the original reward function by adding a shaped reward function ($r^{'} = r+F$, where r is the original reward, $r^{'}$ the modified reward and F the shaped reward). In early work, researchers focused on the additive part, ignoring the fact that this may change the optimal policy. Hu et al. (2020) proposed a new way to shape the reward by implementing parameterized reward shaping. They introduced a weight z_ϕ to the equation, which weights each state-action pair. In this way, a distinction can be made between beneficial and unbeneficial rewards. This results in a bi-level optimization problem where the policy should be optimized given $r^{'}$ and then z_ϕ should be optimized such that the acquired optimal policy maximized the reward as well. The results showed that the parameterized reward shaping method was able to utilize beneficial reward shaping while ignoring unbeneficial reward shaping, which improved the overall results of the algorithms compared to the baselines.

2.5 Deep reinforcement learning applications

An overview of deep reinforcement learning applications in the literature will be given here. This includes scheduling problems and applications in other domains.

2.5.1 Scheduling

Deep reinforcement learning is used in production scheduling studies. Zhou et al. (2020) proposed a DQN algorithm for dynamic scheduling in a smart manufacturing setting. The main reason for choosing DRL was because DRL can deal with uncertain events. Smart manufacturing is similar to job shop scheduling due to the large number of tasks, dynamic sate of services and uncertain events. The model tries to minimize the makespan of all jobs while the system state is modeled

as the queue times of all jobs. The agent should choose between scheduling rules when a job arrives. These scheduling rules are Shortest Processing Time (SPT), Longest Processing Time (LPT), Shortest Queue Time (SQT) and Longest Queue Time (LQT). Once the agent choose a scheduling rule, the reward and next state is observed. In the case study, a fairly basic problem is considered of 3 job types and 5 machines to illustrate how the model works. No benchmark was used and therefore the performance of the model could not be assessed.

The classical job shop scheduling problem was studied by L. Wang et al. (2021) and extended by dynamic events such as machine breakdown and job rework. To deal with the high dimensional state and action space, they use the proximal policy optimization (PPO) instead of a value-based method. The PPO algorithm was chosen because it has been proven to obtain good results by limiting the policy update and reduce the sensitivity of parameter settings. Furthermore, the objective of the model is to minimize the makespan. The authors model the job shop problem as follows: the environment is a set of job lists with assigned machines and processing times. The state space is the processing status of each job at the designated machine and the action is to select jobs for idle machines. After the action is done, the agent receives the reward function which is the machine utilization and makespan. The state, action and reward are stored in a buffer. The policy is updated according to the stochastic gradient ascent method by mini-batch samples from the buffer. The DRL algorithm performed better on average than meta-heuristics and obtained solutions not higher than 10% of the optimal solution. Furthermore, to test the generalization processing times and machine sequences were changed. The trained optimal policy of the first experiment was able to find an optimal schedule in seconds. The GA did find a better solution however it took 28 seconds to find it. This indicates that the DRL model can adapt to changing environments more quickly.

A permutation flow shop scheduling problem was examined by Brammer et al. (2021) which included multiple production lines and demand plans, however no uncertainty was included. The action to be taken by the agent is to decide which job should be sequenced at position t. Afterwards the environment goes to the next state, which ensures the policy is trained in an iterative way. When the agents schedules a job to an invalid position the reward will be 0 and the state remains unchanged. The reward is based on idle time of a machine while the state contains information about the remaining jobs and remaining processing times on all machines. Furthermore, PPO is used for policy learning combined with a policy gradient method instead of a DQN. This is done because policy gradient methods guarantee convergence to a local optimum and allow learning of a stochastic policy. The method outperformed heuristics and was just 0.42% worse than the optimal solution.

Marchesano et al. (2021) proposed a deep reinforcement learning model to solve a flow shop scheduling problem. A DQN was employed to approximate the value function and the representation of the states and actions. Goal of the agent is to choose the best dispatching rule for a given state. This is difficult since there is not one dispatching rule that is superior in every situation. Back propagation is used to calculate the error by means of the loss function. The reward function is based on the throughput of the production line. The state of the system is modeled as the job characteristics waiting in the queue and the line's current production status. The action of the agent was to choose between three dispatching rules, first in first out (FIFO), shortest processing time (SPT) and latest processing time (LPT) Results showed that the agent chooses the SPT rule in most of the cases.

S. Lee et al. (2020) applied DRL to schedule the production process of injection mold. This process differs from the compound feed industry since each product has a different manufacturing process. Yet, it shows similarities in the production process as their shared objective is to minimize the tardiness, products are mostly made to order and sequence dependent setup times occur. Furthermore, the authors included a dynamic environment in their study. A MDP was proposed to transform the problem to a DRL model. They included a set of idle machines statuses in

the state, which is the setup type per machine and dedicated operation type. Furthermore, the waiting jobs are included. The action of the agent is to choose which job is assigned to an idle machine. The reward function is the weighted average processing time since the actual completion time is unknown when the agent acts. A DQN was used to solve the case and results showed that the DRL model outperformed other dispatching rules once trained without retraining. However, retraining was needed when the job types, machines and job sequences changed.

Kardos et al. (2021) studied a Q-learning DRL algorithm to solve the dynamic scheduling problem in a job shop production process. The states included information about the job type and the number of jobs in the system. With this information the total process and setup times can be derived. The agent should allocate a job to a machine in order to minimize the total lead time. The reward of the model is based on the lead time. The actual lead time is compared to a baseline lead time. Again the DRL algorithm was compared to dispatching rules and results showed that the DRL algorithm reduced the average lead time compared to dispatching rules and became more beneficial for more complex production processes.

In the work of Luo et al. (2021) a multi-objective flexible job shop scheduling problem is solved with a DQN algorithm. The objective is to minimize the weighted tardiness and machine utilization. The authors extended the original DQN by developing a two hierarchy DQN. The higher-level DQN determines the temporary goal of the lower-level DQN. The lower-level DQN combines the current state with the temporary goal to choose a dispatching rule. In the state, the number of machines, due date tightness and the expected inter arrival time are included. Together with information about the machine utilization, completion rate of jobs and tardiness. Actions include various dispatching rules at each rescheduling point. The reward function consists of four different goals including estimated total weighted tardiness, actual tardiness, estimated tardiness and average machine utilization. The higher-level DQN adaptively selects different goals at different rescheduling point to balance the two objectives. During training, a new job arrival is defined as a rescheduling point and is based on the DDQN of Van Hasselt et al. (2015). The algorithm was compared with classical dispatching rules and other RL based methods. Results showed that the proposed algorithm outperformed the dispatching rules and on most instances the other RL methods.

Ren et al. (2021) examined a flow shop scheduling problem and proposed a RL algorithm to solve it. The state included the ratio of waiting jobs per machine, mean processing time of jobs in the queue per machine, the ratio of minimum and maximum processing time of jobs in the queue per machine and the ratio of remaining processing time of the jobs per machine. Actions consists of choosing the dispatching rule that leads to the minimization of the makespan. Therefore, the reward function is about the makespan and is the negative summation of the idle times of all machines. A Sarsa algorithm was used to solve the problem and showed good performance even with bigger instances than the trained set.

In the work of Hoon Lee and Lee (2021), DRL was used to schedule the production in a semiconductor factory. The state denotes information about idle machines, current setup type, number of setup changeovers, number of jobs in the queue, target production and actual production. The agent should decide which job to process next on each machine and the reward is designed such that more reward is received when the agent chooses an idle machine to process next. Furthermore, negative reward for setup times is included. A DQN was used to solve the problem and results showed that the algorithm performed equally good compared to dispatching rules. However, with new instances the algorithm was still able to produce a good schedule. Therefore, the algorithm could be used for different production processes as well.

The lot scheduling problem is somewhat compared to flow shop scheduling with sequence dependent setup times since setup times are of great importance with lot scheduling as well. Here standardized products should be scheduled on a single machine that has significant setup times

between product types. Therefore, producing more of one product at a time is more time efficient than producing according demand. Rummukainen and Nurminen (2019) studied this problem and tried to solve it with a PPO algorithm. They modeled their system as a Semi-Markov decision process (SMDP) instead of a regular MDP. In SMDP the state changes once an event has occurred, while in MDP the state changes after a predefined time interval. This could enhance the efficiency of the algorithm since SMDP require the algorithm to decide upon an action once the system state changes and thus an action is required.

In the case of Rummukainen and Nurminen (2019) this is when an item has been produced or when an order arrives during idle time. They model the state as the inventory level per product type and the agent can choose to produce a product or leave the machine idle. A continuous time discount factor was added to the advantage factor to make it more suitable for SMDP modeling. Furthermore, they excluded the entropy term in the loss function as they found it ineffective in practice. In the experiments they compared multiple PPO algorithms and they found that one of their algorithms outperformed an older version. Although they mentioned it was hard to find a good neural network architecture and hyperparameters for their problem. This led to the conclusion that DRL is best suited for problems where no existing scheduling method is satisfactory.

Hubbs et al. (2020) designed a DRL model to schedule a dynamic chemical production process. The authors used an Advantage-Actor-Critic (A2C) algorithm to solve the problem which is a policy gradient algorithm that used both value function approximation (critic) and policy function (actor). A prediction of the future inventory is used to decrease the action space. Furthermore, the state includes information about demand, forecast of demand, one-hot encoded current schedule and simulation time. The reward is the same as the objective of the mathematical model, namely the profit of the schedule. This includes the discounted standard margin times the shipped amount per order minus the costs for inventory. Results showed that, once trained, the DRL model outperforms other more computationally intensive scheduling methods. Moreover, this holds in new instances, showing the robustness of the model. However, there are drawbacks as well. There is no guarantee that a global optimum is found and the model required many training sampled to learn a good policy.

H. Wang et al. (2021) studied a combined job scheduling with preventive maintenance problem and proposed a Q-learning algorithm to solve the case. The agent should choose between four dispatching rules for job scheduling and should choose when to execute preventive maintenance. The state therefore contains information about the mean normal processing time and estimation of mean lateness of remaining jobs. The reward function consists of immediate and final reward. Immediate reward is the ratio processed jobs divided by the makespan, whereas the final reward is the difference between the objective value in the current episode and the historically optimal objective value. Results showed that the Q-learning algorithm performed better than other maintenance strategies.

From these articles can be concluded that DRL algorithms often perform better than meta-heuristics in dynamic environments. Furthermore, DRL algorithms are able to schedule comparable scenarios once trained, while other solution approaches require retraining. Besides, it is clear how current studies model the states, actions and reward. In the state space, the jobs in the queue and resources status are often included. The actions consists of either actual assigning jobs to machines or a scheduling heuristic. Reward functions have some component for tardy or on time delivered orders, makespan, throughput and a penalty for setup times. Furthermore, PPO algorithms are often used, instead of value based algorithms due to their fast convergence and ability to deal with high dimensional state and action space. Last, modeling the environment as a SMDP seems interesting due to the fact that it requires less unnecessary state transitions, thus less computational time.

2.5.2 Other applications

Deep reinforcement learning was used in other operational domains as well. Although these problems are not the same as the problem examined in this thesis it is still useful to review these articles. Cals et al. (2021) studied an order batching and sequencing problem (OBSP). Here it should be decided how (picked by order or by batch) and when orders should be batched and picked in a warehouse while minimizing the number of tardy orders. The algorithm contains two components, the sequencing decisions are done by heuristics while a DRL decides upon the picked-by-order or picked-by-batch question. Since their model heavily depends on time, a semi-Markov decision process is used. This relaxes the fixed transition time to a next state constraint. Now, the transition time is triggered by the arrival of a new order and the agent can react accordingly.

The state space contains information about remaining orders, available capacity, tardy orders, number of processed orders and simulation time. Actions include the decision if an order should be picked by batch or order. The reward function has a small penalty component for infeasible actions and tardy orders, while the ratio processed orders against all orders accounts for the largest part of the reward. The PPO algorithm was used to find a (near) optimal policy. To compare the DRL algorithm, several heuristics were tested as well. Results showed that the proposed algorithm outperformed the heuristics on most instances. Furthermore, the trained model was able to find equal results on new instances compared to agents which were trained on these instances showing the robustness of the proposed algorithm.

In the study of Farahani et al. (2021) a DRL algorithm was used to allocate containers to trucks and trains in a sequential decision making problem. Goal was to minimize the costs while concerning the capacity of the trains. Costs of using a truck is far greater than using a train. Therefore, the algorithm should assign as many containers to trains as possible. Furthermore, unlimited truck capacity was assumed. The state contains information about train capacities with arrival and departure time, and information about the next container that must be allocated including earliest available transportation day and due date. This container is selected by means of a heuristic. Furthermore, the actions include the allocation of a container to a train or the unconstrained truck.

The reward function consists of costs incurred with choosing a certain action. First, if the truck or a train which leads to a delay is chosen, the costs are transportation costs by truck. Second, if a train is chosen that delivers the container on time only the train transportation costs are incurred. Note that transportation costs by truck are much higher than delivery by train. The optimal policy is searched by a DQN. Furthermore, action masking is used to mask infeasible actions such as allocating containers to trains without capacity or trains which departed before the container arrived. The actual action is then chosen by a customized epsilon-greedy method. Besides, replay memory and mini-batch is used to decrease the update variance. To address the performance of the model it was compared with an earliest due date and first in first out heuristic. Results showed that it outperformed both heuristics and was just 0.63% worse than the optimal model in the high uncertainty case. This shows the potential of DRL in uncertain environments.

DRL applied to preventive maintenance in a serial production line is studied by J. Huang et al. (2020) as well. The state is defined by the machine age which specifies the probability of random failures, the buffer level of each machine and the remaining maintenance duration for ongoing maintenance per machine. The agent should decide when to turn off a machine and perform preventive maintenance. Furthermore, the reward exist of incurred costs corresponding to maintenance and profit loss due to production loss. A DDQN is used to schedule the maintenance. The authors normalize the input and output of the NN to ensure all values have the same scale. The algorithm was compared to various maintenance heuristics and results showed that it outperformed the heuristics for all instances.

In the work of Mao et al. (2016), DRL was used for dynamic multi resource cluster scheduling. Jobs arrive in the system over time and the agent should allocate jobs to resource clusters, where each cluster is treated as a single machine. The objective is to minimize the normalized makespan, i.e. completion time divided by the ideal duration. According to the authors, normalizing the makespan prevents a bias towards large jobs. In the state space, the current allocation of jobs to clusters and the required clusters by waiting jobs are included in images which serve as input for the neural network of the DRL algorithm. To have a fixed state representation, only the longest waiting m jobs are maintained in images. Other arriving jobs are stored in the backlog of the state space. This appears to be a good solution since longer waiting jobs should be processed first.

An additional advantage is that the action space is constrained, which makes the learning process more efficient. During the action phase, the time is frozen until the agent chooses otherwise. The action space consists of choosing which job to process next and the agent is allowed to schedule multiple jobs at once. When the agent chooses an invalid action or wishes to stop scheduling during this timestep, time proceeds and newly arrived jobs are revealed to the agent. The agent only receives a reward at the end of an action and the reward function is minus the sum of the ideal completion time of all jobs in the system (in process and waiting). A variant of the REINFORCE algorithm was used to solve the problem and results showed that the algorithm is comparable to heuristics. W. Chen et al. (2017) improved the work of Mao et al. (2016) by extending the model with a new reward function and a convolutional input layer. In the new reward function, a greater penalty is given to waiting jobs. It appeared that the model was able to find better solution together with a faster convergence rate.

The storage space allocation problem can be compared to the bin packing problem according to Boland et al. (2011), which is known to be NP-hard (Garey & Johnson, 1979). Even though multiple heuristics have been proposed to solve the bin packing problem, there is not one general heuristic that works well on all instances (Gomez & Terashima-Marín, 2018). Often single heuristics are used to select the next item to be placed, the bin and the position of the item inside the bin. Therefore, bin packing problems generally exists of two sub-problems: one for item selection and one for item placement. Since only the first sub-problem is applicable to the problem of this thesis, the latter will not be discussed. First of all, the first fit heuristic which considers all open bins in a fixed order and places the next item in the first fitting bin. Variants on the first fit include sorting all items and then start with largest or smallest item. Furthermore, the next fit heuristic places the next item in the current bin, if that does not fit a new bin is selected. Again, all items can be sorted first by largest or smallest item. Other methods are the best/worst fit heuristic which places the item in the best/worst fitting bin. This can be combined with sorting the items first in a decreasing or increasing order.

Ouhaman et al. (2020) studied a storage space allocation problem in a bulk material setting. Bulk material had to be stored in six identical hangers with the same capacity in a seaport. Their heuristic sorts the products based on the arrival time and subsequently chooses the storage space that has the least space left such that it fits in the storage area. The heuristic was tested on a large dataset and was able to find a quick solution (within one minute) although it was not near the optimal solution.

DRL is used in inventory management as well. De Moor et al. (2022) studied how potential-based reward shaping can improve the use of DRL in perishable inventory management. Reward shaping adjusts the reward of the model to guide the algorithm towards the desired outcome. This reduces the sensitivity of the algorithm. Results showed that reward shaping can transfer knowledge embedded in the heuristic that is used to teach the reward shaping algorithm, as long as the teacher performs well on the given problem. Furthermore, reward shaping improves the learning process since the objective function is better and variability is lower compared to unshaped models.

2.5.3 Offline reinforcement learning

One of the flaws of reinforcement learning is the dependence on interaction with the environment to learn the optimal policy (R. Sutton & Barto, 1998). This online learning is impractical when the agent is trained in the real world since it will make sub-optimal decisions in the early stages of training, which can be expensive. Training the agent with a simulator overcomes this problem although this has flaws as well. Building a realistic simulator can be time consuming and difficult, especially in complex environments. Besides, flaws in the simulator will be exploited by the agent if beneficial. Offline reinforcement learning could be a solution for this problem since it utilizes previously collected data to update its policy, without interacting with the environment. The dataset should include information about the transitions of the MDP, $D = \{(s_t^i, a_t^i, s_{t+1}^i, r_t^i)\}$. Note that s_t^i is the state, a_t^i are the actions and r_t^i is the reward in each transition. The offline reinforcement learning algorithm has to learn the best policy based on the dataset, which is similar to classical supervised learning (Levine et al., 2020).

At first glance this seems a promising way to deal with the largest problem of online reinforcement learning. Unfortunately, challenges arise with offline reinforcement learning as well. To start, the quality of the algorithm heavily depends on the quality of the dataset and exploration is not possible. If there are no transitions of high-reward regions in the state space, the algorithm will not explore these regions. Furthermore, the goal of offline reinforcement learning is to find a different policy than the policy in the dataset. This is contradictory to current machine learning techniques where the goal is to find a distribution that matches the distribution of the training dataset. As a result, the corresponding reward to a new policy remains unknown during training (Agarwal et al., 2020).

Offline reinforcement learning algorithms are mostly applied to fields where online training is difficult. One example is the application in visual learning for robots in real world environments, which requires a massive dataset with all sorts of real world objects (Russakovsky et al., 2015). Healthcare is another domain where offline reinforcement learning is well suited due to the consequences of errors. Although this brings some extra challenges such as survivor biases in the dataset (Gottesman et al., 2019). In the work of L. Wang et al. (2018) medical records were used as input for an offline actor-critic algorithm to determine drug recommendations. Furthermore, offline reinforcement learning was used in combination with online reinforcement learning in the work of Nair et al. (2020). They combined an offline sample-efficient dynamic programming algorithm with online fine tuning of the policy by maximum likelihood updates. The algorithm was able to converge faster and find better policies than other off policy and offline reinforcement algorithms.

To the authors knowledge, Gabel and Riedmiller (2008) is the only study which used some form of an offline reinforcement learning algorithm on the job shop scheduling problem. Their algorithm was not fully trained offline, they used offline reinforcement learning to have less interaction with the environment. First, a MDP transition dataset was generated by interaction with the simulation model. Then, an adapted version of the Q-learning algorithm was used to find a policy on the dataset. Subsequently, the policy was updated by regular Q-learning, thus by interaction with the simulation model. The adapted Q-learning algorithm outperformed dispatching-rules. Furthermore, it performed better than the benchmarks on unknown scenarios.

Q-learning tend to overestimate the value functions due to the different distribution in the learned policy compared to the dataset (Kumar et al., 2020). To overcome this property of Q-learning algorithms at offline training, some adaptions are proposed in the literature. Kumar et al. (2020) adapted the Q-learning learning algorithm such that it learns a lower bound of the policy value and named their algorithm the conservative Q-learning algorithm. Furthermore, Fujimoto et al. (2018) proposed a batch-constrained Q-learning algorithm, which limits the action space to guide the agent towards a near-optimal policy.

Offline reinforcement learning has some useful properties for the case of the thesis. To start, building a simulation model is time consuming and prone to errors. Offline reinforcement learning could solve this problem when a dataset with MDP transition is available. Furthermore, online training requires interaction with the environment. This can be difficult for premix manufactures since there is no room for errors during the early stage of training. Especially offline reinforcement learning algorithms that deal with overfitting are interesting for this thesis. Moreover, little research has been done in applying offline reinforcement learning to scheduling problems which leaves a gap in the literature.

2.6 Research gap & contributions

The literature review forms a basis and academic motivation for this thesis project. It can be concluded that there are no existing studies which tackled the exact same problem, i.e. production scheduling with truck and storage allocation in a dynamic setting at premix feed producers. Furthermore, little research has been done on the integrated production and distribution scheduling problem. Hou et al. (2022), Leung and Chen (2013) and Mortazavi et al. (2015) proposed different meta heuristics to solve the problem. However, the instances were relatively small and not one of them included storage of finished products between production and delivery. Moreover, none of them included a dynamic environment. Leaving a clear gap about research in dynamic integrated production and distribution scheduling.

During the literature review, separate parts of the dynamic integrated production and distribution scheduling problem were investigated to get an understanding of the problem and find current (meta) heuristic solutions. Although the proposed solutions perform well for the studied problems, they are not suitable for dynamic problems with a high dimensional environment. DRL appears to be a promising technique to overcome this problem since it can handle large state spaces and deals with dynamic events through decision making over time. Especially the PPO algorithm (Schulman et al., 2017) could be a good method to solve the production and distribution scheduling problem due to its ability to tackle bigger state spaces compared to DQN (Mnih et al., 2015) and other DRL algorithms. A downside of DRL is the design of states, actions and rewards, which heavily influence the performance of the model. This should be accounted for in the environment design in a later stage of the research. Moreover, there are little DRL studies who included multiple actions in their model during the production scheduling process. Therefore, this research aims to make two scientific contributions.

1. We are the first to study the integrated production and distribution scheduling problem, including sequence-dependent setup times in a dynamic and uncertain environment. This makes the study more realistic as compared to offline equivalent studies. Moreover, larger instances are included than previous studies, which cannot be solved exact.
2. A novel way to solve this problem will be studied by means of a deep reinforcement learning based solution. This includes multiple decision moments in the production process and distribution schedule.

Furthermore, it should be mentioned that the conducted literature review is by no means an exhaustive review of the existing literature on the integrated production and distribution scheduling problem and deep reinforcement learning. Instead, it provides a broad review to get a better understanding of the topic.

3 Problem description

This section elaborates on the brief problem statement that was provided in the introduction. First, the current production and distribution processes at a typical premix feed producer are explained. Followed by a description of the current production and distribution scheduling process. Then, the challenges in this problem are discussed, after which the scope of the thesis is defined. Last, the problem is formulated as a mathematical model.

3.1 Production and distribution process

The production of premix can be described as a permutation flow shop production process. A flow shop is referred to as an ordered set of stages in a production setting such that the first operation of each job is executed at stage 1, the second at stage 2 until the m^{th} operation at stage m (Garey et al., 1976). Furthermore, a flow shop production process is called permutation flow shop if the processing sequence of jobs cannot change from one machine to the next (Rossit et al., 2018). Each stage represents a machine in the production process of premix, which performs an operation on (raw) materials to make the final product. The customer orders form the production requests. These requests are defined as jobs in a flow shop setting. Before production starts, suppliers deliver raw materials to the production plant. Delivery is done in bulk and bags. Examples of raw materials are vitamins, minerals, amino acids and fillers. Ingredients that are used for many products are delivered in large quantities and in bulk. These materials must be weighed with dosing and weighing equipment (possibly with Alfra) before storage in silos until usage. Raw materials that are used less often are stored in bags.

