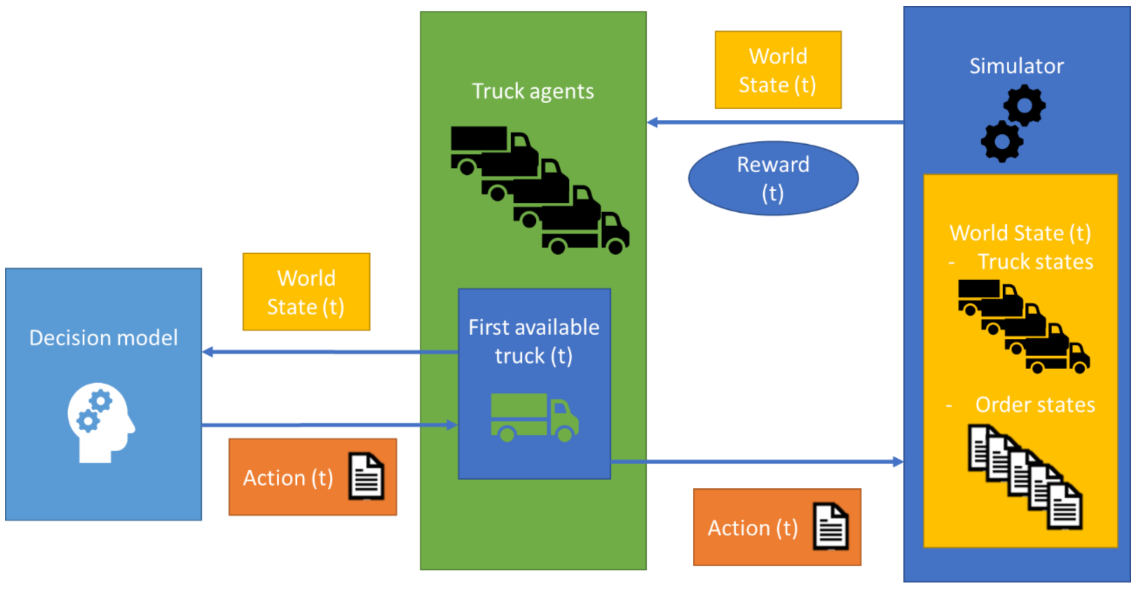
Reinforcement learning for order distribution in self-organizing logistics



Yorick C. de Vries

Reinforcement learning for order distribution in self-organizing logistics

by Yorick C. de Vries

To obtain the degree of Master of Science in Computer Science at Delft University of Technology, to be defended publicly on Tuesday August 31, 2021 at 11:00.

Student number: 4636317

Thesis duration: October 2020 – August 2021

Thesis committee:

Dr. Neil Yorke-Smith	Algorithmics, TU Delft, supervisor
Dr. Emir Demirović	Algorithmics, TU Delft, supervisor
Dr. Luís Miranda da Cruz	Software Engineering, TU Delft
Dr. Wendelin Böhmer	Algorithmics, TU Delft
Dr. Bilge Atasoy	Transport Engineering and Logistics, TU Delft
Christian van Ommeren	TNO, supervisor
Dr. Jan-Pieter Paardekooper	TNO, supervisor

An electronic version of this thesis is available at https://repository.tudelft.nl

Abstract

With the increasing global demand for logistics, supply chains have grown a lot in volume over the last decades. To be able to operate effectively within the capacity constraints of the carriers, proper collaboration and optimization of order allocation is required. Van Berkel Logistics facilitates the transport of containers by trucks from sea terminals in Rotterdam to inland customers and back. This logistical planning problem is manually solved by planners on a daily basis. Within this research it is investigated to what extent reinforcement learning could be applied for solving this planning problem of moving containers in an automated way. A simulation environment was constructed which represents the container planning dynamics. It was made as accurate as reasonably possible with the help of historic data. Three reinforcement learning models, the OnePass, Iterative and Attention model, have been developed and tested for their ability to learn to choose proper orders such that the orders are as much on time as possible. A main challenge in constructing these models was to design them such that they could cope with a varying state and action space. In an experimental evaluation, it was found that the models are able to learn to make better decisions over time and eventually perform similar to the heuristic baseline tested out in terms of total lateness observed. In terms of driven distance and fraction on time orders, the OnePass and Iterative model were able to beat the heuristic choices. Overall, the Iterative model has shown the best performance and is able to learn scenarios as big as real-life scenarios van Berkel Logistics deals with. However, it also tends to be slower than the other models due to its iterative approach.

Preface

In front of you lies my thesis on reinforcement learning for order distribution in self-organizing logistics. The aim of this thesis is to provide a step into the direction of a future where logistical supply chains are decentrally managed with the use of self-learning agents. Although such scenario seems quite futuristic, I hope to give insight into how reinforcement learning could play a role in making this possible.

This thesis was conducted at the Algorithmics department at Delft University of Technology as final part of my Master Computer Science program. Between October 2020 and August 2021, I have worked at the Sustainable Transport and Logistics department of TNO in The Hague. Unfortunately, due to the COVID pandemic, it was not feasible to work very often in the office in the Hague. However, I feel that the people at TNO were very welcoming and facilitated a good internship experience despite of the restrictions. I feel my experience at TNO gave me a good view on how research can be applied to real life cases and what the challenges herein are.