An overview of the production process is given in Figure 3. A real production plant is built in height because downward transportation of materials can be done by gravity. This reduces energy usage of the production plant and therefore decreases the production costs. Production starts with dosing raw materials according to the product's recipe. Advanced dosing machinery, such as Alfra, are used to assure the final product has the same proportion of ingredients as the recipe. The dosed materials are then sequentially processed on a production line with different machines (stages) to manufacture the product. First, ingredients are mixed with each other at the mixing machine. Here, bag ingredients are manually added if required. Mixed ingredients are stored in a buffer silo after mixing. Then, depending on the desired packaging, products are either filled in small bags or big bags and is called sacking. If a customer requests bulk delivery, the product is temporarily stored in silos upon loading to bulk trucks. Bagged products are stored in a warehouse on pallets. The premix production process is done in batches, which means that the whole batch has to be finished before a new batch can be processed. Dosing takes about 45 minutes and multiple dosing machines operate at the same time to make sure the mixing machine is not idle when recipes are dosed. Mixing takes about 6 minutes and it can process 2000 kg per batch. Most factories have multiple mixing machines operating in parallel. Sacking is a continuous process where 10.000 kg / hour can be placed in small bags and 8.000 kg / hour can be placed in big bags.

As stated before, a part of the distribution is done in bulk trucks. These trucks have different compartments to prevent contamination during transport. Often, trucks transport multiple orders and thus have to visit multiple customers in one ride. To precisely load the trucks in terms of product quantity, again dosing machines are used. Loading can be done directly from finished product silos or via so called 'contrabins'. A contrabin is a smaller silo located right above the hatch of a truck's compartment in the loading street. Direct loading is done with either movable dosing and weighing machines or with a weighbridge. The machine doses the correct amount of product from a silo and transports it to the truck. Once arrived at the truck, the weighing machine is placed above the correct compartment and unloads the product directly in the truck. Loading with weighbridges is the cheapest and the longest option. Here, no dosing and weighing machines

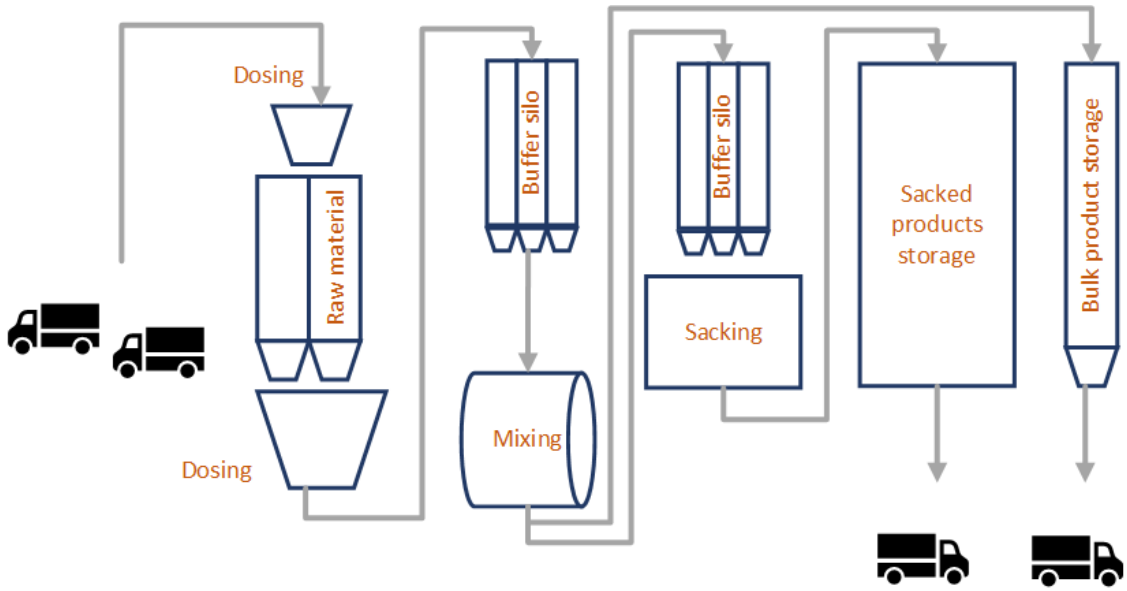

Figure 3: Production process compound feed producer

are used. Instead, finished product is loaded manually in a compartment until the truck driver thinks the correct amount is loaded. Then, the truck has to weight on the weighbridge to determine the actual loaded quantity of finished product. Logically, this process should be repeated until the requested quantity is loaded. This way of loading is the least precise method. Contrabins are used to accelerate the loading process and is costlier than direct loading. Products that are scheduled for the next truck are loaded into the contrabins in advance. For each compartment of a bulk truck there is one contrabin. In this way, the truck only has to park in the loading street which is below the contrabins and loading all compartments can be done after one another without intermediate dosing of products by a dosing machine. Once loaded, the truck delivers the products to the customers.

3.2 Current planning process

The production and distribution processes are scheduled by planners of the premix producer. Currently, the distribution planning is scheduled prior to the production planning and this is often done with third party optimization software. This can be compared to the predetermined truck departure time scheduling integration definition from Karaoğlan and Kesen (2017), since the truck's departure time is not adapted at rescheduling. Instead, the production schedules are adjusted to the distribution schedule. Input for the distribution planning are the orders, which have a due date, delivery location and required quantity. The software assigns orders to trucks and decides upon the departure time at the production plant such that all orders are delivered on time. Orders do not necessarily contain enough products to fill up exactly one truck, therefore trucks may contain multiple orders. Furthermore, delivery is done in nearby regions, thus trucks can do multiple rides on one day. When completed, the distribution planning is loaded into the Promas ST software system. Promas ST shows the latest production finishing time of a product such that it can be delivered on time. Based on the distribution planning, planners schedule the production orders manually. Scheduling includes deciding whether requested products should be produced or retrieved from stock. If a product has to be be produced, the planner decides when to produce and on which mixing line and sacking line. Furthermore, planners should allocate produced products to storage places in the warehouse or silos and compartments of the truck.

The Promas ST software assists the planner by visualizing the chosen production order in a Gantt chart and by showing if due dates are met with the current planning in a list. Besides, planners should schedule production of MTS products. MTS production orders arrive when inventory of a particular product is below its reorder level. These products are often produced during nighttime when no customer orders arrive.

3.3 Desired situation

Due to the complexity of the scheduling problem manual (re-)planning leads to sub-optimal schedules. This is reflected in the daily operations as trucks have to wait for production to finish and vice versa. As a consequence, customers of KSE desire an integrated and automated production and distribution planning. This has been proven to enhance operational efficiency (Yağmur & Kesen, 2020). After all, profit margins are small in this industry, thus adequate use of resources can lead to competitive advantages for premix producers. Previous attempts by KSE to integrate production and distribution schedules have failed so far. Customers of KSE are therefore sceptical that this is possible for such a complex scheduling problem. KSE wants to investigate if combining these schedules is possible so they can implement this in their factory automation software package Promas ST.

3.4 Challenges

Combining the production and distribution planning brings certain challenges. These challenges can be divided into general challenges and industry specific challenges.

3.4.1 General challenges

Traditional scheduling tools assume a deterministic and static environment, while in reality this is never the case. In static settings, all jobs and corresponding processing times are known before scheduling. On the other hand, in dynamic settings jobs enter the system over time and planners do not have full knowledge about upcoming jobs (K. Lee et al., 2019). Since not all information is known in advance, re-planning may be required when new jobs enter the system. In the case of premix producers, not only new jobs arrive but unforeseen events occur as well such as machine breakdown, lack of raw materials and truck delay. In previous attempts to automate both production and distribution schedules, static scheduling methods were used. As these methods assumed all information was known before scheduling, the schedules were not able to deal with the dynamic nature of the environment. During the day, various unforeseen events take place such as changing orders, this includes change in product type or product quantity. Other unforeseen events are delay of trucks, equipment downtime and lack of raw materials, which makes the planning environment dynamic (Jackson, 1957). Planners at premix manufacturers constantly reschedule the production and distribution planning. They do this based on expert knowledge gained by experience. There are no clear guidelines for adapting the planning when an unforeseen event occurs as every situation needs a different approach.

3.4.2 Industry specific challenges

Contamination of raw materials makes the production planning challenging as this constrains production sequences. For instance, the lowest concentration of copper in sheep feed kills them,

while copper is an essential nutrition for poultry. Therefore, sheep feed cannot be produced immediately after the production of poultry feed. Either the production line is cleaned by producing a so called 'flush batch', which is cheap material that is discarded after it ran through the production line. Another option is to produce sufficient non-contaminating batches in between the contaminating products. This is referred to as sequence-dependent setup times in the literature (Toso et al., 2009). Furthermore, when a machine or silo is emptied, a small amount of material remains behind, for instance because it sticks to the walls. Compound feed producers know this ratio for each material type per machine type or silo. With this information it can be calculated how many production batches should be between two product types where contamination is dangerous. For example, when 0.1% of copper remains in a mixing machine every time it operates and sheep are allowed to have 0.001% of copper in their food, then at least two other batches should be produced in between the production of sheep and poultry feed.

Another industry specific challenge is **limited storage space** at production plants. Most producers of premix do not have enough storage spaces and silos to store each type of product and these spaces differ in size. Each silo can store one product type at the time to avoid contamination. No matter what amount is stored in a silo, it is considered full until the silo is emptied. Obviously, production orders with the exact same production type can be stored in the same silo. Therefore, planners must smartly assign finished products to silos. Currently, the storage capacity is not used to their full extent. Often production orders are assigned to silos with ample capacity, blocking the whole silo and thus using far more storage space than necessary. The same logic can be applied to storage spaces for bagged products. Storage spaces differ in capacity and one order is stored in a storage space.

Furthermore, **time constraints and delivery expectations** in the premix industry makes combined production and distribution scheduling more challenging. High competition between premix producers enables compound feed producers and farmers to demand same day delivery for emergency orders. They will simply go to a competitor if a premix producer cannot meet its demands. Consequently, new orders arrive during the day which requires re-planning of the production schedule as well. High competition has made production and distribution schedules tight already, leaving little room for re-planning.

3.5 Scope

With an introduction to the premix industry and the description of the current production and distribution planning, the scope of this thesis can be defined. The objective of the study is to show that an integrated and automated schedule for both production and distribution processes at premix producers is possible, while accounting for unforeseen events. To make the setting of the study as realistic as possible within the given timeframe, the following challenges are included in the project. First, a dynamic setting is assumed. This includes the arrival of new orders over time and unforeseen events. Unforeseen events consists of machine breakdown and truck delay. Furthermore, the industry specific challenge of contamination is accounted for by means of sequence-dependent setup times. In addition, finite storage space is assumed. While incorporating the aforementioned challenges, the following will be considered out of scope:

1. **Routing of trucks**. However, it will be assumed that trucks have a departure time and that orders need to be allocated to trucks.

2. **Intake of raw materials**. The assumption is made that raw material is always available.

3. **Material dosing**. It will be assumed that jobs do not require dosing of raw materials, i.e. they are immediately available for mixing once scheduled.

4. **Multiple product request in one order**. It will be assumed that orders contain one product type.
5. **Transportation time within the factory**. Transporting jobs between stages in the production process are neglected since this does not influence the scheduling process.
6. **Perishability**. The assumption is made that raw material and finished products do not spoil over time.

3.6 Problem formulation

This section includes a mathematical model of the integrated production and distribution scheduling problem that will be solved in this thesis. The model helps to precisely define the problem and is therefore not meant to be exactly solvable. The integrated production and distribution planning requires simultaneous decisions for production and distribution. For the production part it should be decided when an order is produced and on which mixing machine and sacking machine. Furthermore, it should be decided which storage space is used to store a produced product. For the distribution part, order to truck allocation is considered. These decisions will be modeled as decision variables. The mathematical model is based on the works of (Hou et al., 2022), (Solina & Mirabelli, 2021) & (Z. Li et al., 2021), who studied similar problems.

Parameter	Description
$ontime_j$	Binary variable indicating if job j is one time
d_j	Due date of job j
s_{jm}	Start time of job j on machine m
ct_{jm}	Completion time of job j on machine m
$st_{jj'm}$ where $j \neq j'$	Setup time between job j and j' on machine m
q_j	Ordered quantity of job j
cap_s	Capacity of storage space s
cap_{tr}	Truck capacity of truck tr
dep_{trj}	Departure time of truck tr and assigned job j
y_{jmt}	Job j is assigned to machine m at time t
$BigM$	Big number
Decision variable	**Description**
X_{jmixt}	Assign job j to mixing machine mix at time t
X_{jsackt}	Assign job j to sacking machine $sack$ at time t
Z_{jst}	Assign job j to storage space s at time t
V_{jtrt}	Assign job j to truck tr at time t

Table 1: Parameters & decision variables

Objective function

$$\max \quad \sum_{j \in J} ontime_j$$

s.t.

$$\sum_{j \in J} y_{jmt} \leq 1 \quad \forall m \in M \quad \forall t \in T \tag{7}$$

$$\sum_{mix \in Mix} X_{jmixt} = 1 \quad \forall j \in J \quad \forall t \in T \tag{8}$$

$$\sum_{sack \in Sack} X_{jsackt} = 1 \quad \forall j \in J \quad \forall t \in T \tag{9}$$

$$\sum_{j \in J} Z_{jst} \cdot q_j \leq cap_s \quad \forall s \in S \quad \forall t \in T \tag{10}$$

$$\sum_{j \in J} V_{jtrt} \cdot q_j \leq cap_{tr} \quad \forall tr \in Trucks \quad \forall t \in T \tag{11}$$

$$st_{jj'm} \leq ct_{j'm} - s_{jm} \quad \forall j, j' \in J, j \neq j' \quad \forall m \in M \tag{12}$$

$$BigM \cdot ontime_j \geq max\left(d_j - dep_{trj}, 0\right) \quad \forall tr \in Trucks \quad \forall j \in J \tag{13}$$

A textual explanation of the model is given here. Customer orders contain information about the order due date, ordered product type and quantity. Each order is translated to a job. Thus, jobs contain information about the due date, product type and quantity as well. Since the objective of premix manufacturers is to deliver as much orders as possible on time, the objective function is to maximize the number of on time delivered jobs. The 7th constraint ensures that at most one job can be processed on a machine at the time. Constraint 8 and 9 ensure that all jobs must be assigned to a mixing machine and a sacking machine at a given time t. Note that this notation assumes discrete time. In reality, this is not the case due to the varying processing times of jobs. Besides, for bulk products the sacking machine is a dummy machine. Furthermore, the capacity of the storage space and a truck cannot be exceeded when a job or order is allocated (Constraint 10 & 11). Constraint 12 is about sequence-dependent setup times and ensures that the difference between completion time of job j' and start time of job j must exceed the setup time between job j and j'. Finally, job j is considered on time when the departure time of its assigned truck tr is lesser than its due date (Constraint 13).

4 Solution method

This section explains the solution method that was implemented. To start, the Semi-Markov Decision Process framework is discussed. Followed by a description of reward shaping. After that, the simulation model is explained together with the simulation setup. Furthermore, the DRL algorithm, offline reinforcement learning and the benchmark algorithm are discussed.

4.1 Semi-Markov Decision Process

In order to use DRL for the given problem in section 3, the problem must be modelled as a Markov Decision Process. As described in section 2, MDP is a discrete-time stochastic framework which is used by DRL models to solve an optimization problem. In the work of Rummukainen and Nurminen (2019), a Semi-Markov Decision Process (SMDP) was used as framework for a DRL algorithm. In a SMDP, the state transitions to the next state once the state of the environment changes, i.e. when an event happens and an action from the agent is needed. As a consequence, time is continuous in SMDP's. For comparison, in a MDP the state transitions with a fixed time interval regardless of events. This means that in a MDP the state could transition when no action is required by the agent. Consequently, the agent experiences many transitions where it learns nothing, which leads to a longer training phase.

State transition triggered by events, require less state changes when the environment is modelled as a SMDP and is especially useful when a discrete-event simulation environment is used to train an agent. For the reason that simulated time requires little computational time. On the other hand, state transitions require time consuming observation space updates. When less state transitions are required, the total computaional time descreases which is the case with SMDP's. A downside of SMDP modelling is that most DRL algorithms are built for MDP frameworks, which could affect the DRL agent's performance. Rummukainen and Nurminen (2019) adapted the PPO algorithm (Schulman et al., 2017) for the SMDP setting by adding continuous time discounting in the advantage estimator. Furthermore, they excluded the entropy term while tracking the 100-rollout interval average reward rate during training. Since these adaptions of the original PPO algorithm are relatively small, it is likely that the original PPO algorithm produces good results in combination with a SMDP framework. In addition, the advantage of less state changes and less training time in SMDP's outweighs this downside. Therefore, a SMDP framework was used in this project.

The SMDP formulation requires a state space S, action space A and a reward function R. As explained in section 2, the transition function T cannot be determined because the transition probabilities are unknown. Instead, a DRL agent is trained to estimate the policy. Each component is elaborated on below.

4.1.1 State space

The state space should include all relevant information for the agent to decide the next action. Although it should be limited to information that is beneficial for learning, since complex state spaces obstruct the agent in learning the optimal policy (Fu et al., 2017). Furthermore, a larger state space requires more computational time and could enlarge the action space. In order to find the best state space, various settings have been investigated. The settings that were not included in the final state space are discussed at the end of section 7. Here, only the final state space is discussed. An overview of position of each state space component is given in Figure 4. To start, the state space was divided into three components:

1. order information
2. resource information
3. additional information

Order information
Order information itself was divided into jobs waiting in the queue and orders ready for shipment. Information about an order included the product type, quantity, packing type, status, location and due date. When an order arrived into the system, it was converted to a job in the queue waiting to be produced. The state space stored the following information about the jobs in the queue:

1. **Number of jobs per product type**. For each product type, there was a variable which tracked the number of jobs in the queue.

The jobs were grouped by product type, because that was the most important selection criteria besides the due date. After a job was produced and stored in the warehouse (a job in bulk was stored in a silo), it was added to the state space again in a new array. Now, all information about the order was explicitly shown, whereas jobs were grouped per product type. The agent allocated the finished job to a truck and once this was done the job was removed from the state space. The top two finished jobs were shown in the state space. For the reason that in each state transition at most four jobs were finished as there were four sacking lines. Since the probability that all (or three out of four) sacking lines were ready at the exact same time was small, only two jobs were shown to limit the state space. If there were less than two orders ready to be allocated to trucks, dummy jobs were included to have a consistent state space.

Resource information
The resource information was divided into information about the mixing lines, sacking lines, warehouse storage and trucks. For the mixing and sacking lines the following information was included:

2. **occupied**. This is a binary variable which indicated if the mixing line was occupied. Once a job was assigned to the mixing line, it was set to one until the job was finished and assigned to a sacking line. Then, the variable was set to zero.
3. **contamination history**. This is an array with binary variables. A one indicated that the product type could be produced without setup time, while a zero meant that setup time was required.

The contamination history was included for the mixing lines and sacking lines to account for the contamination of subsequent jobs. Which product types contaminate with each other is known to premix manufacturers. Thus, with the knowledge of the next job's product type (as explained above) and the contamination information, it could be determined if consecutive jobs contaminated and therefore required setup time. If jobs contaminated, an additional setup time was included to the next job's production time.

Sacking lines included another variable; the **remaining processing time** of that particular machine. This turned out to be beneficial for learning an optimal policy during training. The warehouse storage was included in the state space as well. For each storage space, the following information was incorporated:

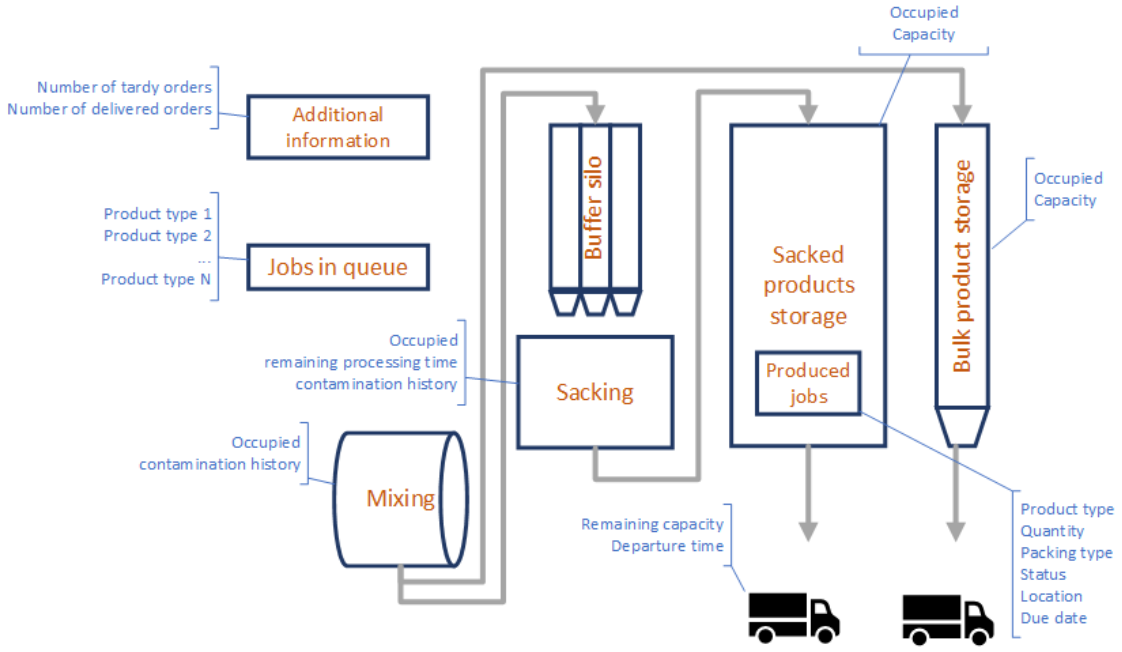

Figure 4: Overview of the observation space in the production & distribution process

4. **occupied**. For the storage spaces the binary occupation variable was included to track its availability.

5. **capacity**. The capacity of the storage space was included because some jobs did not fit in each storage location. Therefore, the agent had to smartly assign jobs to storage spaces.

Last, information about the trucks was included in the state space. Per truck the following information was included:

6. **remaining capacity**. The remaining capacity indicated how much space was left in the truck.

7. **departure time**. The departure time showed when the truck left the system.

Unlike the storage spaces' capacity, the trucks' capacity was decreased each time a job was assigned. This was done for the fact that multiple jobs could be assigned to a truck, while this was not the case for storage locations. The departure time was predetermined since scheduling the trucks was out of scope for this thesis. A job was considered on-time when a job's due date was on or after the departure time of its assigned truck.

Additional information
As additional information, the number of delivered orders and the number of tardy orders were added. It was expected that including these variables explicitly in the state space enhanced the learning process, since this information embodied the objectives of the model.

4.1.2 Action space

In the scheduling process, multiple actions are made. Therefore, the action space was modelled as a multi-discrete action space with four actions. This limited the algorithm choice since not all DRL algorithms are able to handle multi-discrete action spaces. Converting the multi-discrete action space to a single discrete action space was not possible since the action space exploded with all combinations of four different actions. The location of the action in the production and distribution process is given in Figure 5 with the numbers. The following actions were included:

1. Job in the queue to mixing line allocation
2. Job from mixing line to sacking line allocation
3. Job from sacking line to storage allocation
4. Job in storage to truck allocation

The first action was about choosing which job in the queue to produce next and on which mixing line. Since delivering orders on time was the most important objective of this problem, it was logical to schedule jobs based on their due date. Due to contamination of product types, it could be better to select an other job that had the same product type as its predecessor. Finding a balance between these scheduling rules had to be learned by the agent to optimize its performance. To guide the agent towards such a policy, the first action was about deciding which product type to produce next. Based on this action, the job with that product type and earliest due date was produced next on the idle mixing line.

All actions were one hot encoded, which means that each product type - mixing line pair was a separate discrete action. In other words, A_{ij} where i was the product type and j was the mixing line and resulted in product type - mixing line pairs $A_{11}, A_{12}, A_{21}, \ldots$. Therefore, the total number of actions equaled the number of product types times the number of mixing lines. Moreover, a 'waiting' action was included which had to be chosen when no other option was available. The second action allocated a finished job on a mixing line to a sacking line. Bulk products do not need sacking, thus were directly allocated to a storage space. The third action decided which storage place was assigned to sacked jobs. Last, the fourth action allocated stored jobs to a truck.