This thesis is not solely made by myself as many people have aided me along the way. Firstly, I would like to thank Neil and Emir for being my university supervisors. In our meetings, you gave me good academic feedback and triggered me to think critically of my work. Secondly, I would like to thank Chris and Jan-Pieter for my supervision within TNO. Chris helped me a lot in converting the use case of van Berkel Logistics to a simulation environment as realistically as possible. Jan-Pieter has aided a lot in connecting the reinforcement learning techniques to this environment. In addition, I would like to thank Wendelin for the input and help in implementing the Attention based reinforcement learning model. Lastly, I would like to thank all the other people within TNO who I have met throughout my internship. It was very pleasant to be part of TNO and I look back on it as very valuable experience.

I am grateful for all the good time I have had at the Delft University of Technology and am proud to have this thesis as concluding work. I wish you a pleasant read.

Yorick de Vries

Schiedam, August 2021

Contents

Chapter 1. Introduction

With the increasing global demand for logistics, supply chains have grown in volume over the last decades [1], leading to more costs and congestion on the roads [2]. To be able to operate effectively within the capacity constraints of the carriers, proper collaboration and optimization of order allocation is required. On a global level, there is much potential to improve logistics from both a cost and environmental view [3]. Research has shown that a significant CO_2 emission reduction can be achieved with such improvements [4]. With efficient coordination within a company, 5% of the costs can be saved [5]. When different stakeholders collaborate together, this cost reduction can increase to 14% [5]. For such optimization, organizing supply chains by a central planner makes intuitively most sense, however in practice this is hard to achieve when considering there are many parties in a logistics network. In reality several smaller supply chain partners that each manage their own information and needs are involved. Integrating these needs between those partners in a central manner raises many concerns (in terms of security, commercial benefit, privacy etc.) and does therefore rarely happen in practice [6].

As alternative to having several central planners for optimizing logistics, one can look at a fully decentralized approach where not these central planners, but individual vehicles themselves (i.e. trucks and ships) decide on the distribution of orders [6]. In this self-organizing logistics approach, the vehicles cooperate with each other and make decisions on who is doing what in order to efficiently transport goods. Information sharing and negotiation between vehicles can happen on a local level and no central planner needs to be contacted for making decisions. In the short-term, trucks can be planned quicker and in a more automated way as there is less reliance on a central planner. In the long term, this approach could result in a more efficient and robust supply chain as cooperation of logistics between supply chain partners can be tackled this way. Furthermore, in case of disruptions, self-reorganization through local decision making can occur directly between vehicles instead of relying on a central planner. Local decision making tends to be quicker as no decisions for the whole system need to be taken [4].

TNO, an independent research organization in the Netherlands, is interested in innovating supply chains to make them more robust and sustainable in the future. It has performed a trial applying self-organizing logistics in a simulation environment to be able to organize the supply chain of container distribution between the port of Rotterdam to inland customers and back. Within this trial, trucks autonomously negotiate with each other on which trucks will transport which containers [6]. This simulator uses data of van Berkel Logistics[1] to dynamically generate orders and the trucks negotiate with each other about task division with a distributed constraint optimization problem (DCOP) solver approach [7]. Negotiation occurs via predefined rules to optimize the global cost function based on the capacity utilization of the truck fleet. If an order cannot be executed on time it is not planned.

Such an approach defines how trucks should interact with each other and is therefore limited to what the programmer has considered while designing it. However, it might not necessarily yield the desired results in the real world, especially when situations occur that are not taken into account beforehand.

[1] http://www.vanberkellogistics.eu

Furthermore, such approaches are less universally applicable due to specific constraints of different supply chains they are designed for.

Instead of using an approach to predefine rules on how the trucks should negotiate on orders, one could also think of how the trucks themselves can learn to make good decisions. Within this thesis, machine learning has been applied with the aim to facilitate a more widely applicable approach for solving self-organizing logistics. Machine learning can be roughly divided up in supervised learning, unsupervised learning and reinforcement learning. A supply chain cannot be simply converted into a classification problem needed for classical machine learning. This is because the environment wherein trucks make decisions is complex and the effect of making certain choices is not directly apparent and influence the outcome of subsequent decisions as well. The goal is to use the fleet of trucks as effectively as possible such that all the orders are as much on time as possible by the end of the day. Due to this nature of the planning problem, reinforcement learning is deemed most suitable among the different machine learning variants to tackle this problem and is therefore the scope of the thesis.

Reinforcement learning

Reinforcement learning can be applied to agents that interact with an environment which enables them to learn to maximize long term rewards [8]. This environment is often considered a Markov Decision Process as the state the environment is in, and the step taken from that state solely indicates the next state the environment will be in. Agents can explore and exploit this environment in sequential timesteps based on (positive or negative) rewards they expect to get for their actions. In this way they can learn by optimizing their policy such that the (discounted) value of every action they take in the states of the environment they find themselves in is high.

For a long time, reinforcement learning could only be used for environments with a small state space. Recently however, progress was made by using function approximation to estimate quality (Q) values from a state through deep learning [9]. Via a Deep Q Network (DQN), Mnih et al. [9] were able to generalize states to Q values for the different possible actions. This approach has proven to be successful as it was able to learn how to win several Atari games without prior knowledge and is now the de-facto standard for reinforcement learning. After DQN, there have been several extensions which have shown success in several specific tasks [10]. Chapter 2 will discuss reinforcement learning and existing applications within logistical problems more extensively.

Problem statement

For the implementation of reinforcement learning for optimizing a supply chain, one can see every truck as an agent that interacts with the environment by executing orders. The main goal is to let the trucks make decisions such that all orders are delivered as timely as possible at the end of the day. This long-term reward setting makes reinforcement learning a logical approach to tackle this problem. Furthermore, both the agents and environment are clearly defined in this setting and sequential decisions can be made.

For the use case of van Berkel Logistics, the environment looks as follows: There is a fleet of trucks present which need to be assigned to orders to execute. Orders are container movements and, depending on the order, cannot necessarily be done by all trucks. A terminal truck for example, can only do orders which are in vicinity of its inland container terminal, while a container which needs to be picked up from a port terminal in Rotterdam can only be done by a truck which has the proper emission category. In addition, trucks need to have an empty chassis with them when they are picking up a container from a sea terminal in the port of Rotterdam. Furthermore, when containers need to be transported to inland customers, there are customer specific demands as well. For some customers, the container can be decoupled on location and a truck is directly available again. For other customers the container needs to be loaded or unloaded on location and the container needs to be brought back to the inland container terminal before the truck can start another order. All these constraints together make efficient planning of the orders a challenge.

This thesis aims to use reinforcement learning to construct a self-learning model wherein trucks can make proper choices in which orders to do. We want to prevent the trucks from taking orders greedily which leads to short term benefits that might not be beneficial to the whole system at the end of the day. Instead, we want the agents to focus on the global reward; the timeliness of all orders together at the end of the day (i.e. minimizing the total lateness of all orders). Furthermore, we can run the simulator indefinitely and are not directly limited by the size of training data like in classical machine learning problems as we can basically generate a new, unique problem instance for every trial. Therefore, these aspects make reinforcement learning a suitable method to apply. However, we should be aware to what degree the simulator and the data it uses represent actual real-world scenarios. We cannot account for everything that can happen in real life and therefore need to make some simplifications in the simulation environment (see also Chapter 3). This may limit the effectiveness of the method when applied in the real world.

Research question

The objective of this thesis is to develop a reinforcement learning model for trucks which is able to learn by itself to efficiently choose orders for the inland container transport use-case of van Berkel Logistics. This model can be used to evaluate the feasibility of using reinforcement learning for logistical use cases. To be able to do this, the following main research question is formulated:

Main research question

To what extent is reinforcement learning advantageous for order distribution in self-organizing logistics?

Several sub-questions have been defined to answer this question. For every question it is discussed how this thesis answers this question as well.

Subquestion 1

How should the environment be designed to be compatible with reinforcement learning approaches?

Reinforcement learning works on environments where sequential decisions are made. The environment in our case is a simulator which contains information of present orders and trucks. Orders have properties like an origin, destination, and a time window. Trucks have properties like location and (expected) time they are available. This environment plays out a day in which orders need to be transported and trucks can interact with this environment by executing these orders throughout the day. More precisely, whenever a truck is available, it can choose an order to do and the simulation environment executes that order for the truck. Trucks are available at irregular timesteps and orders have variable durations as well. The environment therefore becomes a Semi Markov Decision Process (Semi-MDP) as the decision moments are happening at irregular intervals.

Subquestion 2

Which reinforcement learning architectures are most suitable?

Reinforcement learning is typically dealing with a fixed action space. For example; in Atari games the actions that an agent can do is a limited set of button presses. These actions can be trivially mapped to separate outputs in a Q network to predict their value. Similarly, in the boardgame Go there is a limited feature space as well as a fixed space wherein the playing stones can be placed.

Translating this approach to our setting is not trivial though. The actions that can be done by trucks at a certain moment are represented by an arbitrary list of orders. The number of orders and their characteristics is highly different between decision moments and are not trivially mapped in a neural network.

Three different networks were designed to overcome this problem: namely the OnePass, Iterative and Attention networks. The OnePass network evaluates many orders as input at the same time and outputs Q values for the orders corresponding to the input, this is most similar to classical Q networks. The Iterative network model evaluates one order at the time in respect of the other available orders and has therefore only one output which is interpreted as Q value. The Attention network works on sets of elements and enables message passing between the elements. The elements in our case are trucks and orders and the output for the order elements can be interpreted as Q values.

It was found that the Iterative network is well able to learn the environment even when a large number of orders and trucks are present. The OnePass network also able to learn the environment, but takes longer to converge though. The OnePass model is not able to learn large instances of the problem though. The Attention network is able to learn the environment to a lesser extent and is not as stable in convergence. However, when the problem is scaled up to real life scenarios, it is found that the Iterative and OnePass model do not scale along that well. However, the Attention network scales better due to the ability to process sets of elements of any size.

Subquestion 3

In what way can the environmental state best be presented for the agents?