Action masking
As mentioned above, action masking was applied to guide the agent towards the optimal policy. Action masking was chosen because it has been proven to decrease the training time, simply because the agent does not have to learn what valid and invalid actions are in a particular state (S. Huang & Ontañón, 2020). Masking invalid actions required extra computational time, although the extra computational time did not outweigh the benefits. The choice for SMDP modelling led to many invalid actions since it was unlikely that all four actions required a decision from the agent in a state transition. Thus, in many transitions the agent had to select the wait option. Actions were masked based on the state of the system. For each action the following situations led to the masking of an action:

1. For the first action busy mixing machines and product types without jobs were masked.
2. For the second action busy or idle mixing machines, occupied sacking machines and non matching sacking types and sacking machines were masked.
3. For the third action busy or idle sacking lines, occupied storage locations and storage locations with insufficient capacity were masked.
4. For the fourth action trucks with insufficient capacity were masked.

4.1.3 Reward function

The reward function is arguably the most important part of the environment design in reinforcement learning problems, since the agent defines its policy based on maximizing the reward function. Therefore, the reward function had to reflect the objective of the actual process. In order to find the best possible results, multiple reward functions were tested. Negative reward functions that were analyzed included a penalty for tardy orders, contamination cleaning, makespan and throughput. Contradictory, positive reward function were tested as well and included a reward for on time orders, no contamination cleaning, a reward for each valid action and a reward at the end of an episode for the ratio tardy against delivered orders. All components were analyzed in different compositions and with different weights in a small experiment. Parameter values were chosen such that there was under capacity of the resources. Then, for each reward function a test run was made with 30 episodes. Afterwards, the reward functions, delivered orders, tardy orders, makespan and setup times required were compared to find the best combination of reward components.

Shaping the reward function such that the agent is guided towards the intended behavior can be difficult and negatively influence the performance of the agent (Armstrong et al., 2020). Besides, a complex reward function may prevent the agent from learning the actual dynamics of the environment. Therefore, reward functions should be as basic as possible without compromising on functionality (Koenig & Simmons, 1996). Moreover, a simple reward function improves the generalizability of the DRL agent. It was found that the reward function with a large reward for on time delivered orders and a small penalty for the makespan led to the steepest learning curve, although there was not much difference between the reward functions. Therefore, the decision was made to use the simple reward function as follows:

$$reward = \begin{cases} 1 & \text{for each on time delivered order} \\ -0.0001 * makespan & \text{for each stored job} \\ 0 & \text{otherwise} \end{cases} \tag{14}$$

The reward was calculated at the end of each state and given to the agent as feedback for its behavior. The positive reward ensured that the agent learned to produce and deliver the orders on time, while maximizing the throughput of the system by minimizing the makespan. The makespan was calculated as the difference between the time the job was stored in the warehouse and the arrival time. Where the reward component was awarded is shown in Figure 5.

4.2 Reward shaping

From the literature review came forward that the reward function is one of the most important design choices in the DRL framework (Armstrong et al., 2020). As discussed in the previous section, the reward function was carefully selected based on expert knowledge, the literature review and experiments. It turned out that a multi-component reward function led to the best results for the integrated production and distribution scheduling problem. Unfortunately, this brought a new challenge, namely balancing the weights of the two components. During the reward function selection the order of magnitude for the two components was assessed, yet the best weights could not be determined. Although this was not surprising, as determining the weights is a time consuming task. Since the reward function has such a major influence of the performance of the DRL agent, it pays off to find the best ratio between the reward function components (Ng et al., 1999).

One of the most promising frameworks to solve a multi-component optimization problem is

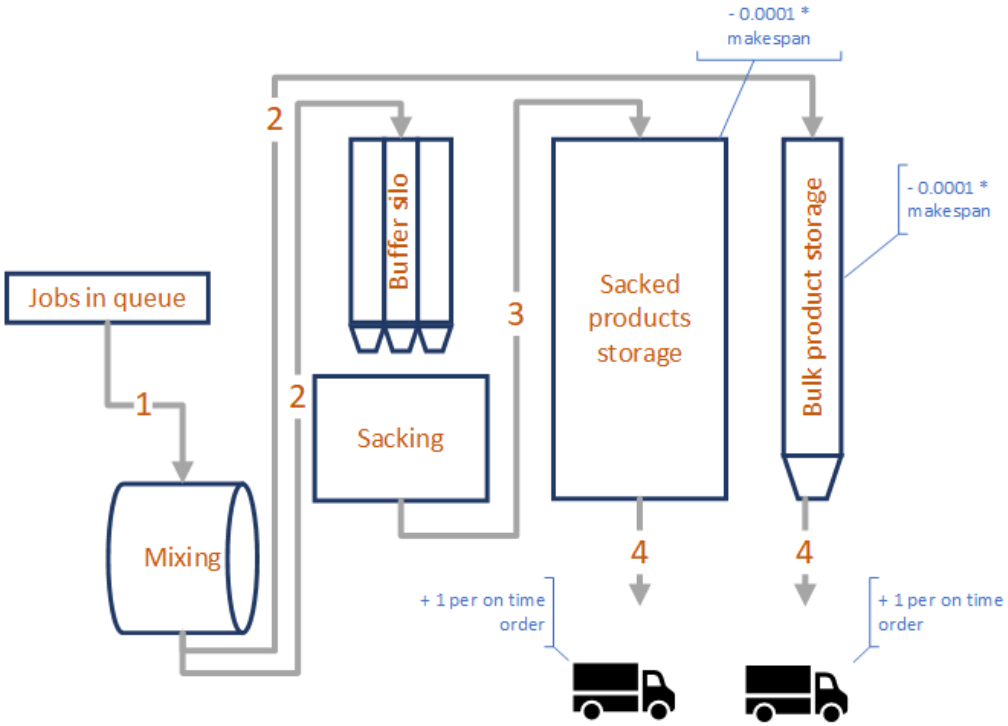

Figure 5: Overview of the action space and reward function in the production & distribution process

Bayesian optimization (Mockus et al., 1978). This technique uses a surrogate function for the objective function $p(y|x)$ to track the previous evaluation results. The surrogate function is a probability model of the actual objective function and is easier to optimize. Each iteration, the surrogate function is used to find the best parameter values. Then, these values are applied to the actual objective function and these results are subsequently used to update the surrogate function. By continuously updating the surrogate function, the optimal parameter settings of this function will eventually approach the optimal parameter settings of the actual objective function. This is where Bayesian optimization fundamentally differs from random search or grid search techniques: the next parameter values depend on past results. Although more time must be spent on selecting the next values, this will lead to better results in fewer iterations (Bergstra et al., 2013).

Within Bayesian optimization, there are several sub-techniques that are based on this framework. One of them is sequential model-based optimization (SMBO). There are numerous variants of the SMBO technique. They differ in the method of building the surrogate function and the criteria of next parameter value selection. Often, the surrogate function is modelled by Gaussian processes, random forest regressions or Tree Parzen Estimators (TPE). The decision was made to use TPE because it yielded better results than non Bayesian optimization techniques and was able to find the optimal settings faster than GP methods (Bergstra et al., 2011). TPE models applies Bayes Rule to represent $p(y|x)$. Furthermore, $p(x|y)$, the probability of the parameter values given the outcome on the actual objective function, is represented by:

$$p(x|y) = \begin{cases} l(x) & \text{if } y < y^* \\ g(x) & \text{if } y \geq y^* \end{cases} \tag{15}$$

In this equation, y^* is the threshold value of the objective function, x is the chosen set of parameter values, y is the outcome of the actual objective function and $p(y|x)$ is the surrogate function. As can be observed in the equation, two functions are made for the parameter values: one where the outcome of the objective function is below the threshold ($l(x)$) and one where the outcome of the objective value is above the threshold ($g(x)$). For each parameter, two probability functions are made. One with the values that lead to an outcome below the threshold and one with the values that yield outcomes above the threshold. Furthermore, the expected improvement method is mostly used to select the next parameter values.

$$EI_{y^*}(x) = \int_{-\infty}^{y^*} (y^* - y)p(y|x)dy \tag{16}$$

TPE takes sample parameter values from $l(x)$ and $g(x)$, evaluates them on the expected improvement method and returns the values that leads to the best outcome. As stated before, these values are finally evaluated on the actual objective function. The Bayesian optimization with TPE sampling is implemented with Optuna (Akiba et al., 2019). This is an easy to implement parameter optimization library, which has been proven to efficiently find optimal parameter settings (Oono & Suzuki, 2019) & (Saito et al., 2020).

4.3 Simulation model

Online RL agents learn by interacting with a real life environment or with a simulated environment. Although learning via a real life environment is preferred, this is rarely the case. By interacting with the real life environment, the agent can explore all facets of the environment. However, as explained in section 2, this can be costly due to the sub-optimal actions the agent chooses in the early stage of training. Therefore, a simulation model is regularly used to train the RL agent, after which it can be exploited on real life situations. The downside of simulated environments is that it is difficult to capture all dynamics of the real life environment in the model. Besides, flaws in the simulation model can be used by the agent to its advantage. Nevertheless, the downsides of a simulation model do not outweigh the downsides of training the RL agent on the real life environment. Thus, a simulation model was made to mimic the environment of a premix production and distribution process.

4.3.1 Discrete-event simulation

The environment was modelled as a continuous-time discrete-event simulation (DES), which models the system as a discrete sequence of events over continuous-time. Each event occurs at some moment in time and changes the state of the system (Robinson, 2008). Time was continuous due to the use of a SMDP framework (R. S. Sutton et al., 1999), where the state changed based on events. Therefore, the simulation could immediately advance to the next event. The choice for DES was made because it is frequently used in scheduling literature (Kardos et al., 2021), (Waschneck et al., 2018) & (Rummukainen & Nurminen, 2019). Besides, it is an efficient way to mimic complex, dynamic and stochastic production environments without the need of an analytical model (Hedtstück, 2013).

4.3.2 Events

There were seven events in the simulation environment that should be discussed. These events described the journey of an order through the system. Each event is elaborated on below in

chronological order:

1. Order arrival. Orders were released into the system based on a arrival rate. Once arrived, the order was transformed to a job in the queue, waiting for production. Orders were production requests from customers and contained a single product. Included information about the order were the product type, quantity, packing type and due date.
2. Job to mixing machine allocation. When a mixing machine was idle, a job could be allocated to the machine. Due to capacity restrictions of the mixing machines, jobs had to be produced in multiple batches in most cases. Furthermore, transportation time between machines was neglected in the whole production process.
3. Job operation on mixing machine. Once allocated, the job was produced on the mixing machine. Due to batch production, the total mixing time depended on the requested quantity and the mixing machine's capacity. The mixing time was automatically determined in the simulation. It was assumed that the whole job was produced in one go, i.e. in a series of batches before a new job was produced on that particular machine.
4. Job to sacking line allocation. When the mixing machine was done and a sacking machine was available, the job had to be allocated to a sacking line. Bulk production requests skipped this step and went directly to the storage location for bulk products.
5. Job operation on sacking machine. After the job was allocated to a suitable sacking machine, it was packed in bags. Each small bag sacking machine had two buffer silos, which means that if another job was being processed on the sacking machine, the job had to wait until the other job was ready. Bulk products skipped this step as well and were immediately stored upon shipment without sacking.
6. Job to storage allocation. Once the job was bagged, it had to be allocated to a storage location. The job was then temporarily stored in the warehouse until it was loaded into a truck.
7. Job to truck allocation. Finally, jobs were allocated to a truck. Once allocated, the job was immediately transferred to the corresponding truck. Consequently, the storage location became available for other jobs.
8. Truck departure. When the departure time of a truck was reached, the truck left the system. For each order in the truck it was determined if it was delivered on time or late.

4.3.3 Entities

Next to the events, there were several entities that contained information about the orders and resources. These entities were required in the simulation model to keep track of the system's state. The following entities were used in the simulation model:

1. Orders. When an order arrived, it was tracked until it left the system. For the agent, only finished orders in the warehouse were visible.
2. Queue. In the queue, all jobs were listed that are waiting to be scheduled. As mentioned before, arriving orders were placed in the queue. Jobs in the queue had a product type, quantity, packing type and due date.
3. Mixing & sacking machines. Since there were multiple mixing and sacking machines operating in parallel in each production stage, the status of each machine was documented separately.

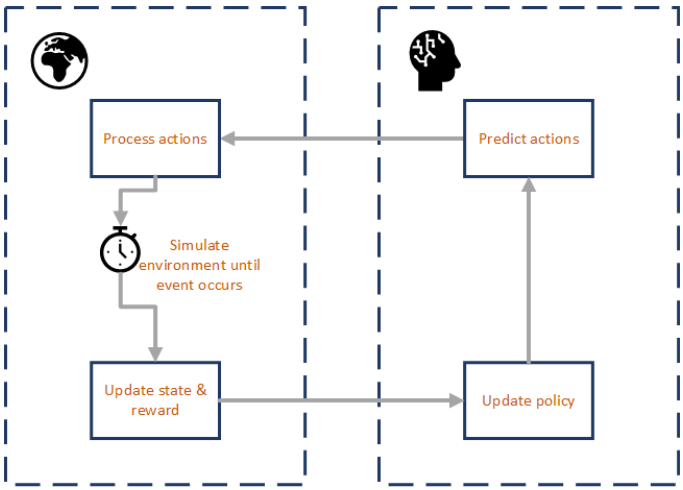

Figure 6: DRL agent - environment interaction

4. Warehouse storage. For the storage locations, the capacity and availability was kept track of.

5. Trucks. For the trucks, the remaining capacity and departure time was recorded.

Besides these entities, there were multiple arrays that were used in the simulation model to track the status of the system. This included arrays that track jobs that were being processing on all machines and the status and location of an order. Moreover, there was an entity that tracked the idleness of machines. This was necessary because the agent made decisions based on the availability of resources, not on specific orders. For example, the agent observed that a sacking machine was finished and assigned it to a storage space, without observing which job was actually on the machine. However, to simulate the environment, the simulation model had to know which job was transferred between the sacking machine and storage space. Without that information it remained unknown when a job was finished and thus when an order could be shipped to the customer. Although this information was required for the simulation model, this was useless for the agent. Therefore, these arrays were not included in the observable state space to prevent information overload for the agent.

4.3.4 Interaction of DRL agent & simulation model

Now that the environment was modelled as a SMDP and DES model, a DRL could interact with it to make the required decisions. An overview of the interaction is shown in Figure 6. In terms of RL theory: given state s, the agent performed action a, which was received by the model and simulated until state s changes to state s'. Therefore, the following steps were repeated until the episode terminates:

- Choose action
- Simulate environment until event occurs
- Return reward and proceed to next state

As mentioned before, the environment was modelled as a SMDP. This means that the state changed

once an event occurred that required an action from the agent. What these events were depended on the state itself. Events that led to state change are:

- Order arrival. When mixing machines were idle and no orders were waiting in the queue, the state changed as soon as an order arrived.
- Resource capacity change. Once a machine was done with processing a job, the state changed in order to assign the finished job to the next phase of production. Furthermore, the state changed once a job was stored in the warehouse to enable the agent to allocate the job to a departing truck.

Once the state has changed, the agent should perform an action. Invalid actions were masked as explained in section 4.1.2. Although invalid actions were masked, it could still happen that the chosen action prevented the state from changing due to the wait action. In that case, the state advanced to the next state until the agent chose an action that led to a state change. The episode terminated when it reached timestep 2.880, which is three working days. Upon termination, the simulation model returned several performance metrics, which are discussed in section 7.

4.4 Simulation setup

The agent - simulation model interaction was modelled with the OpenAI Gym framework since it is widely used in reinforcement learning studies and most algorithms rely on it. The OpenAI Gym framework requires the following functions:

- Observation space. The observation space is the part of the environment that can be observed by the agent. This could be the whole environment or a part of it.
- Reset. The reset function initializes the environment when an episodes starts.
- Step. The step function translates the agent's decision into actual actions.

The step function executes the actions and returns the updated observation space to the agent. Furthermore, the reward is calculated at the end of the step function and shown to the agent as well. In addition, the episode termination flag is returned. The OpenAI Gym framework provides an option to include additional information but this is not used in the thesis.

4.4.1 Initialization

First, the simulation model was initialized. This can be divided into two parts. The initialization of the whole simulation, which was done only at the start of an entire run and the initialization of an episode, which was done with the reset function. To start, initializing the whole simulation included the definition of parameters about the environment. For this case, the number of mixing lines, sacking lines, storage spaces and trucks were defined, along with their capacity. Furthermore, the processing time of each machine was defined, as well as the contamination clean time. Besides, the number of orders and jobs in the observation space were defined.

The reset function initialized episode dependent variables. Here, the simulated time was reset, as well as the observation space and all KPI's. Furthermore, a starting point of the environment was randomly generated to mimic a real life setting. Starting without orders in the system would give an unrealistic view to the agent. An overview of the environment's initialization is provided

in Algorithm 1. The number of initial orders in the system could be chosen by the user, as well as the number of initial jobs on the mixing lines, sacking lines, storage and queue. To keep the initialization as simple as possible, five initial orders were generated: two on the mixing lines, two on the sacking lines and one in the queue. Each variable was chosen based on a probability distribution. The probability distributions were based on data from a customer of KSE where possible. This is further elaborated in section 5. After an order was generated, the array that tracks the state information was updated accordingly.

Algorithm 1 Initialize

```
for number of initial orders do
    product type ← probability distribution
    packing type ← probability distribution
    quantity ← probability distribution
    due date ← probability distribution
    for first two initial orders do
        location ← random sacking line
        update sacking line
    end for
    for third and fourth initial orders do
        location ← random mixing line
        update mixing line
    end for
    for remaining orders do
        location ← queue
        update jobs in queue
    end for
end for
update total orders
```

4.4.2 Step function

After the episode was initialized, the step function executed the actions in the simulation environment (Algorithm 2). The agent made a decision for each of the four sub-actions. The sub-actions were executed when the agent did not decide to wait for that particular sub-action. Each action is explained below. Once all actions were carried out, the step function entered a loop were production was executed until an event occurred that led to a state change. These events were explained in section 4.3.2. Each time the loop was executed, one minute of simulated time passed by. Then, the reward was calculated and the state was updated and presented to the agent.

4.4.3 Action execution

The action execution function was triggered by the step function. The first action assigned jobs from the queue to mixing lines. The pseudocode is shown in Algorithm 3. First, it was checked if the mixing line was available and if the chosen product type had jobs in the queue. These checks were not necessary if action masking was used. Then, it was checked which mixing line was selected. If the second mixing line was selected, bulk jobs could not be produced. After that, the contamination criteria were checked, which is further elaborated on in section 6.1. If the successive jobs contaminated each other, an additional cleaning time was added to the production time and the contamination history was reset. After that, the job was removed from the queue and the mixing line was set to occupied.

Algorithm 2 Step

```
action ← agent decision
if first action is not wait then
    execute first action
end if
if second action is not wait then
    execute second action
end if
if third action is not wait then
    execute third action
end if
if fourth action is not wait then
    execute fourth action
end if
while (all mixing lines are busy or no jobs in queue or successive sacking lines are
    occupied) and (current producing sacking lines are not ready or no jobs in queue
    or storage is full) and (current busy storage spaces are not ready or trucks are full) do
    process jobs on mixing lines
    process jobs on sacking lines
    process jobs to storage spaces
    if departure time of truck equals current time then
        depart truck
    end if
    if inter arrival time plus arrival time previous order equals current time then
        generate order
    end if
end while
calculate reward
update state
```

Algorithm 3 First action

```
if mixing line is idle then
    if allocated product type has jobs in queue then
        if mixing line id is 1 then
            if contamination criteria are met then
                assign job with chosen product type and earliest due date to
                mixing line without cleaning time
            else
                assign job with chosen product type and earliest due date to
                mixing line with cleaning time
            end if
        else
            if contamination criteria are met then
                assign job with chosen product type, no bulk packing and
                earliest due date to mixing line without cleaning time
            else
                assign job with chosen product type, no bulk packing and
                earliest due date to mixing line with cleaning time
            end if
        end if
        remove job from queue and set mixing line to occupied
    end if
end if
```

The second action assigned jobs from the mixing line to the sacking line (Algorithm 4). Again, the initial checks were not required when action masking was used. First, it was checked if the mixing line was done and if the sacking line was unoccupied. Note that there was a difference between occupied and ready: a job could be ready with producing on a particular machine, yet still occupying the machine when it was not assigned to its subsequent machine. For sacking lines, the contamination criteria were checked as well, and production time was adapted accordingly. The contamination history of the mixing line was updated once the job was transferred to the sacking line. Last, the mixing line was set to unoccupied while the sacking line was set to occupied.

Algorithm 4 Second action

```
if mixing line is ready and sacking line is unoccupied then
    if contamination criteria are met then
        assign job to mixing line without cleaning time
        update contamination history mixing line
    else
        assign job to mixing line with cleaning time
        reset contamination history
    end if
    set mixing line to unoccupied and set sacking line to occupied
end if
```

The third action assigned finished jobs on sacking lines to storage spaces and its pseudocode is shown in Algorithm 5. It was checked if the job was ready and if the storage space had sufficient capacity. The job was assigned to the storage space and the contamination history of the sacking line was updated. Furthermore, the sacking line was set to unoccupied while the storage space was set to occupied.

Algorithm 5 Third action

```
if sacking line is ready then
    if storage space capacity is sufficient then
        assign job to storage space
        update contamination history sacking line
        set sacking line to unoccupied and storage space to occupied
    end if
end if
```

The fourth action was the final step in the production process and assigned orders to trucks. The pseudocode is shown in Algorithm 6. The action was triggered once orders were in a storage space. First, the algorithm checked if there was sufficient capacity remaining in the truck. Then, the order was assigned to a truck and the storage space was set to unoccupied. Furthermore, the truck's capacity was updated.

Algorithm 6 Fourth action

```
if order is in storage then
    if truck's remaining capacity is sufficient then
        assign order to truck
        set storage space to unoccupied and update truck's capacity
    end if
end if
```

4.4.4 Production process

The production process was simulated until the state changes. The process was simulated by subtracting one minute of simulated time from the remaining processing time each time the processing function was called. This can be seen in the pseudocode for the mixing lines (Algorithm 7), sacking lines (Algorithm 8) & storage spaces (Algorithm 9). Logically, this was only done for machines that were actually producing, which was checked at the second line of the pseudocode (Algorithm 7 & 8).

Algorithm 7 Process jobs on mixing lines

```
1: for each mixing line do
2:     if mixing line producing then
3:         subtract processing time with one
4:     end if
5: end for
```

Algorithm 8 Process jobs on sacking lines

```
1: for each sacking line do
2:     if sacking line producing then
3:         subtract processing time with one
4:     end if
5: end for
```

Algorithm 9 Process jobs to storage space

```
1: for each storage space do
2:     if storage space is busy then
3:         subtract processing time with one
4:     end if
5: end for
6: if storage space is ready then
7:     update order information
8: end if
```

4.4.5 Order generation

When the inter arrival time plus the arrival time of the previous order equaled the current time, a new order was generated (pseudocode in Algorithm 10). All variables were chosen based on a probability distribution, as was explained in section 6.1. The chosen quantity depended on the packing type since it was found that the ordered quantity differs for each packing type. Once an order was generated, it was added to the queue and total orders in the system. Furthermore, a new inter arrival time was generated. The distribution of the inter arrival time will be explained in section 5.2. If the inter arrival time was greater than zero, the while loop ended.

4.4.6 Truck departure

When the departure time of a truck equaled the current simulated time, the truck departed. The orders that were assigned to the truck left the system. If their due date equaled or was after the current time, the order was considered delivered on time, while orders with a due date after the departure time were considered tardy. Based on this information, the KPI's were calculated.

Algorithm 10 Order generation

```
while generate order is True do
    product type ← probability distribution
    packing type ← probability distribution
    if packing type is bulk then
        quantity ← probability distribution
    end if
    if packing type is small bag then
        quantity ← probability distribution
    end if
    if packing type is big bag then
        quantity ← probability distribution
    end if
    due date ← probability distribution
    add order to total orders and include job in queue
    generate new inter arrival time
    if current time + new arrival time is not current time then
        generate order ← False
    end if
end while
```

The KPI's are discussed in dept in section 6.5. Then, the orders were removed from the system and the truck's capacity was set to the original capacity. Furthermore, a new departure time was generated for the truck. The pseudocode of the truck departure is shown in Algorithm 11.

Algorithm 11 Truck departure

```
for each truck do
    if departure time equals current time then
        calculate tardiness for each order in truck
        update KPI's
        remove all orders in that truck from system
        restore truck's capacity and generate new departure time
    end if
end for
```

4.4.7 Action masking

At the end of each step, invalid actions for the next iteration were masked by the action masking function. The pseudocode is shown in Algorithm 12. As was explained in section 4.1.2, all invalid actions were masked through hardcoding. For the first action, busy mixing lines and product types without jobs in the queue were masked. In addition, bulk jobs were masked for the second mixing line. For the second action, options were masked when the mixing line was either not ready or idle, if the sacking line was not suitable for the packing type of the mixing line's job. The options of the third action were masked when the sacking line was not ready or idle and if the storage space was occupied or has insufficient capacity. For the fourth action, dummy jobs were masked, as well as trucks with insufficient remaining capacity.

Algorithm 12 Action masking

```
for first action do
    if mixing line is busy then
        mask mixing line
    end if
    if job in queue is dummy then
        mask job
    end if
    if job is bulk product then
        mask second mixing line
    end if
    if real action is available then
        mask wait action
    end if
end for
for second action do
    if mixing line not ready or idle then
        mask mixing line
    end if
    if sacking line not suitable for packing type or occupied then
        mask sacking line
    end if
    if real action is available then
        mask wait action
    end if
end for
for third action do
    if if sacking line not ready or idle then
        mask sacking line
    end if
    if Storage space is occupied or has insufficient capacity then
        mask storage space
    end if
    if real action is available then
        mask wait action
    end if
end for
for fourth action do
    if order is dummy then
        mask order
    end if
    if truck has insufficient capacity then
        mask truck
    end if
    if real action is available then
        mask wait action
    end if
end for
```

4.5 PPO algorithm

From the literature review could be concluded that there are three classes of DRL algorithms: value-based, policy-based and actor-critic-based methods. Each class has their own (dis)advantages and there is not one algorithms that outperforms other algorithms in all cases. Therefore, an algorithm had to be chosen based on their features. There is no unambiguous answer for this question in the literature. Some DRL scheduling studies implement a value-based algorithm (Zhou et al., 2020) & (Marchesano et al., 2021), while others used a policy-based (L. Wang et al., 2021) & (Brammer et al., 2021) or actor-critic-based algorithm (Hubbs et al., 2020).

For this case, it was decided to implement the PPO algorithm (Schulman et al., 2017), which was described in detail in section 2. The PPO algorithm was chosen because it is the latest major development in DRL algorithms, based on the TRPO (Schulman et al., 2015). It convergences faster than value-based an actor-critic algorithms and requires less computational time because it updates just the policy instead of the estimated Q-values. Furthermore, the PPO algorithm has been proven to be better suitable for high dimensional state space as compared to value-based or actor-critic-based algorithms. Due to the complex problem of this thesis, the state space becomes large and high dimensional. Another reason for choosing the PPO algorithm is its ability to handle multi-discrete action spaces. In early experimentation it was found that converting the multi-discrete actions to a single action leads to an explosion of the action space and therefore a memory error on regular notebooks. Last, the PPO algorithm is able to use action masking. Considering the high number of invalid actions in each state, this is useful for training the algorithm.

The maskable PPO algorithm of Stable Baselines 3 (S. Huang & Ontañón, 2020) was implemented for this thesis since it is able to handle both multi-discrete action spaces and action masking. Furthermore, Stable Baselines is known for its easy implementation and uses OpenAI Gym as framework for the simulation environment. Moreover, the package has an option to include action masking when the agent is exploited on new situations. It is likely that this leads to better results since there are many invalid actions in this particular case, thus a higher chance of invalid action selection by the trained agent.

4.6 Offline reinforcement learning

During the literature review, it became clear that offline reinforcement learning is a promising and powerful technique that can be used to train an RL agent when interaction with the environment is difficult. However, the technique is relatively new and little research has been done in applying offline RL to scheduling problems. Moreover, no studies have been found that investigated such a complex problem as the case in this thesis. This makes it difficult to predict the performance of these algorithms. Furthermore, not one algorithm appeared to be superior to other algorithms in all studies, as was expected since this is similar to online RL. Therefore, two offline RL algorithms were trained and compared in order to assess their performance. Based on the literature review, conservative Q-learning and batch-constrained Q-learning were selected and implemented for this project.

Both algorithms were implemented with the d3rlpy library (Kumar et al., 2020) & (Fujimoto et al., 2019). The d3rlpy library is one of the most popular offline RL python package. It is easy to implement due to the ready to use algorithms. The algorithms use the standard transitions dataset as input and provide various metrics to assess the quality of the algorithm. Furthermore, the trained agent can be tested on online environments. The only disadvantage of these algorithms is the fact that multi-discrete action spaces are not supported. However, it should be mentioned that no existing offline RL algorithm can handle multi-discrete action spaces. Therefore, the dataset

required an extra pre-processing step before it could be used to train the offline RL algorithms. From the collected transition datasets, all unique actions were retrieved. Each unique action got its unique identifier. With this identifier, the multi-discrete actions were transformed to single actions. Subsequently, these single actions could be used to train the offline algorithms. After training, the single actions that were chosen by the agent had to be transformed to multi-discrete actions before they could be used.

4.7 Benchmark algorithm

In order to assess the quality of the proposed DRL algorithm, a benchmark algorithm was developed. Ideally, this was the current scheduling technique of premix manufacturers. However, as was described in the problem statement, the lack of clear schedules rules was the reason for the study. Instead, the current production and distribution scheduling is done by expert knowledge. Nevertheless, the main considerations are known and these rules were incorporated in the benchmark algorithm. To start, the job sequence is determined based on order due date and contamination. Furthermore, the storage allocation is done based on best fitting storage space. Last, the truck allocation is determined based on the best match between truck departure and order due date. These scheduling rules were, where possible, included in the benchmark algorithm.

The benchmark algorithm was based on a greedy algorithm, since greedy algorithms produce a reasonable solution in a short amount of time and are relatively easy to implement. An algorithm is greedy when it makes local optimum decisions at each decision moment (Cormen et al., 2022). Generally, this does not lead to an optimal solution, instead it leads to an approximation of the global optimum. Therefore, this type of algorithm design is suitable for assessing the quality of the proposed DRL algorithm. When the results of the greedy algorithm approximate the results of the DRL algorithm, it can be concluded that the DRL algorithm is at least near the global optimum. Due to the different nature of the actions in the thesis's case, a greedy algorithm was developed were each action was decided with a different rule. The four actions can be divided into the following sub-problems:

1. dispatching problem, deciding which job to produce next on a machine.
2. machine allocation problem, deciding which machine to allocate a job to.
3. bin packing problem, deciding which storage space to store a finished job.
4. bin packing problem, deciding which truck to allocate an order to.

The decision rules were chosen based on expert knowledge from (customers of) KSE and the literature. Two greedy algorithms were developed. One where contamination was not accounted for in the first decision and one algorithm were contamination was accounted for in the first decision. This was done to assess the impact of contamination on the performance of the algorithms. The pseudocode of the greedy algorithms is shown in Algorithm 13. Each action was triggered when a decision was required in the simulation environment. The greedy algorithm which did not consider contamination in the first action scheduled the subsequent job based on the earliest due date. The earliest due date dispatching rule was chosen, since the objective of the model was to maximize the on time delivered orders. An exception was made for the second mixing line since it could not produce bulk products. Therefore, the first small or big bag job was scheduled on the second mixing line. The greedy algorithm that considered contamination scheduled the job with the earliest due date if its due date was lesser than the total production time plus contamination cleaning time. Otherwise, it scheduled the first non contaminating job.

The second action was allocated based on the shortest processing time and chose the sacking line that had no setup time. If that was not possible, the first available sacking line was chosen. The shortest processing time was chosen because shorter processing time leads to earlier order delivery and thus a smaller chance of late delivery. Furthermore, shorter processing time leads to a higher throughput, which leads to more on time delivered orders. Although the third and fourth action were both a bin packing problem, they were scheduled based on a different rule. The third action was scheduled based on the best fitting storage space, i.e. the storage space that led to the least loss of capacity. This was consulted with a customer of KSE. To find the best fitting storage space, the quantity of the job was subtracted from the storage space's capacity. Then, from all positive values (including zero) the first match was chosen. For the fourth action, the first fitting truck was chosen, since that truck had the earliest departure time. Thus the highest chance of delivering the order on time. In addition, ample truck capacity was assumed. Therefore, optimal usage of truck capacity was less needed than optimally using storage space.

Algorithm 13 Greedy method

```
for Earliest due date greedy algorithm do
    for idle mixing machine do
        if idle mixing machine is first mixing machine then
            schedule job with earliest due date
        else
            schedule first non bulk job with earliest due date
        end if
    end for
end for
for Contamination greedy algorithm do
    for idle mixing machine do
        if idle mixing machine is first mixing machine then
            if job with earliest due date contaminates and its due date > total production
                time + contamination cleaning time then
                schedule first non contaminating job
            else
                schedule job with earliest due date
            end if
        else
            if first non bulk job with earliest due date contaminates and its due date >
                total production time + contamination cleaning time then
                schedule first non bulk and non contaminating job
            else
                schedule first non bulk job with earliest due date
            end if
        end if
    end for
end for
for finished mixing machine do
    retrieve possible sacking machines based on packing type of job
    calculate total production time for each sacking machine
    choose first sacking machine with least production time
end for
for finished sacking machine do
    calculate fit for each available for storage space
    choose first available storage space with least capacity waste
end for
for order in storage do
    sort trucks based on departure time
    choose first truck with sufficient capacity
end for
```

5 Data description

Data from a customer of KSE was analysed in order to capture the real environment's dynamics in the simulation model. The data was obtained from a premix manufacturer based in the Netherlands and includes information about the factory layout, production and distribution process. The parameters of the simulation environment were chosen based on the data. Furthermore, order data was used for the case study to make the experiment as realistic as possible.

5.1 Production & distribution setting

The factory has two mixing lines with a capacity of 2.000 kg per mixing line and a processing time of 6 minutes per batch. Each mixing line is connected to one big bag sacking line and one small bag sacking line. Big bag sacking lines have one buffer silo, while small bag sacking lines have two buffer silos. The big bag sacking lines have a processing rate of 8.000 kg per hour, while the small bag sacking lines have a processing rate of 10.000 kg per hour. During the startup phase and at the end of sacking, the machine cannot produce at full power due to faltering supply of materials. In the experiments, hitches in the sacking process were neglected due to its minimal influence on the planning. There are 30 storage spaces for bagged products with a capacity varying between 6 and 13 pallets and one pallet can hold up to 1.000 kg of bagged product. Only one batch of bulk product can be stored in the warehouse after it is mixed. This factory layout was copied in the simulation model to get the most useful results for KSE.

5.2 Orders

To get a general understanding of the number of orders arriving during the day, the ordered quantity per order and product type per order, order data was analysed. The dataset included arriving orders in a time interval of two months. There were 493 orders in the dataset with columns containing information about arrival date, quantity, packing type and due date. 44% of the orders requested big bags, 40% of the orders requested small bags and 16% of the orders requested bulk shipment. As shown in Figure 7, the ordered quantity depends on the requested packing type; bulk products are ordered in larger quantities than bagged products. Within bagged products, small bag orders tend to have a larger requested quantity than big bag orders. Therefore, a distinction was made between the packing types when assessing the quantity of a typical order. Based on these distributions, a weight was given to each possible quantity in the simulation model.

Based on the order type distribution, the following discrete distribution was used (Table 2). Please note that these weights are relative to each other. Although there were orders in the dataset with a quantity larger than 13, it was decided to exclude these values from the simulation. Orders with larger quantities would have to be stored in multiple storage locations before shipment, which is out of scope for this thesis.

The due dates were determined based on the difference between arrival date and due date in the dataset. It was found that orders arrive most often seven days before the due date, followed by

Table 2: Quantities with weights used in simulation model

	Quantity (weight)					
Bulk products	4 (5)	6 (8)	8 (8)	10 (5)	16 (2)	20 (1)
Small bags	1 (5)	2 (5)	4 (2)	8 (2)	10 (2)	12 (8)
Big bags	1 (7)	2 (2)	4 (3)	6 (2)	8 (2)	10 (2)

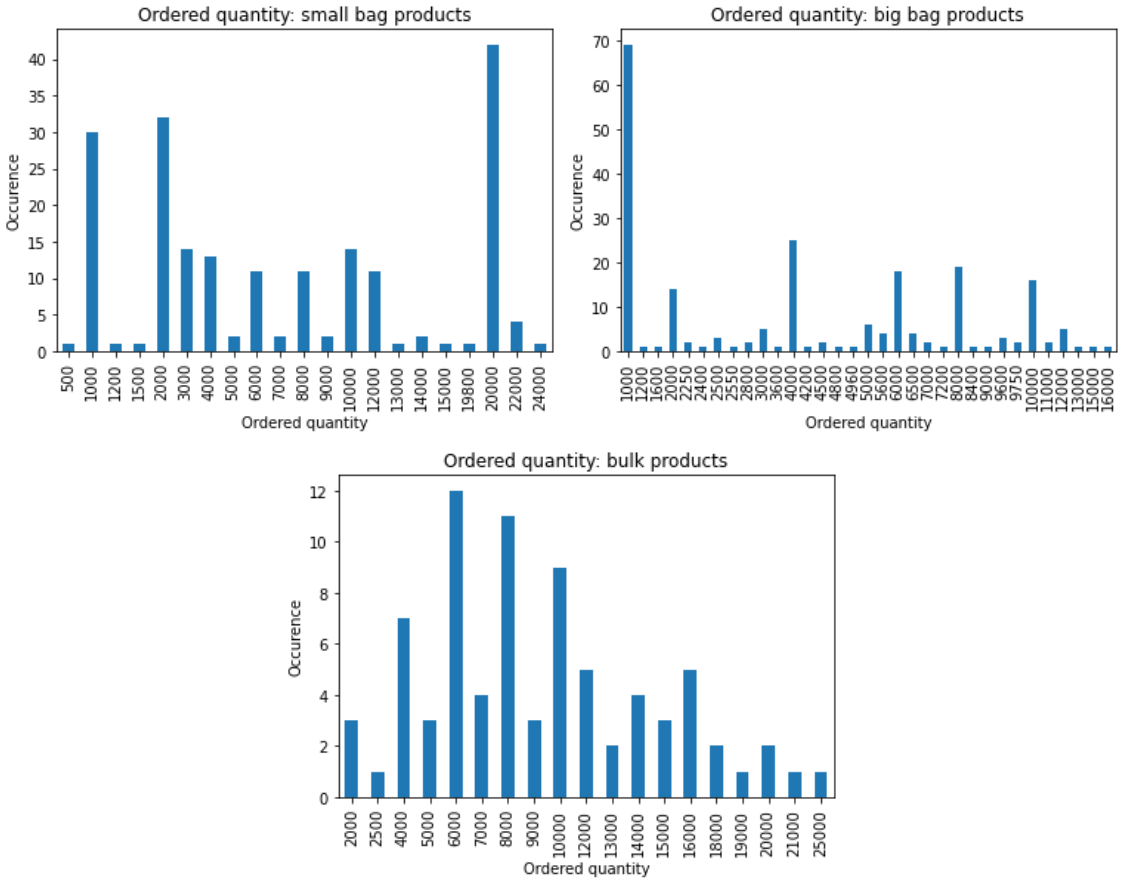

Figure 7: Histograms of ordered quantities per packing type

Table 3: Due dates with weights used in simulation model

	Due date (weight)					
Due date	1 (1)	4 (5)	5 (5)	7 (8)	8 (5)	9 (3)

eight days and four days (Figure 8). Furthermore, it is noticeable that some orders arrive 21 days or even 28 days in advance. After consultation with KSE's customer, it became clear that these orders were from foreign customers. Since the experiments will be executed for 3 working days, these orders were excluded from the simulation model. The due dates that were used in the simulation model, together with the weights, are shown in Table 3.

To determine the order arrival rate, the orders were grouped and counted based on their arrival day (Figure 8). The arrival rate of incoming orders was modelled with a Poisson distribution. The Poisson distribution is well suited to model the occurrence of an event within a fixed time interval when certain assumptions hold. The assumption that events occur independently was violated in this case, since orders could be from the same customer and therefore affect each other. Nevertheless, the Poisson distribution is a good approximation of the arrival rate of incoming orders. To find the arrival rate, the mean number of orders arriving per day was used. This turned out to be 15,9 orders per day. By using this value for the arrival rate together with all other parameter settings (as described in the data description), all orders were delivered on time. This was likely due to less product types in the simulation model as compared to the real life situation. Less product types lead to less contamination cleaning time and therefore a higher

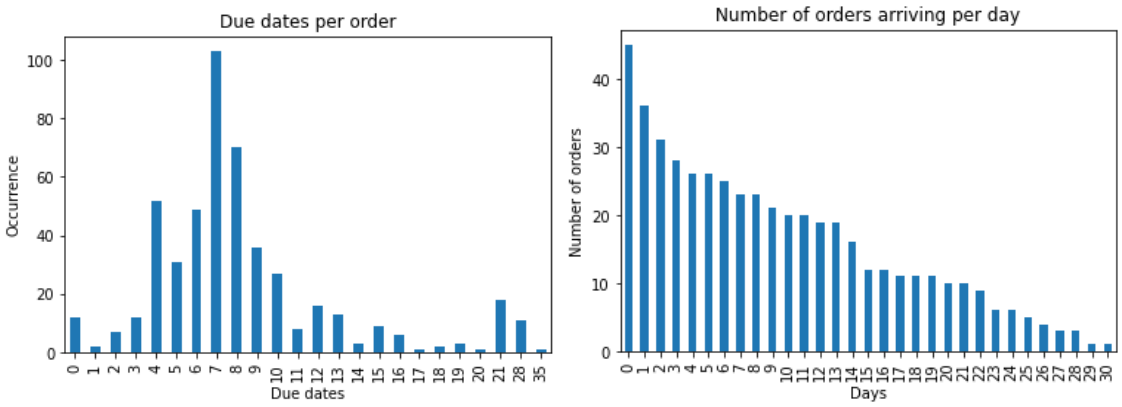

Figure 8: Due dates & number of orders arriving per day

production rate. To overcome this problem, a different value for the arrival rate was used. The used arrival rate was based on the perfect situation where no contamination cleaning time is required. The highest arrival rate which led to almost no tardy orders by a trained DRL agent was used. By trial and error, 64 orders per day was found as arrival rate.

6 Experiment description

This section elaborates on the experiments that were conducted to asses the performance of the DRL algorithm and benchmarks. To start, the parameter values of the simulation environment are discussed. Followed by the training setup of the DRL algorithm and the reward shaping. Furthermore, the scenario analysis is explained. Last, the objective metrics are elaborated on.

6.1 Simulation environment

The simulation environment had various parameters that had to be defined in order to have consistent experiments. General parameters included the total simulation time, which was set to three working days and time was measured per minute. This value was chosen because the production and distribution schedules are made on a daily bases and are adapted throughout the day. Other parameters can be divided into production-, distribution- and order parameters. These parameter settings will be discussed below.

Production parameters
The factory layout from KSE's customer was used during all experiments. Furthermore, the processing time for mixing and sacking was copied from the data. Thus, 6 minutes per mixing batch and 8.000 kg / hour for big bag sacking and 10.000 kg / hour for small bag sacking. On the other hand, some parameters could not be used due to the scope and time limits of the thesis. To start, the cleaning time between contaminating jobs was defined. Between contaminating jobs, the machine had to be cleaned, this took as long as producing a regular job. Therefore, the cleaning time equaled the production time plus dosing and transportation time, to make this as realistic as possible. In total, this was 50 minutes. Thus, 50 minutes was used as cleaning time between contaminating jobs for both mixing machines and sacking machines, although transportation time was neglected. An exception was made for the storage of finished jobs, because the job to storage decision and order to truck decision would be made at the same time if transportation time was left out. Therefore, a job handling time of 15 minutes was included. In other words, there was at least 15 minutes between the third action and the fourth action for each job.

Distribution parameters
In addition, this decision was made because of the distribution process design. It was assumed that a job is transferred to a truck once it was assigned to a truck and thereby removed from the storage location. As a consequence, the storage and truck allocation action would be at the exact same moment in time without storage transportation time. Furthermore, the capacity per truck and number of trucks were chosen in a way that there was ample truck capacity for delivery. For the fact that optimal truck capacity usage was out of scope for the thesis. In the experiments, eight trucks with a capacity of 100 pallets per truck were used. Each truck left the factory exactly one time per day while the departure times were deterministic and spread throughout the day.

Order parameters
The order characteristics have a huge impact on the performance of the algorithm. Too many arriving orders may overload the system, while too little arriving orders can make the setting too easy for the agent. Both scenario's restrict the agent in learning an optimal policy. Since the arrival rate has a major impact on the performance of the DRL agent, it was decided to vary with this rate in the experiments. The investigated arrival rates are explained in the scenario analysis section. Furthermore, early experiments showed that the due dates influenced the outcome as well. The data analysis showed that most orders have a due date of about one full week after arrival, while some have a due date of multiple weeks. This was too late when the simulation ran for three working days. Therefore, a shorter due date of 260, 320 or 380 minutes was used. A random choice was made between the three options for each generated order. Furthermore, three

Table 4: Contamination matrix

	Product type 1	Product type 2	Product type 3
Product type 1	1	0	0
Product type 2	1	1	1
Product type 3	0	1	1

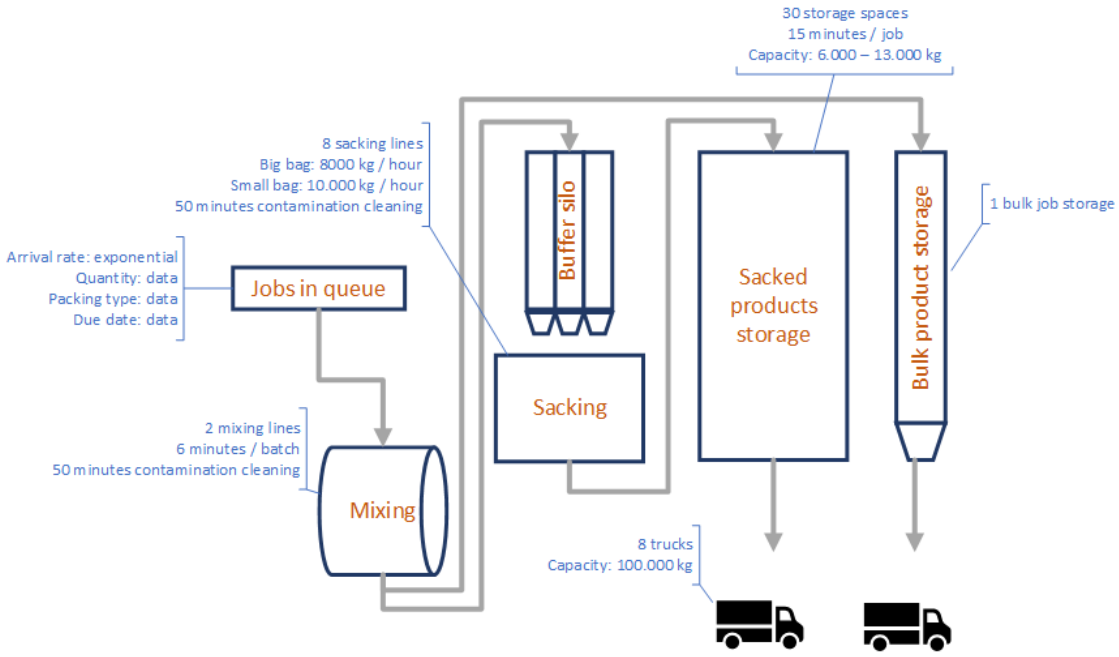

Figure 9: Overview of the parameter values in the production & distribution process

different product types were included which contaminated with each other. The contamination matrix is shown in Table 4. A one indicates that a product can be produced without setup time. For example, product type 2 and 3 cannot be produced after product type 1 without setup time, while product type 1 cannot be produced after product type 3 without setup time. Besides, the quantity of an order was adapted to simplify the experiments. Bulk products were ordered in quantities of either 6.000 kg or 8.000 kg, while small bag and big bag products were ordered in quantities of 1.000 kg or 2.000 kg, and 1.000 kg or 4.000 kg respectively. As can be observed from Figure 7, the used distribution differs from the distribution in the dataset. Furthermore, the occurrence of each packing type equaled the distribution in the dataset, namely bagged products were requested 5 times more often than bulk products.