The environment is characterized by the orders that must be done and trucks that are active in the environment. Orders that already have been done earlier are not relevant for decision making anymore and are not represented in the state space.

Trucks are defined by the time they are available to do a next order and their location at that time. In addition, they have a label for the type of truck. Orders are represented by the locations where from/to a container needs to be shipped as well as the duration of doing that order. An important property is the time window when the order can be done, represented by an earliest start time as well as a deadline. Furthermore, the type of order (matching with the types of truck that can do that order) is represented as well. Lastly, all locations and timestamps are calculated relatively to the truck choosing an order at that point. This has the advantage that the location and time of the current truck do not need to be represented.

Subquestion 4

How should we define the reward function?

The reward function is a key element of reinforcement learning as it describes what to optimize in the long term. We have discussed this with planners of van Berkel Logistics and they indicated that it is of importance to focus on minimizing the lateness of orders as much as possible. Other factors like driving distance are of less importance and are also indirectly taken into account as more efficient routing of trucks leads to less lateness as well.

A straight-forward way to return lateness from the environment is by calculating it once the order is done. However, this makes the penalty of not doing an urgent order less explicit. We can also already calculate the number of minutes that all existing orders are late until a certain timepoint and return that to the trucks as reward. In this way the trucks get an easier grasp on the effect of not choosing an order which runs late. It is found that this approach in defining the reward function helped in convergence speed on the reinforcement learning methods.

Subquestion 5

How does the performance of a reinforcement learning approach compare to other approaches?

The three reinforcement learning models, OnePass, Iterative and Attention, have been evaluated in a variety of problem instances. The performance of the reinforcement learning approaches were compared to the performance of several heuristic approaches. The heuristic of choosing orders with the most urgent deadline deemed the most powerful heuristic to minimize lateness and is often also a main decision driver for human planners at van Berkel Logistics. For smaller instances, the reinforcement learning models were able to perform similarly to this heuristic in terms of lateness. However, the reinforcement learning models have shown that it performs better in terms of total distance driven and fraction of orders that are done on time. For larger instances, the reinforcement learning models are able to learn and make better decisions than random choice and therefore show their potential, however they are not able to beat the above-mentioned heuristic just yet.

Contributions

Within this thesis, it is shown that order distribution can be learnt with the help of reinforcement learning. For solving the logistical truck planning problem of van Berkel Logistics with reinforcement learning, the problem needed first to be incorporated into an environment where truck agents can interact with. The first main contribution was to develop such a simulation environment wherein trucks can sequentially choose orders. For this environment, historic data from van Berkel Logistics may be used as well as synthetic generated data. This environment plays out a day wherein trucks can sequentially make decisions with the goal to minimize the total lateness at the end of the day as much as possible. In addition, a visualization of the environment is made so that the behaviour of the decision models can be evaluated visually.

Three reinforcement learning architectures are proposed within this thesis. For each model different values for hyperparameters (like hidden representation, optimizer etc.) have been tested to properly choose which ones to use in the final models. The three models have been evaluated for their performance on several scenarios. The problem instance size was varied from smaller instances with only 13 orders and 2 trucks (Scenario 1) to real life size scenarios of 255 orders and 37 trucks (Scenario 4). In addition, both the use of synthetic data and real data plus some relaxations have been tested. Relaxations include whether trucks need to wait for an order to start or only need to take deadlines into account, or whether orders are always routed via a central inland terminal or not.

This extensive evaluation has given insight in the strengths and limitations of these models. While designing these models, the key challenge in designing these approaches was how to deal with a state- and action space which varies in size between decision moments. Although we mainly focused on the planning problem proposed by van Berkel Logistics, the proposed models can be extended to other vehicle routing problems as well. In addition, the insights for tackling a variable state and action space can be translated over to other problems.

Outline

The remainder of this thesis will be as follows: Chapter 2 will discuss background needed concerning reinforcement learning and to what extent it is already applied within logistical problems. Chapter 3 will focus on the properties of the use case of van Berkel Logistics we are trying to solve. The raw data used will be described and it will be explained how this data is used for extracting orders. Information on how constructing the simulation environment is done is explained in Chapter 4. In Chapter 5 the three reinforcement learning architectures are explained, as well as the feature representation the architectures use for their decision making. Chapter 6 will cover the experimental setup and explain the instances used for evaluation as well as the key performance indicators (KPIs) used. Chapter 7 will explain the results obtained by the experimental setup and their implications. Lastly, Chapter 8 contains concluding remarks and directions for future research.

Chapter 2. Background

Before the work in this thesis will be explained, some background needs to be understood first. Firstly, a short introduction is given into reinforcement learning and what problems it is able to solve. We go into what Q learning is and how it is extended with neural networks to deep Q learning. In addition, attention networks will be introduced as such networks are used within this work to make a new type of deep Q network. Lastly, we will look into existing applications of reinforcement learning within the domain of logistics.

Reinforcement learning

Reinforcement learning is, together with supervised and unsupervised learning, a type of machine learning [8]. Reinforcement learning aims to make optimal choices when interacting with an environment and it learns based on feedback it receives. This approach has been successful in learning Atari games [9] and the board game Go [11].

Markov Decision Process

A Markov Decision Process (MDP, Figure 1) is a mathematical formulation of an environment with which interaction takes place [12]. When a problem can be formulated as a MDP, reinforcement learning can be applied to it.

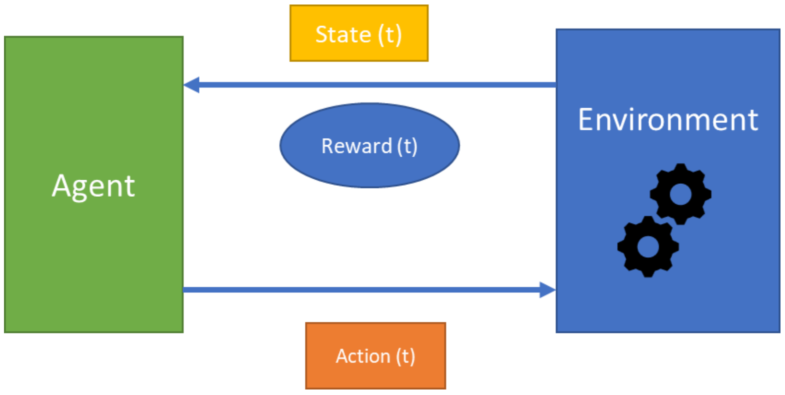


Figure 1 Representation of a Markov Decision Process (MDP). The agent observes the environment at time t and takes an action. In response to that action, the environment changes and a reward is received from the environment.

The environment has a certain state at time t: s_t. In the example of Atari games, this state can be the pixels visible on the screen at a certain frame. This state represents the world at that time and an agent can use this information to choose an action (a) at that moment (t). When an agent performs an action (like pressing a button in Atari games) in this state, the environment changes and a new state is observed. The new state can either be stochastic or deterministic and is determined by a transition function. In the stochastic case, the next state is determined by a chance distribution based on the previous state and the performed action. In the deterministic case, the next state from a previous state and action will always be the same.

This illustrates the Markov Property which is important for reinforcement learning; the next state is only dependent on the observed state + action that is taken. This enables us to frame the problem concisely for the design of learning mechanisms.

In addition to the next state, there is also a reward obtained after doing an action in the environment (R_t). This is the feedback the MDP gives for doing certain actions in certain states. In Atari games this reward can be as simple as the scored points. The goal for an agent is to maximize the total reward in all encountered states. It is therefore not only important for an agent to quickly get a good reward for their actions, but also that an agent goes into next states wherein the chances of additional rewards is high.

A Semi-Markov Decision Process (Semi-MDP) [13] is a small extension of a MDP. In Semi-MDPs actions do not take place in a fixed-length timestep, but in timesteps of varying lengths. The next time a decision can be made can be seen as a stochastic part of the environment.

The mechanism an agent uses to make its choices for actions in an environment is referred to as the policy of an agent. The policy is a function which takes a state as input and outputs an action to do. A policy can be as simple as a heuristic as well as an advanced choice mechanism which learns while interacting with the environment. Within reinforcement learning, the goal is to find a policy such that the total received reward over time is maximized.

Q learning

Q learning is a version of reinforcement learning which aims to predict the quality (Q) of doing certain actions (a) in certain states (s_t) [14], [15]. This quality is referred to as Q values and, when these can properly be estimated, an agent can base its behaviour on these. It can perform actions based on the best available action (action with the highest known Q value) available in a state. The Q value is formally defined as stated in Equation 1. This Q value is recursive and therefore dependent on the rewards (R) in all future states. We would like to make a preference for rewards encountered in the short term over more uncertain rewards encountered in the long run. For this a discount factor (gamma, γ) between 0 and 1 is often used which helps in prioritizing rewards. When γ is low, short term rewards are more important for the Q value than long term rewards. The is the discount factor has typically a value of around 0.99.

Equation 1 Q value function.

$$Q(s_t, a) = R_t + \gamma * {}^{max}_{a'} Q(s_{t+1}, a')$$

In addition, we can define the value of a state to be equal to the Q value when taking the best action (Equation 2).

Equation 2 State value function.

$$V(s_t) = {}^{max}_{a} Q(s_t, a)$$

These two equations can be combined into Equation 3 and this equation is often referred to as the Bellman equation.

Equation 3 Bellman equation.

$$Q(s_t, a) = R_t + \gamma * V(s_{t+1})$$

Whenever the environment arrives in a final state (s_{t+1}) after doing an action in state s_t, the Q value of the previous state (s_t) is solely determined by the observed reward R_t.

Learning form experiences

When an agent interacts with an environment, it obtains experiences in the form of (s_t, a_t, R_t, s_{t+1}) as shown in Figure 1. These experiences can be used to learn from, as agents can iteratively improve their estimates of expected Q values. Q value estimates can be saved in a table and iteratively updated based on learning rate α. The update formula for Q learning is shown in Equation 4.

Equation 4 Q value update function. α is the learning rate.

$$Q(s_t, a) \leftarrow Q(s_t, a) + \alpha * (R_t + \gamma * \max_{a'} Q(s_{t+1}, a') - Q(s_t, a))$$

The experiences are typically obtained via ε-greedy exploration. The agent chooses with a chance of ε a random action (exploration) and otherwise the best-known action according to the known Q values (exploitation). The value of ε is gradually degraded so more exploration is done when the learning process is just started, while more exploitation is done towards the end of learning.