6.2 Training setup

Besides the environment, the parameters of the DRL algorithm had to be defined as well. For each experiment, a new DRL agent was trained to achieve optimal results. Furthermore, the robustness of the DRL agents were assessed by testing the trained agents on different scenarios. The training was done via interaction with the simulation environment. During both training and exploitation, an episode terminated after three working days. From the dataset could be concluded that one workday equals two shifts of eight hours. Therefore, an episode equaled 2.880 simulated minutes. Furthermore, most of the original training parameters in the work of Schulman et al.

Table 5: Reward shaping: weights per reward component

	Overcapacity	Undercapacity	Low uncertainty	High uncertainty
weight on time delivered orders	0,99	1,46	0,78	1,49
weight makespan	0,74	1,41	0,50	0,78

(2017) were used, because the PPO algorithm does not require much hyperparameter tuning. This included the batch size of 64, a clipping range of 0.2 and 10 epochs for optimizing the surrogate loss function. The original discount factor $\gamma = .99$ was used as well, while the learning rate was different. A linear decreasing learning rate was used in order to promote exploration during the early phase of training. The original parameter was set to .0003, while in the thesis a starting value of .0005 was included which linearly decreased each policy update until zero. Moreover, the original neural network design from Schulman et al. (2017) was used. This means that the neural network contained a fully connected MLP with two hidden layers of 64 units. Besides, tanh nonlinearities and a mean Gaussian distribution output were used. To get reliable results, all agents were trained until there was no reward increase for 20 episodes. This termination criteria was chosen because longer training was not possible with the available resources.

6.2.1 Reward shaping

The Bayesian optimization technique with Tree-structured Parzen Estimator (TPE), as discussed in section 4, was used to find the optimal weights of each reward function component. For all previously mentioned scenarios, the Bayesian optimizer was applied. This was done separately for each scenario because it was expected that the weights of the reward function components would differ per scenario and therefore led to better results per scenario. In the uncertainty cases it could be beneficial to provide a bigger weight to the intermediate reward component (makespan), because it could be harder to find an optimal policy due to the noise of unpredictable events. Running the optimizer until it has found the optimal results was computationally too inefficient, therefore it was constraint by limiting the number of runs. Since there are only two components of the reward function that can be shaped, 20 runs were used to optimize the reward function. The results are shown in Table 5. In all cases, the on time delivered orders component received a higher weight, which was expected. The weights in the undercapacity case are almost the same, while the weights of the high uncertainty case are relatively far apart.

6.2.2 Offline reinforcement learning

In order to train the offline RL algorithms, a training dataset was required. As was explained in section 2, the dataset required information about observations, actions, reward and termination. There are five different dataset types that can be used to train an offline RL algorithm: random, expert, mixed, noisy and replay datasets (Schweighofer et al., 2021). In these experiments, both expert and mixed datasets were used. Expert datasets are datasets that were generated with a trained online RL agent. Since the agent is trained, no exploration takes place. This results in a dataset where only the best actions are included and thus will likely lead to a good performing offline RL algorithm. However, the goal of the offline RL is to find patterns that the online RL agent failed to discover. Therefore, it might be beneficial to include random data in the dataset. This is the definition of a mixed training dataset. The mixed training dataset was made by collecting training data from the online RL agent. For all offline RL experiments, the data was split in 80% training data and 20% test data. For offline RL algorithms, there were again many different hyperparameters to tune. In this case, all default settings were used.

6.3 Scenario analysis

A scenario analysis was done to assess the performance of the algorithm under different circumstances. In all experiments, the factory layout from the case study was used. Varying with the number of production lines, storage spaces and other factory dependent parameters was not the priority for KSE. It is in their interest to know how an algorithm performs with different production scenarios and under uncertainty. Therefore, an experiment was conducted where the algorithm and benchmarks were tested against cases with (in)sufficient resources. After that, the most important experiment was done, namely where uncertainty was introduced.

6.3.1 Under- & overcapacity

The first experiment was about testing the algorithm on a case with undercapacity and a case with overcapacity. During both experiments, the production time for all machines was deterministic, as well as the truck departure time. Furthermore, three different product types were included, where the first product type contaminated with the other products and the third product contaminated with the first product (Table 4). All three product types had an equal chance of being ordered. Besides, an arrival rate of 3 orders per hour was used in the overcapacity case. After all, the arrival rate generally leads to overcapacity of the resources, thus optimal usage of the resources was not required. Nevertheless, it was interesting to compare the algorithm with the benchmarks for this case since it gave a quick evaluation of the algorithm. Besides, comparing the results of the overcapacity case with the undercapacity case gave insights in the robustness of the model.

For the undercapacity case, an arrival rate of 4.8 orders per hour was included. Although this rate is relatively close to the arrival rate of the small case, it was expected that this rate led to under capacity of the resources. For the reason that the arrival rate of the small case led to a balanced system if no setup times were included for contaminating sequential jobs. Besides, more orders were tardy due to undercapacity of the resources. Therefore, the agent had to find a way to deliver as much orders on time as possible.

6.3.2 Uncertainty in production & distribution

While in the previous described experiments all events were certain, it was particular interesting to evaluate the algorithm when unexpected events occur. Apart from the fact that these settings were closer to reality, deep reinforcement learning has to ability to deal with uncertainty in the environment. Therefore, it was interesting to compare the performance of the agent against the benchmarks in scenarios where uncertainty existed. In the experiments, two events occurred with a certain probability to simulate uncertainty:

1. **Equipment downtime**. Machines could be down due to equipment failure. These machines had to be repaired before they could be used to process orders. This was modelled in the simulation through extra processing time of two hours for the order that was processed at the time of equipment failure.
2. **Truck delay** Trucks could be delayed during their route. This led to a later departure time from the factory than expected. This was modelled in the simulation by an additional 60 minutes to the original departure time of the truck.

These events were included because they happen most often in reality. In the first experiment, there was relatively low uncertainty and these events occurred with a probability of 3%. In

the second experiment, there was relatively high uncertainty and these events occurred with a probability of 15%.

6.4 Case study

Last, a case study was executed to assess the performance of the DRL agent in a situation that is most comparable to the real life setting of a premix feed producer. In this experiment, most parameter values that were found in the data analysis were used. This included the due dates and quantities for the orders. Besides, 15 product types were included to make the contamination part of the process more realistic. The product types varied in the number of other product types they contaminated with. Some product types could be produced after any other product types while others could only be produced after a few other product types. Since the number of product types was still much lower than reality, a higher order arrival rate was used than the rate in the dataset. Including over 300 product types would require much more computational time. The order arrival rate was set to 3 orders per hour. Furthermore, uncertainty was included in the simulation model to make the experiment as realistic as possible. Machine breakdown and truck delay could occur with a probability of 3%.

6.5 Performance metrics

This section provides an overview of the objectives and performance metrics for all experiments. These variables were used to evaluate and compare the algorithm and benchmarks.

1. **Number of delivered orders**. The number of delivered orders is the most important objective. This number reflects the performance of the algorithms since an efficient planning leads to a higher production and distribution. However, this metric alone was not enough to determine the quality of an algorithm. From the number of delivered orders could not be determined how many orders were delivered on time.

2. **Number of tardy orders**. Therefore, the number of tardy orders were included as well. With this information, the quality of the algorithm could be better assessed. Yet the number of tardy orders alone did not provide much information. Delivering more orders on time at the expense of a few extra tardy orders was better.

3. **Makespan**. The makespan is referred to as the completion time of a job from arrival to sacking. Comparing the average makespan of all jobs between the algorithm and benchmarks, provided an understanding of the efficiency of the algorithm. A lower makespan showed that the algorithm was able to efficiently schedule the jobs.

4. **Number of contamination cleaning**. The number of contamination cleaning showed how well the algorithm was able to deal with sequence-dependent setup times. The times contamination cleaning was required had to be reduced to a minimum in order to maximize the resource usage.

7 Results

In this section, the experimental results are showed and discussed. First, the learning behaviour of the DRL algorithm is discussed. Then, the results of the scenario analysis are elaborated on. Followed by a discussion of the case study. Afterwards, the offline RL results are shown. Then, the results of the robustness analysis are elaborated on. Last, a discussion of the results is given.

7.1 Learning behavior

Before the actual results are discussed, the learning behaviour of the DRL agent is assessed for each experiment. The learning curve is the moving average of the reward over the training episodes and shows if and how the agent learns throughout the episodes. Furthermore, this provides insights in the quality of the DRL framework, i.e. if the states, actions and rewards are well designed.

7.1.1 Overcapacity & undercapacity

Starting with the overcapacity case. As can be observed in Figure 10, the reward quickly increased over the episodes and converged after about 150 episodes around a reward of 120. Then, the reward function remained stable with a little fluctuations. Randomness in the order generation could explain the behaviour, since the behaviour was not detected in the experiment where the same order sequence was used. For the undercapacity case, the agent was able to learn throughout the episodes as well (Figure 10). However, this case was more difficult to learn for the agent since the reward increase was more gradual as compared to the overcapacity case. Besides, some fluctuations can be seen throughout the early stage of training, indicating that the optimal policy was harder to find. In addition, the variation after convergence was higher than the overcapacity case, which seems logical since the policy was harder to find for the agent. Last, in both cases the learning quickly converged (after 150 and 400 episodes respectively) and can be explained by the fact that action masking was used for all experiments. Learning the same behaviour to a DRL agent without action masking took much more training episodes during early experimentation.

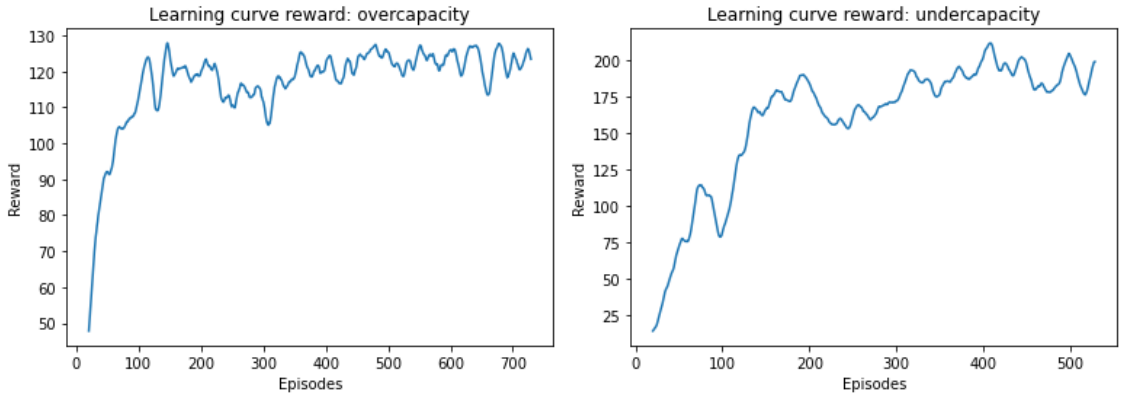

Figure 10: Moving average episodic reward overcapacity & undercapacity. Both DRL agents learned throughout the episodes.

Another important learning indicator is the ability of the agent to learn the contamination rules. Besides scheduling the order with the earliest due date, smartly sequencing orders based on contamination will eventually lead to the best results. Thus, failing to learn this rule will make the policy sub-optimal. The moving average for the number of contamination cleaning used is shown

in Figure 11 for the overcapacity and the undercapacity. From these graphs can be concluded that the agent was able to learn the contamination rules, since the number of contamination cleaning usage decreased for the undercapacity case. The contamination usage stabilized around 30% although the variance was relatively high. On the other hand, Figure 11 shows a steady number of contamination cleaning used for the overcapacity case, which can be explained by the fact that there were little jobs waiting in the queue. In reality, overcapacity will not occur often, therefore the inability to learn contamination in the overcapacity case is not an issue.

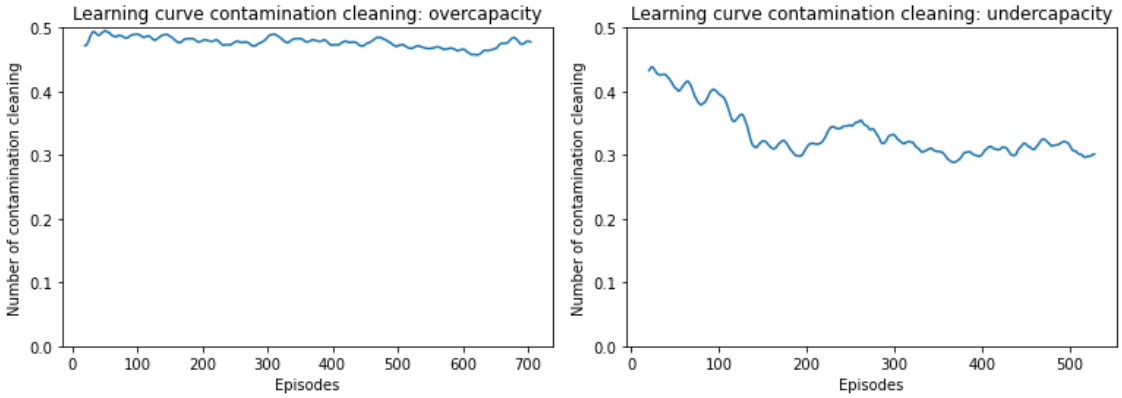

Figure 11: Contamination cleaning usage overcapacity & undercapacity. Clear learning was only visible for the undercapacity DRL agent.

7.1.2 Uncertainty in production & distribution

For the uncertainty cases, the learning behaviour is discussed as well. Figure 12 shows the learning curve for the uncertainty cases. From the left graph can be concluded that the agent was able to learn a good policy in low uncertainty. The reward quickly increased to about 80, followed by a gradual increase with more variation to about episode 150. After that, the reward stabilized and fluctuated around a reward of 100. Note that both uncertainty cases were based on the arrival rate of the overcapacity setting. Logically, the fluctuations after convergence can be explained by the fact that uncertainty occurred throughout the episodes. The uncertainty made it harder for the agent to learn the optimal policy, which can be concluded through the fact that convergence took longer in this case. Regarding the high uncertainty case, an increase in the reward function is observed as well. The reward function converged around 175 episodes, while it fluctuated during the early stage of training. Furthermore, heavy oscillations are observed after convergence, which can be explained by the uncertainty aspect of the experiment again. Besides, it is noticeable that the reward is much lower than the low uncertainty case, which indicates that there were significantly less orders delivered on time.

The number of contamination cleaning used is shown in Figure 13 for the uncertainty cases. In both cases, the agent was able to learn this rule as a significant decrease in the contamination cleaning usage is observed. Since the uncertainty cases used the same order arrival rate as the overcapacity case (with all other parameters kept constant), the earlier given explanation for the lack of learning in the overcapacity case can be confirmed.

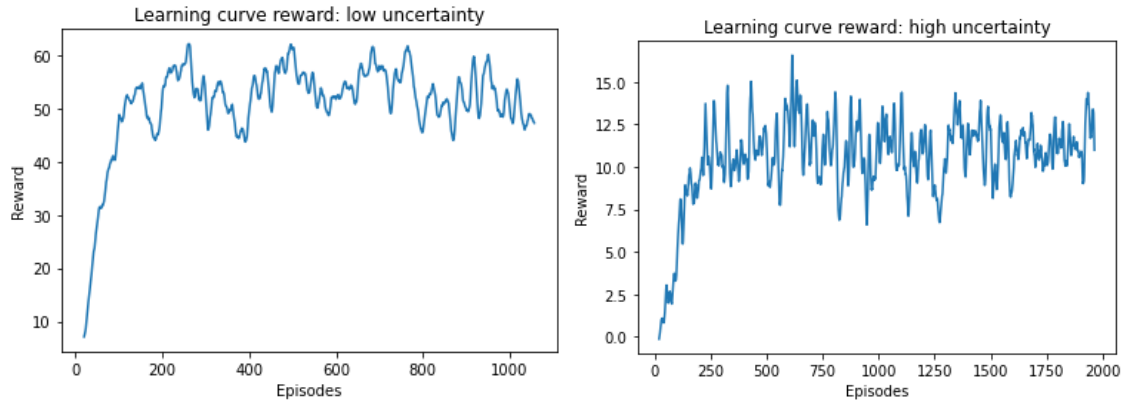

Figure 12: Moving average episodic reward low & high uncertainty. Both DRL agents learned throughout the episodes.

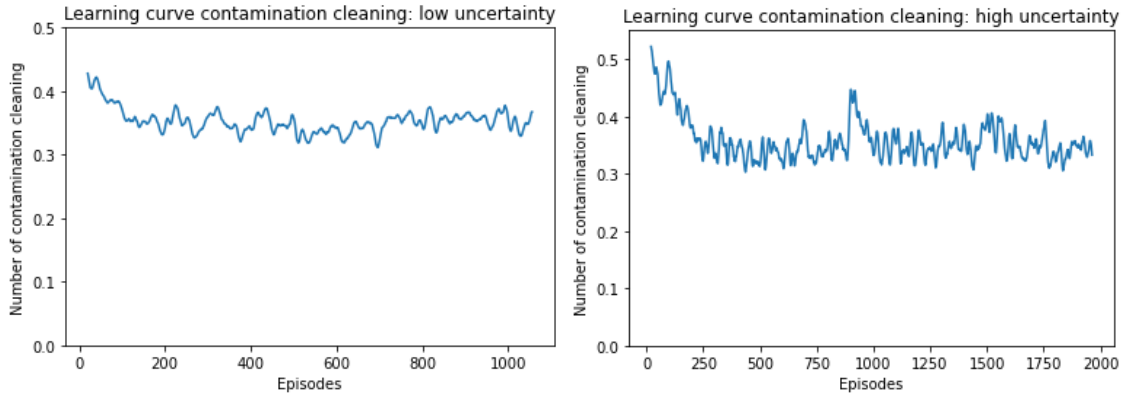

Figure 13: Contamination cleaning usage low & high uncertainty. Both DRL agents learned to avoid contamination throughout the episodes.

7.2 Scenario analysis

In order to determine the performance of the DRL agent, the trained agents were tested on the same environment as their training setting. The difference between training and testing is that during training the agent intentionally takes exploratory actions, while during testing this is not the case. Recall from the literature review that this behaviour is meant to find a better policy. The results of the experiments are shown in tables and figures. The tables include the mean of the metrics for all episodes, while the figures contain box plots per experiment. This presentation of results is based on the work of Henderson et al. (2018). Furthermore, the results from the greedy algorithms that were introduced in section 4 are included for comparison.

Table 6: Results overcapacity: mean of all metrics. Greedy contamination performed best overall.

	DRL agent	Greedy EED	Greedy contamination
Delivered orders (#)	138,05	124,55	140,9
Tardy orders (#)	16,24	25,3	5,39
Makespan (minutes)	124,14	113,86	97,15
Contamination cleaning (%)	41,96	47,05	46,51

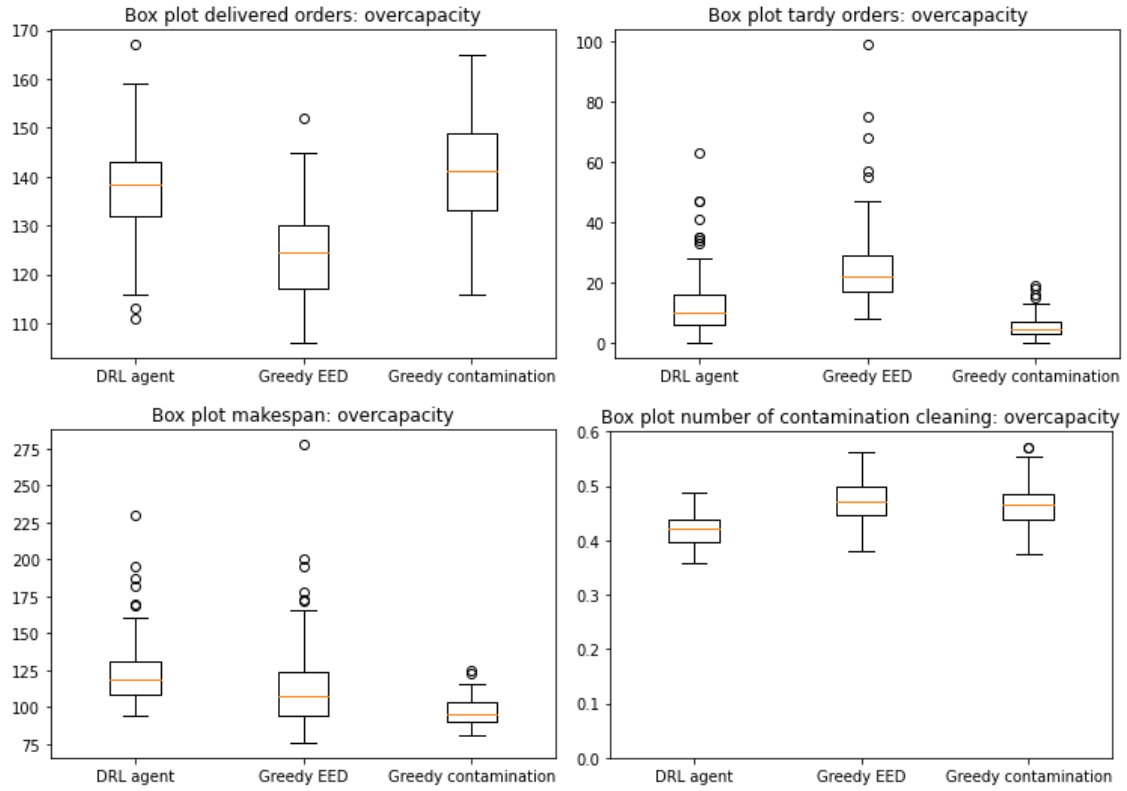

Figure 14: Box plots results DRL agent & greedy algorithms overcapacity. Greedy contamination showed stable performance, while the DRL agent had a few outliers.

Table 7: Results undercapacity: mean of all metrics. Greedy contamination performed best overall.

	DRL agent	Greedy EED	Greedy contamination
Delivered orders (#)	198,68	168,38	209,01
Tardy orders (#)	86,60	160,39	37,02
Makespan (minutes)	202,25	393,44	152,18
Contamination cleaning (%)	30,53	47,14	40,11

The DRL agent performed worse than the contamination greedy algorithm in terms of on time delivered orders (Table 6). The number of delivered orders were comparable, showing that the DRL agent was able to define a good policy for the production part. However, the number of tardy orders were worse (16.24 vs 5.39). The higher average makespan (27,78%) could be an explanation, although the number of contamination cleaning used is 9.79% lower. Since the makespan includes the time between order arrival and production, the higher average makespan could be a sign that the DRL does not always schedules the earliest arriving job. Minimizing the makespan itself is not the overall objective, it is rather a way to deliver as much orders as possible on time. Thus, choosing between the job with the earliest due date and the non contaminating job is the desired behaviour. The DRL agent did outperform the earliest due date greedy algorithm in terms of on time delivered orders. Last, it should be noted that the contamination cleaning used by the greedy algorithms is somewhat comparable, which is due to the overcapacity. After all, with overcapacity there are less jobs in the queue, thus less jobs to avoid contamination. The box plots (Figure 14) show that the DRL agent had more outliers than the contamination greedy algorithm. The number of delivered orders varied for all algorithms, while the number of tardy orders was relatively

stable, especially for the contamination greedy algorithm. Furthermore, the stable contamination cleaning usage of the DRL agent is striking, since Figure 11 shows that the agent was not able to improve the contamination use over the episodes.

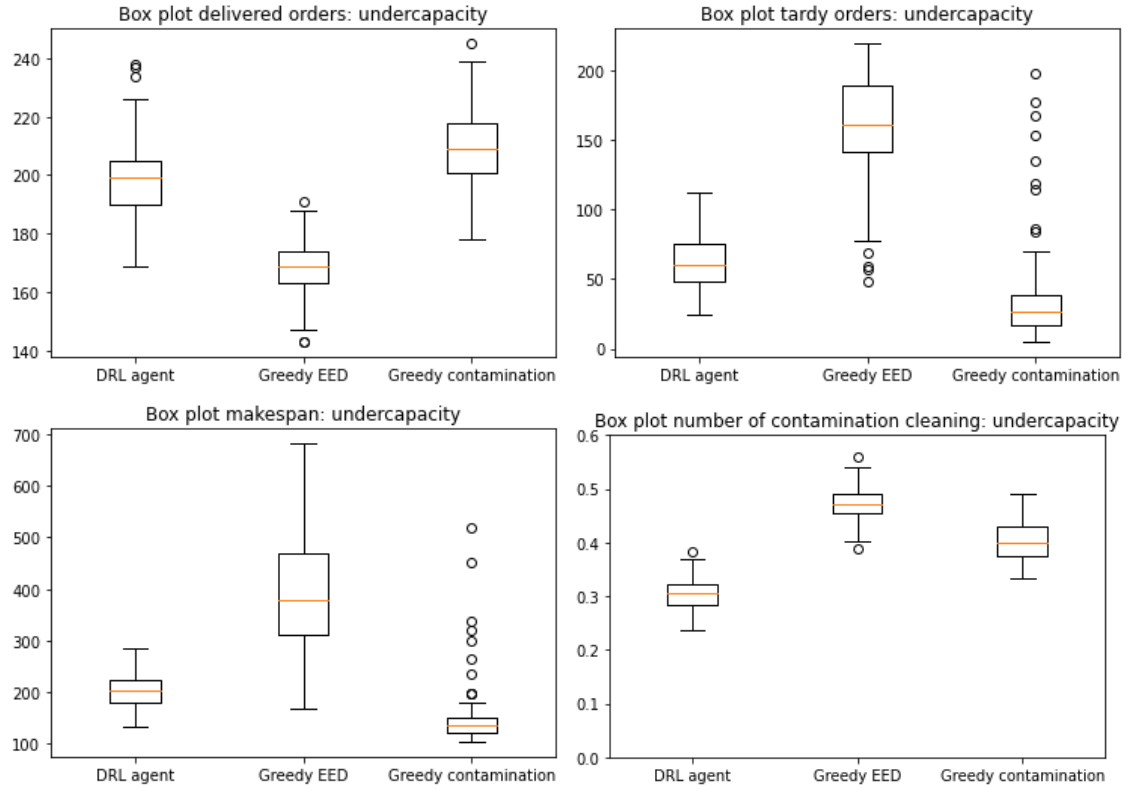

Figure 15: Box plots results DRL agent & greedy algorithms undercapacity. Greedy contamination had many outliers, while the DRL agent performed stable.

For the undercapacity case, the difference between the DRL agent and the contamination greedy algorithm was more prominent. As can be seen in Table 7, the contamination greedy algorithm delivered 53.45% more orders on time. Again, the earliest due date algorithm performed worse than the others on all metrics. Besides, the difference in contamination cleaning usage between the greedy algorithms became clear in this experiment. The contamination cleaning usage of the DRL agent was significantly lower than the greedy algorithms, while the number of delivered orders was lower than the contamination greedy algorithm. The number of contamination cleaning used shows that the DRL agent was able to make good job scheduling decisions, yet failed to define an optimal policy for the truck allocation. However, the box plots (Figure 15) show that the DRL agent defined a more stable policy than the greedy algorithms. Especially the contamination greedy algorithm had many outliers in the number of tardy orders and makespan.

The results of the low uncertainty experiment are shown in Table 8 and Figure 16. In this case, the DRL agent was able to outperform the greedy algorithms. Again, the number of delivered orders from the DRL agent and contamination greedy algorithm were comparable (123.40 vs 120.15), while the number of tardy orders were 15.13% lower for the DRL agent. The makespan and number of contamination cleaning usage was lower for the DRL agent, which could explain the performance gap. The difference between the greedy algorithms is noticeable. The number of contamination cleaning usage was slightly lower for the contamination greedy algorithm and led to 27.65 more on time delivered orders. The box plots (Figure 16) show that the variance in number of tardy orders throughout the episodes was high for all algorithms. On the other hand,

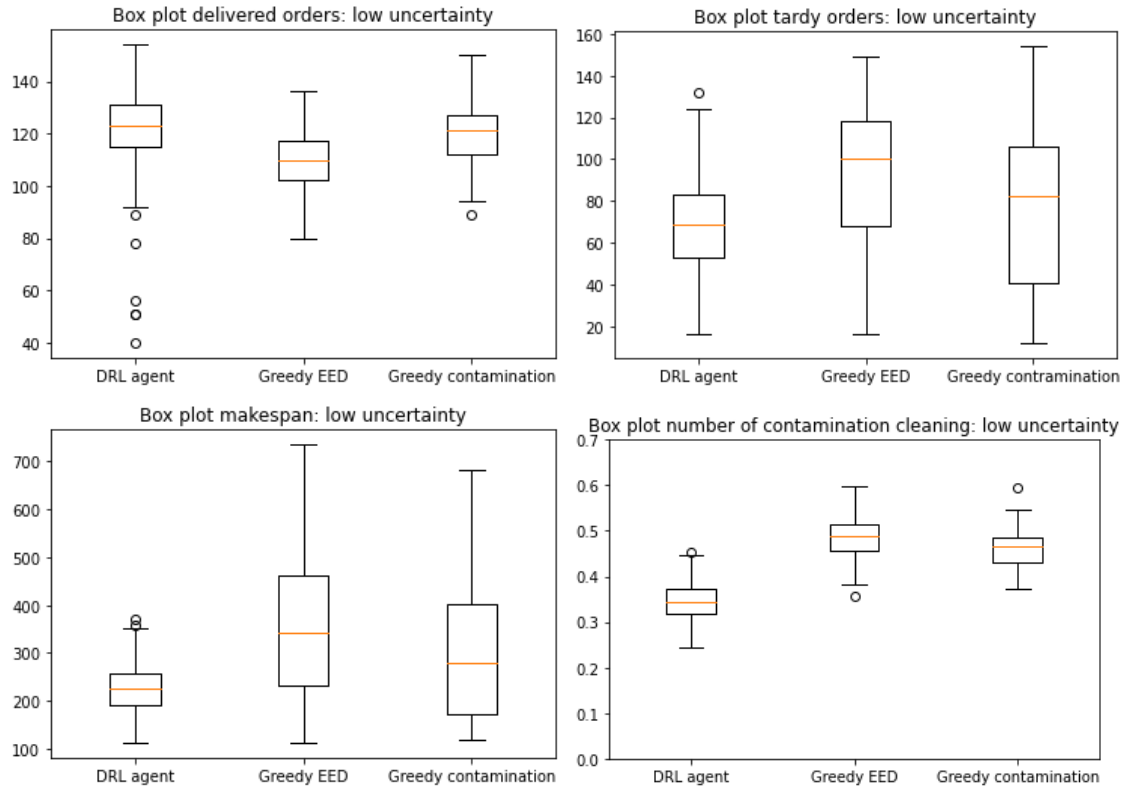

Figure 16: Box plots results DRL agent & greedy algorithms low uncertainty. DRL agent had the most consistent results.

Table 8: Results low uncertainty: mean of all metrics. DRL agent performed best.

	DRL agent	Greedy EED	Greedy contamination
Delivered orders (#)	123,40	108,89	120,15
Tardy orders (#)	64,92	92,88	76,49
Makespan (minutes)	220,84	355,84	296,96
Contamination cleaning (%)	34,67	48,58	46,13

the makespan of the DRL agent was relatively stable as compared to the greedy algorithms. This shows that the DRL agent was able to handle the uncertainty in production, while it struggled to deal with uncertainty in the distribution.

The high uncertainty case shows that none of the algorithms delivered orders on time on average, which is shown in Table 9. Note that the number of tardy orders included the orders that remained in the system after episode termination, thus orders could have been delivered on time. The positive reward function of the DRL agent (Figure 12) indicates that orders were delivered on time. The DRL agent delivered slightly more orders, although more tardy orders remained in the system as well. The objective of the optimization problem could explain the behaviour, since the objective was to maximize the on time delivered orders. Thus, delivering an order on time was more valuable than delivering more orders late. In addition, the makespan of the DRL agent was more than twice as low as the makespan of the greedy algorithms (54.39% and 53.70% respectively). Besides, the contamination cleaning usage of the DRL agent was significantly lower than the greedy algorithms. The box plots (Figure 17) confirm this belief. The results from the greedy algorithms were somewhat comparable, while the DRL agent differed. The number delivered and tardy orders

Table 9: Results high uncertainty: mean of all metrics. All algorithms performed comparable, while the makespan of the DRL agent was significantly lower.

	DRL agent	Greedy EED	Greedy contamination
Delivered orders (#)	44,69	40,56	42,38
Tardy orders (#)	132,56	126,41	129,74
Makespan (minutes)	471,62	1033,97	1018,51
Contamination cleaning (%)	33,32	54,63	55,07

varied greatly throughout the episodes, while the makespan was relatively stable. Furthermore, the maximum value of contamination cleaning usage was about the same as the average of the greedy algorithms. The DRL agent had many outliers, while the greedy algorithms did not show such behaviour. The characteristics of DRL could explain the behaviour, since DRL agents try to find a policy within the uncertain environment and are therefore affected by the unforeseen events. On the other hand, greedy algorithms follow predetermined rules which do not change throughout the episodes. The high variance in the order metrics and relatively low variance in the makespan confirm the believe that the DRL agent was able to handle uncertainty in production, while it could not handle uncertainty in the distribution.

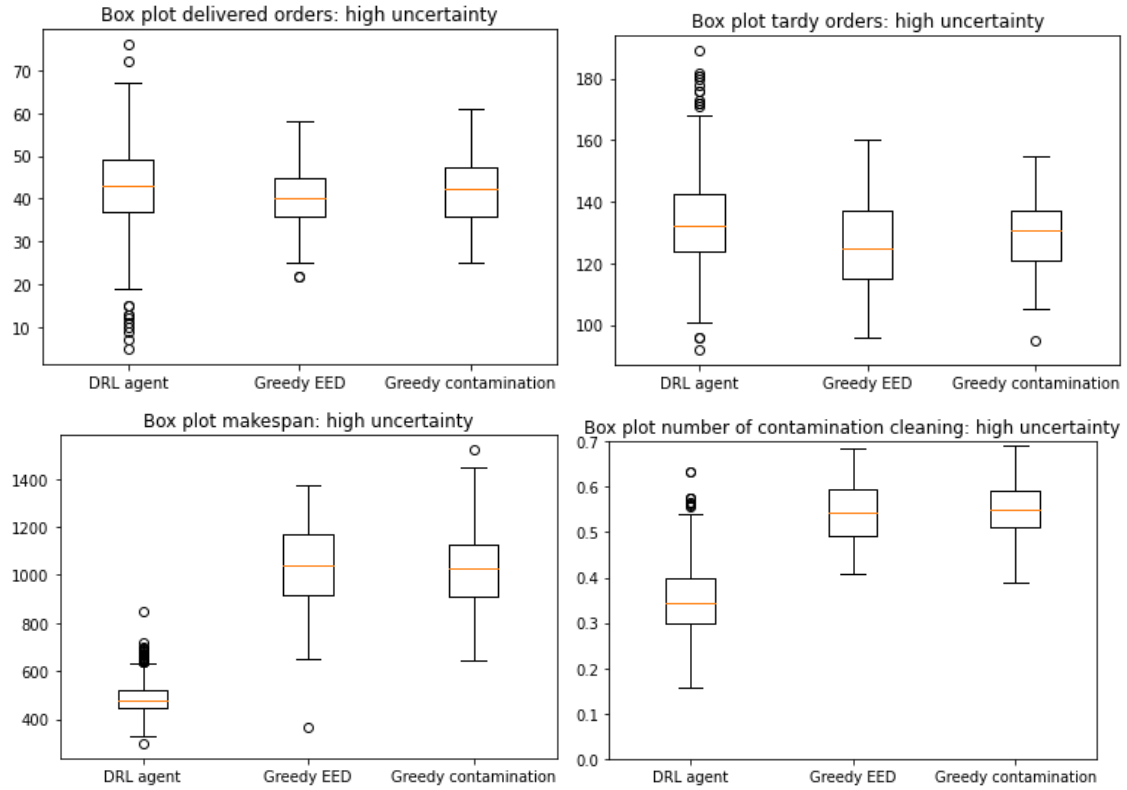

Figure 17: Box plots results DRL agent & greedy algorithms high uncertainty. DRL agent had many outliers.

7.3 Case study

For the case study, the learning behaviour and the performance of the trained agent are discussed. As can be seen in Figure 18, the agent learned throughout the episodes since the moving average

Table 10: Results case study: mean of all metrics. Greedy contamination performed best, although the DRL agent decreased the makespan significantly.

	DRL agent	Greedy EED	Greedy contamination
Delivered orders (#)	583,49	542,18	582,26
Tardy orders (#)	295,85	312,36	181,38
Makespan (minutes)	1013,52	3867,61	2461,55
Contamination cleaning (%)	47,65	56,92	43,12

reward increased. Although it can be concluded that the agent had difficulties with finding the optimal policy, as the reward function fluctuated while gradually moved upwards. These fluctuations tended to decrease in magnitude over the episodes, which indicates that agent learned a policy that is applicable in different situations. Regarding the contamination usage (Figure 18), it can be concluded that the agent gradually learned to avoid contamination during the production. The contamination cleaning learning curve decreased less steep compared to the previous experiments, which can be explained by the extra product types in the case study. The number of delivered orders increased while the number of tardy orders decreased.

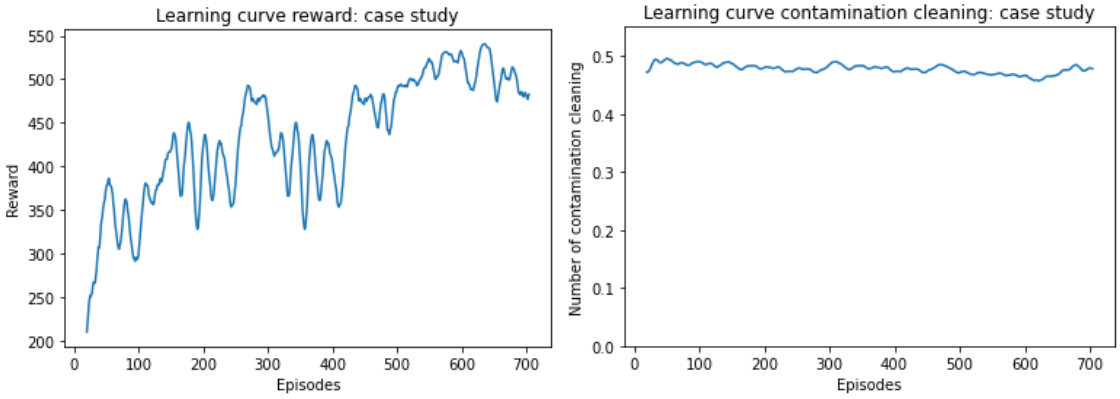

Figure 18: Moving average episodic reward & contamination usage case study. DRL agent learned throughout the episodes and contamination was gradually learned.

The trained agent was evaluated on the same environment to determine its real performance without exploration. The results are shown in Table 10. Furthermore, the results of the greedy algorithms were included in the table as well. The number of delivered orders by the DRL agent was comparable to the contamination greedy algorithm and higher than the earliest due date greedy algorithm. The number of tardy orders was higher than the contamination greedy algorithm and lower than the earliest due date greedy algorithm. Furthermore, the makespan of the DRL agent was 58.83% lower than the greedy algorithms. The contamination cleaning usage was the lowest for the contamination greedy algorithm. Although the DRL agent was able to avoid some contamination, as it was better than the earliest due date greedy algorithm.

However, there is room for improvement of the DRL agent since the number of contamination cleaning is higher than the contamination greedy algorithm. The DRL agents were able to learn these rules in the other experiments, therefore it is expected that the agent will learn this with more training time. In addition, there were sufficient jobs waiting in the queue with these parameter settings. Thus, it is expected that the performance will be even better when less contamination cleaning is needed. The lower makespan of the DRL agent shows its potential, yet it failed to find a good policy at the distribution part.

Table 11: Results offline reinforcement learning: overcapacity case training dataset. Both algorithms did not learn a suitable policy.

	CQL	BCQ
Advantage	-24,68	-14,08
TD error	3,98	2,15
Value scale	1,26	1,44

Table 12: Results offline reinforcement learning: overcapacity case trained dataset. Both algorithms did not learn a suitable policy.

	CQL	BCQ
Advantage	-36,18	-16,54
TD error	3,74	2,38
Value scale	0,75	0,59

7.4 Offline reinforcement learning

The results of the offline RL experiments are shown in Table 11 & Table 12. The table includes the average discounted sum of advantage, average temporal difference (TD) error and the average value estimation. These values assess the difference between the value function of the dataset and the value function of the generated policy (Murphy, 2005). Thus, a large value (both negative and positive) for these metrics indicate that the learned policy deviates from the policy in the dataset. Deviation from the dataset is a sign of action-value pair overestimation and overfitting on the training data. Table 11 & Table 12 show that both algorithms overestimated and overfitted on the training data, since the values for all metrics were too large. Therefore, it can be concluded that the algorithms could not find a good policy. The BCQ algorithm performed better than the CQL algorithm on all metrics except for the value scale of the training dataset case. Furthermore, the results of the training dataset were better than the trained dataset which indicates that the algorithms were able to define a better policy with the training dataset. However, the trained algorithms predicted the same set of four actions on all observations, which indicates that the training failed.

7.5 Robustness analysis

In the previous experiments, the agent was tested on the same environment as it was trained on. However, training the DRL agent is costly in terms of time and resources. Not only due to the time it takes to train the agent, as well as updating the simulation model. Therefore, it is desired that a trained agent can be used on comparable situations as its training environment. To assess the robustness of the trained DRL agents, two experiments were conducted. The trained agents from the previous experiments were used on the other environments. Their performance was then compared to the other agents. Note that, contradictory to the previous experiments, the agents were not trained on the new situations.

To start, all trained agents were used to predict the actions of the overcapacity environment. The objective metrics are shown in Table 13 per agent. The results show that all agents performed somewhat comparable. The low uncertainty agent performed better on all metrics compared to the overcapacity agent. This is striking as the overcapacity agent defined its policy on this environment, while the uncertainty agents defined their policy on the same environment plus uncertainty. The same procedure was followed for the undercapacity environment. The results of these experiments are shown in Table 14. The overcapacity agent delivered significantly less orders

Table 13: Robustness analysis: overcapacity. All DRL agents performed comparable.

	Low	High	Over	Under
Delivered orders (#)	140,25	136,56	138,05	138,64
Tardy orders (#)	13,65	16,32	16.24	15,25
Makespan (minutes)	117,45	131,26	124.14	129,88
Contamination cleaning (%)	39,65	42,69	41.96	41,67

Table 14: Robustness analysis: undercapacity. Low uncertainty agent performed best, overcapacity agent performed worst.

	Low	High	Over	Under
Delivered orders (#)	202,69	188,56	185,71	198,68
Tardy orders (#)	85,32	105,86	131,23	86,60
Makespan (minutes)	203,41	238,46	274,01	202,25
Contamination cleaning (%)	30,96	37,69	33,63	30,53

on time. However, it was able to avoid contamination cleaning since it required cleaning in 33.63% of the production jobs, which confirms the believe that it was able to learn the contamination rules. Moreover, the low uncertainty agent outperformed the other agents. However, the performance is somewhat comparable to the undercapacity agent.

For the low uncertainty case the results are shown in Table 15. The low uncertainty agent performed best. There is a significant difference between the uncertainty agents and other agents, which seems logical as they experienced uncertainty throughout the training. The low and high uncertainty agents performed comparable on all metrics. Besides, the undercapacity agent performed better than the overcapacity agent on this case. Last, the high uncertainty environment will be discussed. The results from all agents are shown in Table 16. Here, the results were closer to each other. The results from the high uncertainty agent were not significantly better than the others, showing that all agents were struggling with this environment.

7.6 Early experiments

During early stages of experimentation with the states, actions and rewards formulation, a different approach for the jobs in the queue was tested. Here, four variables with information about the product type, quantity, packing type and due date for the top 20 jobs with the earliest due date were included. However, it turned out that the agent was unable to learn a good policy with respect to contaminating products. The four variables were stored in the state space for the top 20 jobs with the earliest due date. The number of jobs had to be constant due to the requirement from the DRL framework to have a constant state space. Once a job was assigned to a mixing machine it was no longer waiting in the queue and thus removed from the state space. When there are less than 20 jobs in the queue, dummy jobs were added to the state space to ensure a consistent state space.

Table 15: Robustness analysis: low uncertainty. Low uncertainty agent performed best, overcapacity agent performed worst. Results are close to each other.

	Low	High	Over	Under
Delivered orders (#)	123,40	121,85	110,39	112,54
Tardy orders (#)	64,92	69,87	86,95	80,29
Makespan (minutes)	220,84	231,99	262,58	259,65
Contamination cleaning (%)	34,67	32,14	41,96	39,81

Table 16: Robustness analysis: high uncertainty. All DRL agent struggled in this environment.

	Low	High	Over	Under
Delivered orders (#)	42,58	43,90	41,25	42,08
Tardy orders (#)	132,96	133,38	138,62	131,65
Makespan (minutes)	506,43	490.31	537,03	500,26
Contamination cleaning (%)	46,20	35.82	45,97	42,76

Furthermore, various approaches were tested to learn the agent to deal with contaminating product types. First, the most realistic approach was tested. This was modeled by tracking the contamination history per ingredient on each machine. As described in section 3, premix feed producers keep track of the contaminating ingredients in each product type. Thus, for each product type it is known which ingredients it contains. Furthermore, it is known how many batches are required to be produced after a certain ingredient is processed before it is completely removed from a machine. Moreover, it is known what percentage of an ingredient is allowed to remain in a machine when an other product type is produced. With this information, it can be determined if a production order is allowed without intermediate machine cleaning, and the contamination history can be updated. For example, when a batch of product type x with ingredients y and z is finished on the mixing machine, the contamination history is updated. For ingredient y and z it is known that it takes two and three batches respectively to be removed completely from the mixing machine. Thus, the contamination history variables for ingredient y and z of the mixing machine are set to two and three, while all other variables are subtracted by one if greater than zero. When assigning the next job to the mixing line, the contamination history is compared to the ingredient allowance of the next job's product type to determine if cleaning is required.

During the experiments with the state space, it became clear that the DRL agent was unable to find a good policy for this complex process. Even when explicit rewards were given for job sequences without setup time and a penalty when setup time was required. Therefore, a simpler way of modeling contamination was investigated. In the new situation, ingredients were left out of the model. The contaminating product types were included in the contamination history variables and a contamination matrix was built based on the product types. The state space included an array binary variables for all product types, one meant that the product type could be produced without setup time while zero meant the opposite. Unfortunately, this method did not work either.

7.7 Discussion

The results section provides insight in the performance of the different DRL agents, as well as the greedy algorithms. It can be concluded that DRL agents were able to learn a stable policy for all experiments as the average reward increased during training to a stable return per episode. The agents learned to avoid contamination by smartly sequencing jobs. By comparing the trained agents to the greedy algorithms, the advantage of DRL became clear. The results showed that the DRL agent outperformed the greedy algorithms when uncertainty occurred. The contamination greedy algorithm performed better in the over- and undercapacity cases. Greedy algorithms make local optimum decisions, which are generally good decisions when no uncertainty occurs. The advantage of DRL is that agents anticipate on unexpected events, while greedy algorithms cannot do such things.

Especially the uncertainty in production was anticipated for by the DRL agent as the makespan was relatively low and stable throughout the episodes. The variance in delivered and tardy orders showed that the DRL agent struggled with uncertainty in the distribution. The states and actions design could explain the struggle, as the design limited the DRL agent to deal with truck delays. The order - truck allocation decision was made as soon as the order was stored in the warehouse.

Once that decision was made it could not be changed, even though the departure time was ahead in time. On the other hand, the production scheduling decisions where postponed until a machine became idle, providing ample time to react to uncertainty. The variance between the episodes showed that the DRL agent was more reliable than the greedy algorithms regarding the makespan in the uncertainty and undercapacity experiments. Furthermore, it was more reliable than the contamination greedy algorithm in the undercapacity case in terms of on time delivered orders, although the performance was worse on average.