Deep Q learning

The approach of doing Q learning via a tabular approach and iteratively updating Q values in memory works well for small environments with a finite state and action space. However, in environments where the state space is not small anymore, the tabular approach will not be feasible. For such complex environments, we need to generalize via function approximation.

A technique that has been shown to be very powerful for doing this is using deep neural networks for Q value estimation [9]. So called Deep Q Networks (DQNs) have been shown to generalize states well and were able to learn several Atari games purely based on interaction with the environment and obtaining rewards [9]. Such DQN takes the state of the environment as input (pixels in case of Atari games) and outputs a Q value for every possible action (button presses).

These networks can be trained with the update function in Equation 4. Experiences are saved in a replay buffer and from this replay buffer random experiences are sampled to train the network. However, it is found that such training tends to be unstable when the network is used both for updating Q values as well as for estimating the Q values it needs to update towards. For this reason, a static target network is used for next Q value estimation instead of the same policy network which is being updated. This target network will be updated every so many episodes with the weights of the policy network. This tweak with a separate target network enables a much more stable training.

Double Deep Q Learning

Double Deep Q Learning (DDQN) is a slight extension of DQN with a variation on how the best action is determined [16]. Vanilla DQN tends to overestimate the Q values due to always taking the maximum output of the target network. In DDQN, not always the highest value put out by the target network is used to train on. Instead, the action where the highest Q value is observed in the policy network is checked at the target network, which might be different and lower than the maximum output of the target network.

Attention networks

Attention is a neural network type which is gaining popularity in recent times. Vaswani et al. [17] proposed the transformer network, which is a sequence to sequence neural network with added attention mechanisms. The transformer network is able to achieve high performance in machine translation tasks. The advantage of attention neural networks is that they are able to process sets of elements of arbitrary size and are not limited by fixed number of input features normal neural networks do.

Within the transformer encoder layer of the transformer attention network the input elements can exchange information via messages. The network learns how to construct messages from every input element and these are passed to all other elements. Per element, the importance of these messages is learnt and based hereon, these messages are incorporated in a new feature vector. This is where the Attention model has gotten its name from as the network learns where to put attention to. Multiple transformer encoder layers can be used in succession to allow for advanced information exchange between elements. The aim within this thesis is to use these attention mechanisms to deal with varying input sizes of our problem.

Related work

The travelling salesman problem (TSP) and its extension to multiple vehicles, the vehicle routing problem (VRP) are typically formulated as a constraint problems and can be solved with combinatorial optimisation via integer programming [18]. An extension of VRP with time windows where travel time between nodes has some uncertainty due to delays is analysed by Agra et al. [19]. With the use of integer programming, the authors were successfully able to solve such instances. This method was further improved by Hoogeboom et al. with the use of a branch-and-cut approach for their integer programming approach [20].

In contrast to solving these problems via constraint programming, there have been several works of applying reinforcement learning to routing and logistical planning problems. Nazari et al. [21] were able to solve VRP instances to near optimum with use of reinforcement learning. The authors ran into the problem that their initial state space was not invariant, as two inputs could be swapped and would lead to a different representation for the same environmental state. This was solved by the authors with the use of an attention network. Kool et al. [22] have applied reinforcement learning in combination with attention networks as well to learn heuristics for solving TSP. Their implementation has been extended to VRP where multiple vehicles need to optimize their route in terms of distance [23]. Additionally, it was found that for solving TSP with deep reinforcement learning, convolutional neural networks combined with recurrent neural networks were found to be useful [24] and this approach has shown to generalize well to arbitrary instances. The same authors have also looked into how to facilitate a proper exploration strategy. They have achieved this by adding entropy to the received reward, encouraging exploration for solving vehicle routing problems [25].

When we step away from solving more theoretical TSP and VRP instances, there have been some real-life applications of reinforcement learning in the domain of logistics. It is shown that large-scale fleet management can be achieved through multi-agent reinforcement learning [26]–[28]. Lin et al. [26] were able to manage relocation of taxis over a city based on the current order demand via reinforcement learning. A common averaged reward for all vehicles on a certain location was used to circumvent greedy behaviour of the taxis. Both an actor-critic as well as a DQN approach work well in this setting. This approach has been further extended to facilitate order dispatching for the taxis. Each taxi determines its advantage of taking an order via reinforcement learning and a central matching algorithm determines thereafter the actual order-to-vehicle assignment [27]. This is extended even further by Tang et al. [28] by using a neural network for approximating the advantages for individual vehicles. A similar approach has also been followed by Sultana et al. [29]. In their approach, vehicles indicate in a decentral way via reinforcement learning which customers they would like to serve. Thereafter, a central allocation heuristic is again used to make the actual allocations. Their DQN is constructed such that they iteratively go over all vehicle-customer pairs to estimate preferences. To our knowledge these are the only approaches wherein reinforcement learning has been applied successfully to let vehicles determine which orders they want to do.

Apart from using reinforcement learning in logistics for vehicle routing, another application has been proposed in the field of logistics as well. With international sea freight transport, containers are continuously loaded and unloaded in ports in different parts of the world. Certain sea ports tend to perform more exporting of goods than importing or vice versa. For this reason, an imbalance of containers can arise and to make sure enough empty containers are available at every port to be loaded, occasionally empty containers need to be moved from one sea terminal to another. Li et al. [30] have applied multi agent reinforcement learning to tackle the container repositioning task by formulating it as a stochastic game. At every sea port on their route, ships were able to determine whether to pick up or drop off empty containers to make sure there are enough empty containers available.

Chapter 3. Problem definition

Van Berkel Logistics is in charge of transporting containers between sea terminals and inland locations [31]. This transport happens via one of the inland terminals managed by van Berkel Logistics, located in Veghel, Cuijk and, more recently, Oss. Within this research we limit the scope to the terminals in Veghel and Cuijk. A general view of how containers are moved by van Berkel Logistics is depicted in Figure 2.

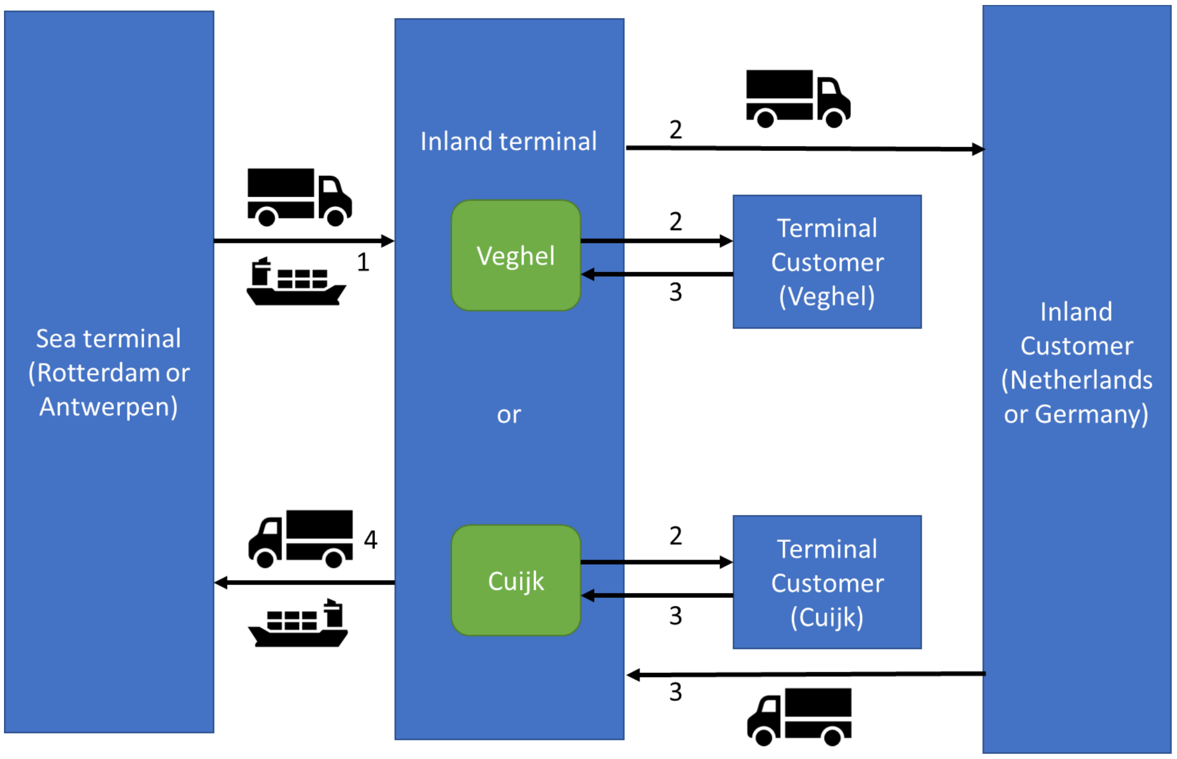

Sea terminal (Rotterdam or Antwerpen)
1
4
Inland terminal
Veghel
or
Cuijk
2
2
3
2
3
3
Terminal Customer (Veghel)
Terminal Customer (Cuijk)
Inland Customer (Netherlands or Germany)

Figure 2 The four different types of container movements (orders 1-4). A container typically moves from a sea terminal via one of the inland terminals to the inland customer and back.

Container movements

From a sea terminal, typically located in Rotterdam or Antwerpen, containers are picked up and transported to one of the inland terminals in Veghel or Cuijk. This transport typically happens by barge, but can also happen via trucks. From Veghel and Cuijk containers are transported to inland customers, typically located in eastern Netherlands or just over the border in Germany. At the inland customer the container is emptied or filled depending on whether it is used for importing or exporting goods. Hereafter the container is picked up again by a truck of van Berkel Logistics and brought back to one of the inland terminals. From there it is again transported to a sea terminal by barge or truck.

Summarizing, a container has typically four movements as depicted in Figure 2. These individual movements will be further referenced to as individual orders which need to be planned. Every order is restricted to a time window in which it should be done and the proper planning of these orders is the scope of this thesis.