The comparison between the greedy algorithms showed that including contamination in the decision making is crucial for the production scheduling in this setting. Besides, it can be concluded that a lower makespan and less contamination cleaning usage does not necessarily lead to more on time delivered orders. As there are more decisions that affect the delivery moment of orders, this conclusion makes sense. Moreover, the makespan includes the time between order arrival and actual production. Thus, a lower makespan does not necessarily lead to more throughput.

The case study showed that the DRL agent was able to learn a scheduling policy for the case that approaches the real life setting. The main difference between the case study and other experiments was the later due date of arriving orders. However, the setting did not encourage the DRL agent to learn the contamination rules fast as it gradually decreased over time and remained higher than the contamination cleaning usage of the contamination greedy algorithm. Again, the DRL agent outperformed the greedy algorithms in terms of makespan, while it failed to deliver more orders on time. In addition, the robustness experiments showed that all DRL agents performed good on comparable problems. These findings confirm the expectations as DRL is known to be robust in comparable settings. From this experiment can be concluded that the low uncertainty agent performed best over all, it even outperformed DRL agents on the setting that they were trained for. The low uncertainty agent likely benefited from the uncertainty, making it more robust for other settings.

Contradictory to the online RL agent, the offline RL algorithms could not find a suitable policy to schedule the production and distribution process. There could be several reasons for this behaviour. To start, the datasets could be too small. In these experiments about 250.000 observations were used, while in other studies datasets up to 50 million observations were included (Agarwal et al., 2020). Another option could be that the datasets were not diverse enough. The offline RL agent requires sufficient understanding of the action-state space to learn a policy, solely the high return trajectories are not enough (Schweighofer et al., 2021). To increase the diversity, the transitions dataset from the training phase was used as well. Besides, the datasets included two times the number of episodes it took the online RL agents to learn a policy.

It is more likely that the problem formulation was the reason for the failed training. Existing offline RL algorithms are not suited for multi-discrete actions. Therefore, the multi-discrete action space was transformed to a single discrete action space. Moreover, the decisions in this particular problem are rather complex with a state space composed of multiple components (order, resource and additional information). In other studies, offline RL was often applied to simpler decisions such as image recognition or classification problems (Levine et al., 2020). Last, overfitting is a general problem in offline RL applications. Specific offline RL algorithms such as CQL and BCQ avoid overestimation of the behavior policy, yet they are prone to underestimation of undersampled actions (Levine et al., 2020). Due to the large action space in this case many actions are rarely used, even though they are necessary to adequately schedule the production and distribution process. The overfitting problem becomes even bigger through the transformation from multi-discrete to single-discrete actions.

8 Conclusion & discussion

This section will draw general conclusions and answer the research question that was given in section 7. Furthermore, a discussion about the thesis including limitations and future research directions are given.

8.1 Answer to the research question

The thesis had the objective to find an automated solution for the mismatch in production and distribution scheduling at premix manufacturers. Based on the problem description that was provided by KSE, an early literature review was done. Here, DRL came forward as a promising technique for automating and integrating the production and distribution schedules. Therefore, the following research question was formulated:

How can the production and distribution schedule of premix feed producers be automated and integrated by means of deep reinforcement learning, while accounting for unforeseen events?

To answer that question, an extended literature review was conducted first. The literature review provided insights in existing solutions (both DRL based and others) to comparable scheduling problems. Furthermore, it showed what DRL is and how it can be applied to scheduling problems. Based on this knowledge and the problem description, a solution method was developed. The sparse literature about SMDP in DRL research showed promising results. Therefore, the decision was made to formulate the environment as a SMDP, which requires less decision moments from the DRL agent as compared to traditional MDP's. Furthermore, the action space design was novel for DRL scheduling problems, since multiple discrete actions were included in the model. Besides, the agent made actual decisions instead of choosing between heuristic rules. Consequently, the state space increased as the agent required information about each part of the production and distribution processes where it had to make a decision. To include all necessary information, a novel state space design was made. Due to the SMDP formulation and multi-discrete action space, many actions were invalid during state transitions. Therefore, action masking was applied. The masking guided to agent towards fast policy learning in the different experiments. To solve the given problem, the PPO algorithm was chosen because it outperformed other DRL algorithms in comparable problems. Besides, PPO it computational more efficiency than others and has the ability to handle multi-discrete action spaces with masking. To assess the performance of the DRL agent, two greedy algorithms were developed. The greedy algorithms made decisions based on common scheduling rules in premix manufacturing environments. One greedy algorithm considered contamination in the first action, while the other did not.

The results showed that the DRL agents were able to learn a policy with the novel state, action and reward design, since the reward function stabilized after a training phase. Most importantly, they could learn the contamination rules. The trained agents were tested on their training environment. These experiments showed that the contamination greedy algorithm performed better in the over- and undercapacity cases, although the DRL agent produced more reliable results in the undercapacity experiment. Furthermore, the DRL agent outperformed the greedy algorithms on the low uncertainty cases in all performance metrics. Besides, it produced better results in the production part for the high uncertainty case. The makespan of the DRL agent was significantly lower, although this was not expressed in the number of on time delivered orders. The action space design is a reason for the lack of extra on time delivered orders, since it failed to enable the agent to deal with uncertainty in the distributional part of the process. The case study showed that the agent was able to learn a stable policy in a more complex environment. However, it failed to significantly decrease the number of contamination cleaning throughout the episodes. It is expected that the DRL agent can learn the contamination rules with sufficient training time,

since that happened at the smaller instances. Nevertheless, the DRL agent performed better than the greedy algorithms in the case study regarding the makespan, although more orders were tardy. Finally, the robustness analysis showed that the DRL agents defined a policy which can handle comparable settings. The offline RL experiments showed that these algorithms were not able to find a suitable policy in the current formulation, which is likely due to the multi-discrete action space and the complex actions.

In conclusion, DRL appears to be a robust technique to automate and integrate the production schedule with sequence-dependent setup times, dynamic order arrival and unforeseen events. The DRL agents outperformed the greedy algorithms on all performance metrics when low uncertainty was included. In addition, the DRL agent performed equally well on the high uncertainty case compared to the contamination greedy algorithm in terms of on time delivered orders. However, the DRL agent reduced the makespan with 53.70% in that case, which showed its potential when the distribution part is improved. Improving the distribution part of the DRL framework is an interesting direction for future research. Furthermore, the DRL agent was capable of learning complex production rules such as contamination between product types. However, it should be mentioned that this depended on the setting it operates in. The case study showed that the DRL agent delivers the same number of orders as the contamination greedy algorithm, although more orders were tardy. Again, the makespan of the DRL agent was much lower (58.83%). The thesis served as a proof of concept that DRL can be applied to premix manufacturing environments, yet additional research is required before it can be implemented in real premix factories.

8.2 Contributions

Based on the results, the contributions to the literature and KSE can be given. To start, the study contributes to the literature in the following way:

1. A unique problem is studied. The integrated production and distribution scheduling problem, including sequence-dependent setup times in a dynamic and uncertain environment was never studied before.

2. A novel solution approach is given. A DRL based solution is made, including a multi-discrete action space and action masking. Furthermore, modelling the environment as a SMDP is included. Although these solution components themselves are not new, the combination of SMDP, multi-discrete actions, action masking and DRL is novel.

3. The experiments showed that DRL is especially useful when uncertainty occurs. The agent produced robust results for the production schedule, where it reduced the makespan with 25.63% for the low uncertainty case and 53.70% for the high uncertainty case compared to the best performing benchmark. Furthermore, it delivered 15.13% more orders on time for the low uncertainty case. The high uncertainty case showed that the distribution part left room for improvement, as the lower makespan could not be used to deliver more orders on time.

Furthermore, practical contributions for KSE are:

4. The study shows that DRL is a good technique to integrate and automate the production schedule of premix feed manufacturers when uncertainty occurs. Especially the production part is scheduled better by the DRL agent than benchmark algorithms. The case study shows that the DRL agent reduced the makespan by 58.83% compared to the best performing greedy algorithm. However, this advantage could not be used to deliver more orders on

time, which leaves room for improvement in the distribution part. Nevertheless, DRL is a promising solution technique to use in premix manufacturing environment due to its ability to handle dynamic order arrival and uncertainty. Therefore, it is advised to further investigate the applicability of DRL on this scheduling problem.

8.3 Limitations & future research

As was discussed above, this project found some interesting results. However, it should be noted that there are several limitations of this research. These limitations are elaborated on below.

To start, the decision was made to use a simulation model to train the DRL agent instead of interacting with the real environment. The disadvantage of a simulation model is that certain design choices have to be made. One of them is the level of detail. The decision was deliberately made to keep the simulation model abstract, since detailed aspects of the production and distribution planning do not affect the scheduling decisions. However, they do have an effect on the production and distribution processes. Therefore, the trained agent cannot be immediately used on real-life scheduling situations at premix manufacturers as it was not trained to handle these events. Furthermore, flaws in the simulation model may have been exploited by the agent to its advantage.

Another limitation of the study is that the PPO algorithm (Schulman et al., 2017) was not adapted for the SMDP setting that was used. The PPO algorithm was specifically designed for environments that rely on MDP transitions. As was explained before, time is continuous in SMDP while the time in a MDP is discrete. In particular this property could influence the performance of the DRL agent. Now, the original PPO algorithm was used which assumes a steady time interval in each state. In the simulation model, the time in a state differed and this has an effect on the goodness of a state. After all, remaining longer in a state means that all resources are busy. On the other hand, immediate state change means that the action was invalid. Rummukainen and Nurminen (2019) did account for this property by multiplying a continuous time discount factor with the advantage and found that this led to better results than the PPO algorithm without adaptions. For future research it would therefore be interesting to investigate how an adapted PPO algorithm would perform in combination with a SMDP in this environment.

From the uncertainty experiments could be concluded that the action design is not suitable for dealing with uncertainty in the distribution and is the consequence of lacking rescheduling options. The action space was designed to allocate jobs to resources. When one of the resources became unavailable due to unexpected events, the agent could not reacted by changing previously made decisions. The lack of rescheduling options became clear at the fourth action: allocating orders to trucks. The design choice was made that once an order was stored in the warehouse, the agent should allocate the order to a truck regardless of the truck's departure time. Thus, orders could have been assigned to a truck while the departure time was way ahead, making it more likely that the truck delayed. Postponing the allocation until right before the truck departure would give the DRL agent the ability to react on truck delays, as was done at the production decisions.

Another option would be to change the action design. For instance by enabling the agent to reallocate orders when they are assigned to a truck. Instead of removing the order from the state space, the order could still be visible for the agent. When the agent chooses the action where the order would be reallocated, the action would then be executed. As a result, the agent would have to learn when to reschedule and when to leave the schedule as it is. It is interesting to investigate reallocation in future research as it is currently unknown if a DRL agent would be able to learn that behaviour. Most studies use a dynamic planning design where the actual scheduling is postponed until a machine is idle to provide the agent more flexibility (Luo, 2020), (L. Wang et al., 2021) & (Yang et al., 2021). Although postponing this action would tackle the delay problem, it does

not necessarily lead to the best schedule. As was described by Y. Li et al. (2020), it is always a trade-off between flexibility and certainty. Moreover, the distribution process in the simulation model was fairly basic. The distribution planning was prescheduled and ample truck capacity was assumed. The scope of this research focused on the production process, which was requested by KSE. Furthermore, the scope had to be narrowed down due to the time limits of the thesis. Nevertheless, it is interesting how a DRL agent behaves in a truly integrated production and distribution scheduling environment. Current research is limited in these integrated production and distribution processes. Let alone DRL studies on this topic, making it even more interesting to investigate this setting.

In addition, the way contamination between product types was modelled had to be simplified throughout the project as well. In reality, contamination is tracked on ingredient level and this was used during early stages of experiments to learn by the agent. Unfortunately, the DRL agent could not learn this complex task, even when explicit rewards were given. Besides, modelling the contamination between products on product type level could not be learned by the DRL agent as well in the way it was modelled in the thesis. Therefore, a simplification was made by letting the agent decide which product type to produce next instead of choosing the specific job. Although this led to better results, it constrained the agent in sequencing jobs. After all, the job with the earliest due date that matched the chosen product type was always scheduled now. This is sufficient for the scope of the thesis, but may fall short in reality. Therefore, it is interesting to study how contamination on ingredient level can be integrated in this process in a way that the agent is able to understand the rules. Moreover, the topic is about a bigger issue in DRL research. Up until now, the inability of DRL to deal with complex tasks remains unsolved.

Another limitation of the study is the quality of the benchmark. Ideally, the DRL algorithm's performance was compared to an optimal solution. The experiments in this thesis could not be solved exact due to their np-hardness (Garey et al., 1976). However, comparable studies showed that smaller instances can be solved with a solver (L. Wang et al., 2021), (Ren et al., 2021) & (Hubbs et al., 2020). Comparing the DRL algorithm with the solver in a small experiment would give general insights in the performance of the DRL algorithm in experiments with larger instances. In the current setting, only the relative quality of the DRL algorithm and benchmark algorithms can be determined. For future research, this is an interesting direction to study.

Furthermore, the results showed that the DRL agent outperformed the greedy algorithms when uncertainty was included. On the other hand, the greedy algorithm that accounted for contamination performed better without uncertainty. Therefore, it is interesting to investigate the possibilities to combine their strengths in a combined algorithm. Not all decisions in the production and distribution process require DRL. Besides, combining DRL and other methods decreases the computation and training time of the DRL agent. Last, the DRL agent's policy remains unknown. It would be interesting to know which considerations the agents make and which variables influence the decisions. Research has showed that users do not use algorithms when they do not trust them (Ribeiro et al., 2016). Trust can be gained when decisions are transparent and explainable (Glass et al., 2008) and can be done with explainable reinforcement learning (XRL). There are various XRL methods that will not be discussed in detail here. In future research this is an interesting direction to investigate. Especially, when DRL is applied to real decision making algorithms where it requires trust from planners.

8.4 Recommendations

The study has shown that DRL is a good technique to integrate and automate the production schedule at premix feed manufacturers. The current DRL framework is especially suitable for scheduling the production part, while the distribution part can be improved. Furthermore, it should be mentioned that these conclusions were drawn on experiments were certain assumptions

had to be made. Nevertheless, the fundamental properties of DRL show promising applicability to the scheduling environment of premix manufacturers. Specifically through its ability to deal with dynamic order arrival and uncertainty, it is well suited for scheduling on a daily basis. Let alone, the fact that DRL research is relatively new, thus its full potential is yet to be discovered. Furthermore, it was concluded that DRL can learn a policy that deals with contamination between production jobs. However, the way it was incorporated in the simulation model had to be simplified throughout the study. Therefore, future research is needed before DRL can be applied to real-life premix scheduling situations. Besides, DRL could be combined with heuristic rules to utilize the strengths of both techniques. In conclusion, we recommend to continue with research on DRL based solutions for automating and integrating the production and distribution scheduling at premix feed producers. The study showed that DRL is well suited for the characteristics of this environment. A direction for future research is the expansion of the distribution part of the process. Besides, the contamination part of this study should be extended before it can be used in actual premix factories. In addition, the simulation model requires more detail before it is suitable to serve as a digital copy of premix factories.

References

Agarwal, R., Schuurmans, D. & Norouzi, M. (2020). An Optimistic Perspective on Offline Reinforcement Learning.

Akiba, T., Sano, S., Yanase, T., Ohta, T. & Koyama, M. (2019). Optuna: A Next-generation Hyperparameter Optimization Framework. *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2623–2631. https://doi.org/10.48550/arxiv.1907.10902

Armstrong, S., Leike, J., Orseau, L. & Legg, S. (2020). Pitfalls of learning a reward function online. *IJCAI International Joint Conference on Artificial Intelligence*, *2021-Janua*, 1592–1600. https://doi.org/10.24963/ijcai.2020/221

Atallah, M. J. & Blanton, M. (2009). *Algorithms and Theory of Computation Handbook, Volume 1*. https://doi.org/10.1201/9781584888239

Baker, K. R., Lawler, E. L., Lenstra, J. K. & Rinnooy Kan, A. H. (1983). Preemptive Scheduling of a Single Machine to Minimize Maximum Cost Subject to Release Dates and Precedence Constraints. *https://doi.org/10.1287/opre.31.2.381*, *31*(2), 381–391. https://doi.org/10.1287/OPRE.31.2.381

Baker, K. R. & Scudder, G. D. (1990). Sequencing with Earliness and Tardiness Penalties: A Review. *https://doi.org/10.1287/opre.38.1.22*, *38*(1), 22–36. https://doi.org/10.1287/OPRE.38.1.22

Bellemare, M. G., Dabney, W. & Munos, R. (2017). A distributional perspective on reinforcement learning. *34th International Conference on Machine Learning, ICML 2017*, *1*, 693–711. https://arxiv.org/abs/1707.06887v1

Bergstra, J., Yamins, D. & Cox, D. D. (2013). Making a Science of Model Search: Hyperparameter Optimization in Hundreds of Dimensions for Vision Architectures. *28*.

Bergstra, J., Bardenet, R., Bengio, Y. & Kégl, B. (2011). Algorithms for Hyper-Parameter Optimization.

Boland, N., Gulczynski, D., Jackson, M. P., Savelsbergh, M. W. P. & Tam, M. K. (2011). Improved stockyard management strategies for coal export terminals at Newcastle. *nova.newcastle.edu.au*, 12–16. https://nova.newcastle.edu.au/vital/access/services/Download/uon:12547/ATTACHMENT01

Brammer, J., Lutz, B. & Neumann, D. (2021). Permutation flow shop scheduling with multiple lines and demand plans using reinforcement learning. *European Journal of Operational Research*. https://doi.org/10.1016/J.EJOR.2021.08.007

Cals, B., Zhang, Y., Dijkman, R. & van Dorst, C. (2021). Solving the online batching problem using deep reinforcement learning. *Computers & Industrial Engineering*, *156*, 107221. https://doi.org/10.1016/J.CIE.2021.107221

Chen, W., Xu, Y. & Wu, X. (2017). Deep Reinforcement Learning for Multi-Resource Multi-Machine Job Scheduling. https://arxiv.org/abs/1711.07440v1

Chen, Z.-L. (2009). Integrated Production and Outbound Distribution Scheduling: Review and Extensions. *https://doi.org/10.1287/opre.1080.0688*, *58*(1), 130–148. https://doi.org/10.1287/OPRE.1080.0688

Cormen, T. H., Leiserson, C. E., Rivest, R. L. & Stein, C. (2022). *Introduction to algorithms*. MIT press.

De Moor, B. J., Gijsbrechts, J. & Boute, R. N. (2022). Reward shaping to improve the performance of deep reinforcement learning in perishable inventory management. *European Journal of Operational Research*, *301*(2), 535–545. https://doi.org/10.1016/j.ejor.2021.10.045

Deng, G., Yang, H. & Zhang, S. (2016). An Enhanced Discrete Artificial Bee Colony Algorithm to Minimize the Total Flow Time in Permutation Flow Shop Scheduling with Limited Buffers. *Mathematical Problems in Engineering*, *2016*. https://doi.org/10.1155/2016/7373617

Fairee, S., Khompatraporn, C., Prom-On, S. & Sirinaovakul, B. (2019). Combinatorial artificial bee colony optimization with reinforcement learning updating for travelling salesman problem. *Proceedings of the 16th International Conference on Electrical Engineering/Electronics,*

Computer, Telecommunications and Information Technology, ECTI-CON 2019, 93–96. https://doi.org/10.1109/ECTI-CON47248.2019.8955176

Farahani, A., Genga, L. & Dijkman, R. (2021). Tackling Uncertainty in Online Multimodal Transportation Planning Using Deep Reinforcement Learning. https://doi.org/10.1007/978-3-030-87672-2{_}38

François-lavet, V., Henderson, P., Islam, R., Bellemare, M. G., François-lavet, V., Pineau, J. & Bellemare, M. G. (2018). *An Introduction to Deep Reinforcement Learning. (arXiv:1811.12560v1 [cs.LG]) http://arxiv.org/abs/1811.12560* (Vol. 2). https://doi.org/10.1561/2200000071. Vincent

Fu, J., Co-Reyes, J. D. & Levine, S. (2017). EX2: Exploration with exemplar models for deep reinforcement learning. *Advances in Neural Information Processing Systems*, *2017-Decem*, 2578–2588.

Fujimoto, S., Conti, E., Ghavamzadeh, M. & Pineau, J. (2019). Benchmarking Batch Deep Reinforcement Learning Algorithms. https://doi.org/10.48550/arxiv.1910.01708

Fujimoto, S., Meger, D. & Precup, D. (2018). Off-Policy Deep Reinforcement Learning without Exploration. *36th International Conference on Machine Learning, ICML 2019*, *2019-June*, 3599–3609. https://doi.org/10.48550/arxiv.1812.02900

Gabel, T. & Riedmiller, M. (2008). Adaptive Reactive Job-Shop Scheduling With Reinforcement Learning Agents. *International Journal of Information Technology and Intelligent Computing*, *24*(4), 14–18. http://ml.informatik.uni-freiburg.de/_media/publications/gr07.pdf

Garey, M. R. & Johnson, D. S. (1981). Approximation Algorithms for Bin Packing Problems: A Survey. *Analysis and design of algorithms in combinatorial optimization* (pp. 147–172). https://doi.org/10.1007/978-3-7091-2748-3{_}8

Garey, M. R., Johnson, D. S. & Sethi, R. (1976). COMPLEXITY OF FLOWSHOP AND JOBSHOP SCHEDULING. *Mathematics of Operations Research*, *1*(2), 117–129. https://doi.org/10.1287/moor.1.2.117

Garey, M. R. & Johnson, D. S. (1979). *Computers and intractability : a guide to the theory of NP-completeness*. W.H. Freeman,

Geismar, J. H., Laporte, G., Lei, L. & Sriskandarajah, C. (2008). The integrated production and transportation scheduling problem for a product with a short lifespan. *INFORMS Journal on Computing*, *20*(1), 21–33. https://doi.org/10.1287/ijoc.1060.0208

Ghaleb, M., Zolfagharinia, H. & Taghipour, S. (2020). Real-time production scheduling in the Industry-4.0 context: Addressing uncertainties in job arrivals and machine breakdowns. *Computers & Operations Research*, *123*, 105031. https://doi.org/10.1016/J.COR.2020.105031

Glass, A., McGuinness, D. L. & Wolverton, M. (2008). Toward establishing trust in adaptive agents. *International Conference on Intelligent User Interfaces, Proceedings IUI*, 227–236. https://doi.org/10.1145/1378773.1378804

Gomez, J. C. & Terashima-Marín, H. (2018). Evolutionary hyper-heuristics for tackling bi-objective 2D bin packing problems. *Genetic Programming and Evolvable Machines*, *19*(1-2), 151–181. https://doi.org/10.1007/S10710-017-9301-4/TABLES/10

Gonzalez-Neira, E. M., Montoya-Torres, J. R. & Jimenez, J. F. (2021). A multicriteria simheuristic approach for solving a stochastic permutation flow shop scheduling problem. *Algorithms*, *14*(7). https://doi.org/10.3390/A14070210

Goodfellow, I. (2016). Deep learning. *Nature*, *29*(7553), 1–73. www.deeplearningbook.org%20http://deeplearning.net/

Gordon, G. J. (1996). Stable Fitted Reinforcement Learning. *Advances in Neural Information Processing Systems*, *8*, 1052–1058. citeseer.ist.psu.edu/gordon96stable.html

Gottesman, O., Johansson, F., Komorowski, M., Faisal, A., Sontag, D., Doshi-Velez, F. & Celi, L. A. (2019). Guidelines for reinforcement learning in healthcare. *Nature Medicine 2019 25:1*, *25*(1), 16–18. https://doi.org/10.1038/s41591-018-0310-5

Graham, R. L., Lawler, E. L., Lenstra, J. K. & Kan, A. H. (1979). Optimization and Approximation in Deterministic Sequencing and Scheduling: a Survey. *Annals of Discrete Mathematics*, *5*(100), 287–326. https://doi.org/10.1016/S0167-5060(08)70356-X

Gupta, D. & Maravelias, C. T. (2019). On the design of online production scheduling algorithms. *Computers & Chemical Engineering*, *129*, 106517. https://doi.org/10.1016/J.COMPCHEMENG.2019.106517

Gupta, D. & Maravelias, C. T. (2020). Framework for studying online production scheduling under endogenous uncertainty. *Computers & Chemical Engineering*, *135*, 106670. https://doi.org/10.1016/J.COMPCHEMENG.2019.106670

Hajiaghaei-Keshteli, M., Aminnayeri, M. & Fatemi Ghomi, S. M. (2014). Integrated scheduling of production and rail transportation. *Computers & Industrial Engineering*, *74*(1), 240–256. https://doi.org/10.1016/J.CIE.2014.05.026

Hedtstück, U. (2013). Ereignisorientierte Simulation diskreter Prozesse. Springer Berlin Heidelberg. https://doi.org/10.1007/978-3-642-34871-6{_}5

Henderson, P., Islam, R., Bachman, P., Pineau, J., Precup, D. & Meger, D. (2018). Deep reinforcement learning that matters. *32nd AAAI Conference on Artificial Intelligence, AAAI 2018*, *32*(1), 3207–3214. https://ojs.aaai.org/index.php/AAAI/article/view/11694

Hoon Lee, Y. & Lee, S. (2021). Deep reinforcement learning based scheduling within production plan in semiconductor fabrication. *Expert Systems with Applications*, 116222. https://doi.org/10.1016/J.ESWA.2021.116222

Hou, Y., Fu, Y., Gao, K., Zhang, H. & Sadollah, A. (2022). Modelling and optimization of integrated distributed flow shop scheduling and distribution problems with time windows. *Expert Systems with Applications*, *187*, 115827. https://doi.org/10.1016/j.eswa.2021.115827

Hsu, C. C.-Y., Mendler-Dünner, C. & Hardt, M. (2020). Revisiting Design Choices in Proximal Policy Optimization. https://arxiv.org/abs/2009.10897v1%20http://arxiv.org/abs/2009.10897

Hu, Y., Wang, W., Jia, H., Wang, Y., Chen, Y., Hao, J., Wu, F. & Fan, C. (2020). Learning to utilize shaping rewards: A new approach of reward shaping. *Advances in Neural Information Processing Systems*, *2020-Decem*.

Huang, J., Chang, Q. & Arinez, J. (2020). Deep reinforcement learning based preventive maintenance policy for serial production lines. *Expert Systems with Applications*, *160*, 113701. https://doi.org/10.1016/j.eswa.2020.113701

Huang, S. & Ontañón, S. (2020). A Closer Look at Invalid Action Masking in Policy Gradient Algorithms. https://doi.org/10.48550/arxiv.2006.14171

Hubbs, C. D., Li, C., Sahinidis, N. V., Grossmann, I. E. & Wassick, J. M. (2020). A deep reinforcement learning approach for chemical production scheduling. *Computers & Chemical Engineering*, *141*, 106982. https://doi.org/10.1016/J.COMPCHEMENG.2020.106982

Huo, Y., Leung, J. Y. & Wang, X. (2010). Integrated production and delivery scheduling with disjoint windows. *Discrete Applied Mathematics*, *158*(8), 921–931. https://doi.org/10.1016/J.DAM.2009.12.010

Jackson, J. R. (1957). Simulation research on job shop production. *Naval Research Logistics Quarterly*, *4*(4), 287–295. https://doi.org/10.1002/NAV.3800040404

Kakade, S. (2002). A natural policy gradient. *Advances in Neural Information Processing Systems*, *14*. http://www.gatsby.ucl.ac.uk

Karaoğlan, İ. & Kesen, S. E. (2017). The coordinated production and transportation scheduling problem with a time-sensitive product: a branch-and-cut algorithm. *International Journal of Production Research*, *55*(2), 536–557. https://doi.org/10.1080/00207543.2016.1213916

Kardos, C., Laflamme, C., Gallina, V. & Sihn, W. (2021). Dynamic scheduling in a job-shop production system with reinforcement learning. *Procedia CIRP*, *97*, 104–109. https://doi.org/10.1016/J.PROCIR.2020.05.210

Koenig, S. & Simmons, R. G. (1996). The effect of representation and knowledge on goal-directed exploration with reinforcement-learning algorithms. *Machine Learning*, *22*(1-3), 227–250. https://doi.org/10.1007/BF00114729

Konda, V. R. & Tsitsiklis, J. N. (1999). Actor-Critic Algorithms. *Advances in Neural Information Processing Systems*, *12*.

Kullback, S. & Leibler, R. A. (1951). On Information and Sufficiency. *The Annals of Mathematical Statistics*, *22*(1), 79–86. https://doi.org/10.1214/aoms/1177729694

Kumar, A., Zhou, A., Tucker, G. & Levine, S. (2020). Conservative Q-learning for offline reinforcement learning. *Advances in Neural Information Processing Systems*, *2020-Decem*.

Kundu, O., Dutta, S. & Kumar, S. (2019). Deep-Pack: A Vision-Based 2D Online Bin Packing Algorithm with Deep Reinforcement Learning. *2019 28th IEEE International Conference on Robot and Human Interactive Communication, RO-MAN 2019*. https://doi.org/10.1109/RO-MAN46459.2019.8956393

Lee, K., Zheng, F. & Pinedo, M. L. (2019). Online scheduling of ordered flow shops. *European Journal of Operational Research*, *272*(1), 50–60. https://doi.org/10.1016/J.EJOR.2018.06.008

Lee, S., Cho, Y. & Lee, Y. H. (2020). Injection Mold Production Sustainable Scheduling Using Deep Reinforcement Learning. *Sustainability 2020, Vol. 12, Page 8718*, *12*(20), 8718. https://doi.org/10.3390/SU12208718

Leung, J. Y. & Chen, Z. L. (2013). Integrated production and distribution with fixed delivery departure dates. *Operations Research Letters*, *41*(3), 290–293. https://doi.org/10.1016/J.ORL.2013.02.006

Levine, S., Kumar, A., Tucker, G. & Fu, J. (2020). Offline Reinforcement Learning: Tutorial, Review, and Perspectives on Open Problems. http://arxiv.org/abs/2005.01643

Li, K., Jia, Z. H. & Leung, J. Y. (2015). Integrated production and delivery on parallel batching machines. *European Journal of Operational Research*, *247*(3), 755–763. https://doi.org/10.1016/J.EJOR.2015.06.051

Li, K., Ganesan, V. K. & Sivakumar, A. I. (2006). Scheduling of single stage assembly with air transportation in a consumer electronic supply chain. *Computers and Industrial Engineering*, *51*(2), 264–278. https://doi.org/10.1016/J.CIE.2006.02.007

Li, M. & Lei, D. (2021). An imperialist competitive algorithm with feedback for energy-efficient flexible job shop scheduling with transportation and sequence-dependent setup times. *Engineering Applications of Artificial Intelligence*, *103*, 104307. https://doi.org/10.1016/J.ENGAPPAI.2021.104307

Li, Y., Carabelli, S., Fadda, E., Manerba, D., Tadei, R. & Terzo, O. (2020). Machine learning and optimization for production rescheduling in Industry 4.0. *International Journal of Advanced Manufacturing Technology*, *110*(9-10), 2445–2463. https://doi.org/10.1007/s00170-020-05850-5

Li, Z., Ray, ·., Zhong, Y., Ali, ·., Barenji, V., Liu, ·. J. J., Yu, ·. C. X. & Huang, G. Q. (2021). Bi-objective hybrid flow shop scheduling with common due date. *Operational Research*, *21*, 1153–1178. https://doi.org/10.1007/s12351-019-00470-8

Lin, L.-J. (1992). Self-improving reactive agents based on reinforcement learning, planning and teaching. *Machine Learning*, *8*(3-4), 293–321. https://doi.org/10.1007/bf00992699

Liu, P. & Lu, X. (2014). A best possible on-line algorithm for two-machine flow shop scheduling to minimize makespan. *Computers & Operations Research*, *51*, 251–256. https://doi.org/10.1016/J.COR.2014.06.014

Luo, S. (2020). Dynamic scheduling for flexible job shop with new job insertions by deep reinforcement learning. *Applied Soft Computing*, *91*, 106208. https://doi.org/10.1016/J.ASOC.2020.106208

Luo, S., Zhang, L. & Fan, Y. (2021). Dynamic multi-objective scheduling for flexible job shop by deep reinforcement learning. *Computers & Industrial Engineering*, *159*, 107489. https://doi.org/10.1016/J.CIE.2021.107489

Mao, H., Alizadeh, M., Menache, I. & Kandula, S. (2016). Resource management with deep reinforcement learning. *HotNets 2016 - Proceedings of the 15th ACM Workshop on Hot Topics in Networks*, 50–56. https://doi.org/10.1145/3005745.3005750

Marchesano, M. G., Guizzi, G., Santillo, L. C. & Vespoli, S. (2021). Dynamic Scheduling in a Flow Shop Using Deep Reinforcement Learning. *IFIP Advances in Information and Communication Technology*, *630 IFIP*, 152–160. https://doi.org/10.1007/978-3-030-85874-2{_}16

Marsetiya Utama, D., Setiya Widodo, D., Faisal Ibrahim, M. & Kusuma Dewi, S. (2020). An effective hybrid ant lion algorithm to minimize mean tardiness on permutation flow shop

scheduling problem. *International Journal of Advances in Intelligent Informatics*, *6*(1), 23–35. https://doi.org/10.26555/ijain.v6i1.385

Mirjalili, S. (2015). The Ant Lion Optimizer. *Advances in Engineering Software*, *83*, 80–98. https://doi.org/10.1016/J.ADVENGSOFT.2015.01.010

Mishra, A. & Shrivastava, D. (2018). A TLBO and a Jaya heuristics for permutation flow shop scheduling to minimize the sum of inventory holding and batch delay costs. *Computers & Industrial Engineering*, *124*, 509–522. https://doi.org/10.1016/J.CIE.2018.07.049

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., Petersen, S., Beattie, C., Sadik, A., Antonoglou, I., King, H., Kumaran, D., Wierstra, D., Legg, S. & Hassabis, D. (2015). Human-level control through deep reinforcement learning. *Nature 2015 518:7540*, *518*(7540), 529–533. https://doi.org/10.1038/nature14236

Mnih, V., Puigdomènech Badia, A., Mirza, M., Graves, A., Harley, T., Lillicrap, T. P., Silver, D. & Kavukcuoglu, K. (2016). Asynchronous Methods for Deep Reinforcement Learning. https://proceedings.mlr.press/v48/mniha16.html

Mockus, J., Tiesis, V. & Zilinskas, A. (1978). The application of Bayesian methods for seeking the extremum. *Towards global optimization*, *2*(117-129), 2.

Mortazavi, A., Khamseh, A. A. & Naderi, B. (2015). A novel chaotic imperialist competitive algorithm for production and air transportation scheduling problems. *Neural Computing and Applications*, *26*(7), 1709–1723. https://doi.org/10.1007/S00521-015-1828-9/FIGURES/13

Murphy, S. A. (2005). A generalization error for Q-learning. *Journal of Machine Learning Research*, *6*, 1073–1097.

Nair, A., Gupta, A., Dalal, M. & Levine, S. (2020). AWAC: Accelerating Online Reinforcement Learning with Offline Datasets. http://arxiv.org/abs/2006.09359

Nawaz, M., Enscore, E. E. & Ham, I. (1983). A heuristic algorithm for the m-machine, n-job flow-shop sequencing problem. *Omega*, *11*(1), 91–95. https://doi.org/10.1016/0305-0483(83)90088-9

Ness, M. R. & Walker, S. (1995). Average cost pricing in the compound feed industry. *British Food Journal*, *97*(1), 27–35. https://doi.org/10.1108/00070709510077953

Ng, A. Y., Harada, D. & Russell, S. (1999). Policy invariance under reward transformations : Theory and application to reward shaping. *Sixteenth International Conference on Machine Learning*, *3*, 278–287.

Nielsen, M. (2021). Neural Networks and Deep Learning. *Machine learning*. https://doi.org/10.7551/mitpress/13811.003.0007

Nouiri, M., Bekrar, A. & Trentesaux, D. (2018). Towards Energy Efficient Scheduling and Rescheduling for Dynamic Flexible Job Shop Problem. *IFAC-PapersOnLine*, *51*(11), 1275–1280. https://doi.org/10.1016/J.IFACOL.2018.08.357

Oono, K. & Suzuki, T. (2019). Graph Neural Networks Exponentially Lose Expressive Power for Node Classification. https://doi.org/10.48550/arxiv.1905.10947

Ouchiekh, R., Fri, M., Touil, A. & Echchatbi, A. (2021). Total Weighted Tardiness in the Permutation Flow Shop under Uncertainty. *IFAC-PapersOnLine*, *54*(1), 1174–1180. https://doi.org/10.1016/J.IFACOL.2021.08.138

Ouhaman, A. A., Benjelloun, K., Kenné, J. P. & Najid, N. (2020). The storage space allocation problem in a dry bulk terminal: a heuristic solution. *IFAC-PapersOnLine*, *53*(2), 10822–10827. https://doi.org/10.1016/J.IFACOL.2020.12.2868

Paprocka, I., Krenczyk, D. & Burduk, A. (2021). The method of production scheduling with uncertainties using the ants colony optimisation. *Applied Sciences (Switzerland)*, *11*(1), 1–14. https://doi.org/10.3390/APP11010171

Raghavachari, M. (1988). Scheduling problems with non-regular penalty functions-a review. *Opsearch*, *25*(3), 144–164.

Rahmani, D. & Ramezanian, R. (2016). A stable reactive approach in dynamic flexible flow shop scheduling with unexpected disruptions: A case study. *Computers & Industrial Engineering*, *98*, 360–372. https://doi.org/10.1016/J.CIE.2016.06.018

Ren, J., Ye, C. & Yang, F. (2021). Solving flow-shop scheduling problem with a reinforcement learning algorithm that generalizes the value function with neural network. *Alexandria Engineering Journal*, *60*(3), 2787–2800. https://doi.org/10.1016/J.AEJ.2021.01.030

Ribeiro, M. T., Singh, S. & Guestrin, C. (2016). "Why should i trust you?" Explaining the predictions of any classifier. *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, *13-17-Augu*, 1135–1144. https://doi.org/10.1145/2939672.2939778

Rifai, A. P., Mara, S. T. W. & Sudiarso, A. (2021). Multi-objective distributed reentrant permutation flow shop scheduling with sequence-dependent setup time. *Expert Systems with Applications*, *183*, 115339. https://doi.org/10.1016/J.ESWA.2021.115339

Rijsdijk, J., Wu, L., Perin, G. & Picek, S. (2021). Reinforcement Learning for Hyperparameter Tuning in Deep Learning-based Side-channel Analysis. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, *2021*(3), 677–707. https://doi.org/10.46586/TCHES.V2021.I3.677-707

Robinson, S. (2008). Simulation: the practice of model development and use. *Journal of Simulation*, *2*(1), 67–67. https://doi.org/10.1057/palgrave.jos.4250031

Rossit, D. A., Tohmé, F. & Frutos, M. (2018). The Non-Permutation Flow-Shop scheduling problem: A literature review. *Omega*, *77*, 143–153. https://doi.org/10.1016/J.OMEGA.2017.05.010

Rummukainen, H. & Nurminen, J. K. (2019). Practical Reinforcement Learning -Experiences in Lot Scheduling Application. *IFAC-PapersOnLine*, *52*(13), 1415–1420. https://doi.org/10.1016/J.IFACOL.2019.11.397

Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., Huang, Z., Karpathy, A., Khosla, A., Bernstein, M., Berg, A. C. & Fei-Fei, L. (2015). ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision*, *115*(3), 211–252. https://doi.org/10.1007/S11263-015-0816-Y/FIGURES/16

Saito, Y., Yaginuma, S., Nishino, Y., Sakata, H. & Nakata, K. (2020). Unbiased recommender learning from missing-not-at-random implicit feedback. *WSDM 2020 - Proceedings of the 13th International Conference on Web Search and Data Mining*, 501–509. https://doi.org/10.1145/3336191.3371783

Schaul, T., Horgan, D., Gregor, K. & Silver, D. (2015). Universal value function approximators. *32nd International Conference on Machine Learning, ICML 2015*, *2*, 1312–1320. https://proceedings.mlr.press/v37/schaul15.html

Schulman, J., Levine, S., Moritz, P., Jordan, M. & Abbeel, P. (2015). Trust region policy optimization. *32nd International Conference on Machine Learning, ICML 2015*, *3*, 1889–1897. https://proceedings.mlr.press/v37/schulman15.html

Schulman, J., Wolski, F., Dhariwal, P., Radford, A. & Klimov, O. (2017). Proximal Policy Optimization Algorithms. https://arxiv.org/abs/1707.06347v2%20http://arxiv.org/abs/1707.06347

Schweighofer, K., Hofmarcher, M., Dinu, M.-C., Renz, P., Bitto-Nemling, A., Patil, V. & Hochreiter, S. (2021). Understanding the Effects of Dataset Characteristics on Offline Reinforcement Learning. *arXiv preprint arXiv:2111.04714*.

Sharma, S. (2020). ACTIVATION FUNCTIONS IN NEURAL NETWORKS. *International Journal of Engineering Applied Sciences and Technology*, *04*(12), 310–316. https://doi.org/10.33564/ijeast.2020.v04i12.054

Snoek, J., Larochelle, H. & Adams, R. P. (2012). Practical Bayesian Optimization of Machine Learning Algorithms. *Advances in Neural Information Processing Systems*, *4*, 2951–2959. https://arxiv.org/abs/1206.2944v2

Solina, V. & Mirabelli, G. (2021). Integrated production-distribution scheduling with energy considerations for efficient food supply chains. *Procedia Computer Science*, *180*, 797–806. https://doi.org/10.1016/J.PROCS.2021.01.355

Stecke, K. E. & Zhao, X. (2007). Production and transportation integration for a make-to-order manufacturing company with a commit-to-delivery business mode. *Manufacturing and Service Operations Management*, *9*(2), 206–224. https://doi.org/10.1287/msom.1060.0138

Sutton, R. S., Precup, D. & Singh, S. (1999). Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning. *Artificial Intelligence*, *112*(1), 181–211. https://doi.org/10.1016/S0004-3702(99)00052-1

Sutton, R. & Barto, A. (1998). *Introduction to reinforcement learning*. https://login.cs.utexas.edu/sites/default/files/legacy_files/research/documents/1%20intro%20up%20to%20RL%3ATD.pdf

Tang, C. Y., Liu, C. H., Chen, W. K. & You, S. D. (2020). Implementing action mask in proximal policy optimization (PPO) algorithm. *ICT Express*, *6*(3), 200–203. https://doi.org/10.1016/j.icte.2020.05.003

Tangour, F., Nouiri, M. & Abbou, R. (2021). Multi-objective production scheduling of perishable products in agri-food industry. *Applied Sciences (Switzerland)*, *11*(15). https://doi.org/10.3390/APP11156962

Toso, E. A., Morabito, R. & Clark, A. R. (2009). Lot sizing and sequencing optimisation at an animal-feed plant. *Computers & Industrial Engineering*, *57*(3), 813–821. https://doi.org/10.1016/J.CIE.2009.02.011

Van Hasselt, H., Guez, A. & Silver, D. (2015). Deep Reinforcement Learning with Double Q-learning. *30th AAAI Conference on Artificial Intelligence, AAAI 2016*, 2094–2100. https://arxiv.org/abs/1509.06461v3

Wang, H., Yan, Q. & Zhang, S. (2021). Integrated scheduling and flexible maintenance in deteriorating multi-state single machine system using a reinforcement learning approach. *Advanced Engineering Informatics*, *49*, 101339. https://doi.org/10.1016/J.AEI.2021.101339

Wang, L., Hu, X., Wang, Y., Xu, S., Ma, S., Yang, K., Liu, Z. & Wang, W. (2021). Dynamic job-shop scheduling in smart manufacturing using deep reinforcement learning. *Computer Networks*, *190*, 107969. https://doi.org/10.1016/J.COMNET.2021.107969

Wang, L., He, X., Zhang, W. & Zha, H. (2018). Supervised reinforcement learning with recurrent neural network for dynamic treatment recommendation. *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, *18*, 2447–2456. https://doi.org/10.1145/3219819.3219961

Wang, Q., Batta, R. & Szczerba, R. J. (2005). Sequencing the processing of incoming mail to match an outbound truck delivery schedule. *Computers & Operations Research*, *32*(7), 1777–1791. https://doi.org/10.1016/J.COR.2003.11.029

Wang, Z., Zhang, J. & Yang, S. (2019). An improved particle swarm optimization algorithm for dynamic job shop scheduling problems with random job arrivals. *Swarm and Evolutionary Computation*, *51*, 100594. https://doi.org/10.1016/J.SWEVO.2019.100594

Wang, Z., Schaul, T., Hessel, M., Van Hasselt, H., Lanctot, M. & De Frcitas, N. (2016). Dueling Network Architectures for Deep Reinforcement Learning. *33rd International Conference on Machine Learning, ICML 2016*, *4*, 2939–2947. https://proceedings.mlr.press/v48/wangf16.html

Waschneck, B., Reichstaller, A., Belzner, L., Altenmüller, T., Bauernhansl, T., Knapp, A. & Kyek, A. (2018). Optimization of global production scheduling with deep reinforcement learning. *Procedia CIRP*, *72*, 1264–1269. https://doi.org/10.1016/j.procir.2018.03.212

Watkins, C. J. C. H. & Dayan, P. (1992). Q-learning. *Machine Learning 1992 8:3*, *8*(3), 279–292. https://doi.org/10.1007/BF00992698

Williams, R. J. (1992). Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, *8*(3-4), 229–256. https://doi.org/10.1007/bf00992696

Wu, Y.-C. & Rasmussen, C. E. (2019). TAM- Using trainable-action-mask to improve sample-efficiency in reinforcement learning for dialogue systems. (NeurIPS).

Yağmur, E. & Kesen, S. E. (2020). A memetic algorithm for joint production and distribution scheduling with due dates. *Computers & Industrial Engineering*, *142*, 106342. https://doi.org/10.1016/J.CIE.2020.106342

Yang, S., Xu, Z. & Wang, J. (2021). Intelligent Decision-Making of Scheduling for Dynamic Permutation Flowshop via Deep Reinforcement Learning. *Sensors 2021, Vol. 21, Page 1019*, *21*(3), 1019. https://doi.org/10.3390/S21031019

Yu, J. J., Yu, W. & Gu, J. (2019). Online Vehicle Routing with Neural Combinatorial Optimization and Deep Reinforcement Learning. *IEEE Transactions on Intelligent Transportation Systems*, *20*(10), 3806–3817. https://doi.org/10.1109/TITS.2019.2909109

Zahari, M. W. & Alimon, A. R. (2005). Use of palm kernel cake and oil palm by-products in compound feed. *Palm oil developments*, *40*, 5–8.

Zahavy, T., Haroush, M., Merlis, N., Mankowitz, D. J. & Mannor, S. (2018). Learn what not to learn: Action elimination with deep reinforcement learning. *Advances in Neural Information Processing Systems*, *2018-Decem*, 3562–3573.

Zandieh, M., Fatemi Ghomi, S. M. & Moattar Husseini, S. M. (2006). An immune algorithm approach to hybrid flow shops scheduling with sequence-dependent setup times. *Applied Mathematics and Computation*, *180*(1), 111–127. https://doi.org/10.1016/J.AMC.2005.11.136

Zhang, B., Rajan, R., Pineda, L., Lambert, N., Biedenkapp, A., Chua, K., Hutter, F. & Calandra, R. (2021). On the Importance of Hyperparameter Optimization for Model-based Reinforcement Learning. https://proceedings.mlr.press/v130/zhang21n.html

Zhang, X. & Van De Velde, S. (2010). On-line two-machine job shop scheduling with time lags. *Information Processing Letters*, *110*(12-13), 510–513. https://doi.org/10.1016/J.IPL.2010.04.002

Zheng, Q.-Q., Zhang, Y., Tian, H.-W. & He, L.-J. (2021). An effective hybrid meta-heuristic for flexible flow shop scheduling with limited buffers and step-deteriorating jobs. *Engineering Applications of Artificial Intelligence*, *106*, 104503. https://doi.org/10.1016/j.engappai.2021.104503

Zhou, L., Zhang, L. & Horn, B. K. (2020). Deep reinforcement learning-based dynamic scheduling in smart manufacturing. *Procedia CIRP*, *93*, 383–388. https://doi.org/10.1016/J.PROCIR.2020.05.163