Transport from sea terminal

While transport between the sea terminal can happen either via barge or truck, we limit the scope of this research to truck transport. When a container needs to be picked up from a sea terminal, the container is craned to the truck and the truck therefore needs to arrive with an empty chassis to be able to transport the container. Vice versa happens when a container is dropped off at a sea terminal; the container is craned off a truck and the truck is left with an empty chassis. Furthermore, when a sea terminal is located on the Maasvlakte in Rotterdam, the truck motor needs to adhere to Euronorm 6[2], meaning that it needs to be low enough in its emission.

Inland transport

Transport of containers from the inland terminal to inland customers occurs solely via trucks. There is a distinction between customers located closely to the terminals in Veghel or Cuijk and customers which are further away located from the inland terminals. Container transport to these closely located terminal addresses is typically done by so called terminal trucks. When a container needs to be transported to an address further away from the terminal, it cannot be transported by a terminal truck anymore and needs to be transported by a region or port truck.

When a container is transported to an inland customer, the container can either be decoupled or live-loaded. When a container is decoupled, the container, including the chassis it stands on, is left at the customer address and will be picked up another time. Hence, decoupling is typically done on locations which are frequently visited by van Berkel Logistics (so called key-customers). On locations which are less frequently visited, live-loading is preferred. When a container is live-loaded, the truck waits at the customer for a container to be emptied (or filled in case it is an export container). It drives thereafter back to an inland terminal to drop off the container again.

Trucks

Van Berkel Logistics has a fleet of trucks which are used for container transport. There are different types of trucks and the orders a truck can do is dependent on this type. Both inland terminals Veghel and Cuijk have terminal trucks which are only allowed to drive in close proximity of the respective terminal. Region trucks are allowed to drive outside the terminal and can visit all locations as long as there are no special emission restrictions required. This means region trucks can go to inland customers as well as sea terminals in the port of Antwerpen, as there are no emission restrictions in that port. When a truck needs to visit a sea terminal location located on the Maasvlakte in Rotterdam, it needs to be low in emission. Such trucks are further referenced to as port trucks. Lastly, when a truck planner has too many orders it needs to do, charter trucks can be rent from an external party. These trucks are typically Euronorm 6 trucks and can visit port addresses as well. In Table 1 the truck-address compatibility is summarized.

[2] https://www.rotterdam.nl/werken-leren/verkeer-maasvlakte

Table 1 Truck type - address type compatibility.

	Veghel terminal address	Cuijk terminal address	Region address	Port address
Veghel terminal truck	X	-	-	-
Cuijk terminal truck	-	X	-	-
Region truck	X	X	X	
Port truck	X	X	X	X
Charter truck	X	X	X	X

Based on these addresses the orders can be categorized as well, this is done in Table 2. When the order and destination are on the same terminal, the order type will be of that terminal. And when a truck needs to visit a port as one of the addresses, it is a port order. Otherwise, it is considered a region order.

Table 2 Order types based on origin and destination address types.

Origin/Destination	Veghel terminal	Cuijk terminal	Region	Port
Veghel terminal	Veghel terminal	Region	Region	Port
Cuijk terminal	Region	Cuijk terminal	Region	Port
Region	Region	Region	Region	Port
Port	Port	Port	Port	Port

Central coordination by a planner

Currently the decision which trucks do which orders at which moment is determined by central planners at van Berkel Logistics [31]. This planning is done manually per day in Excel sheets which are then communicated to the truck drivers.

Planners typically start their planning by first scheduling orders which need to go to sea terminal and back. These orders tend to have quite some urgency because they are dependent on international sea transport. Rather often these orders can only be done by port trucks and are therefore more restrictive while planning. The planner tries to combine the pickup and drop off of containers on sea terminals as otherwise trucks need to drive a lot of distance unloaded. Hereafter region orders are planned. These orders are quite restrictive as there is an agreed time the orders need to arrive at the customer. Lastly, we have the terminal orders. These are planned last as these orders take less time to do due to their proximity to the inland terminal. In addition, most trucks are compatible to do these orders. When the planner encounters that not enough trucks are available to plan the orders, external charter trucks can be rent.

At the start of the day the trucks still tend to stick to their original planning. The planning is not set in stone however; during the day the trucks tend to deviate from their schedule for various reasons like traffic jams, deviating processing times at the customer, etc. For this reason, the planner adjusts the truck schedules on the fly. Furthermore, when trucks are available earlier than planned, they can do extra orders which were not scheduled yet at the start of the day.

While planning, the planner is mostly focused on scheduling the orders as much on time as possible. There are agreed times when containers need to arrive at the customer and it is of most importance not to deviate from these agreed times. This is also indirectly achieved by efficient use of the truck fleet. The planner tries to minimize the waiting time at customers (times when trucks are too early) and the distance trucks drive without moving a container.

Simplifications made

In reality the truck planning has more factors which need to be taken into account and for the scope of this thesis some simplifications needed to be made. A list of these simplifications which are not taken into account within this thesis is shown below;

- When a container is emptied at a customer, it can theoretically directly be moved to another customer instead of an inland terminal.
- The truck fleet is fixed at the start of the day and no charter trucks can dynamically be rent.
- Overhead processing time differs per customer and planners know these times by heart.
- Special types of containers might need special treatment. This could for example be containers with refrigerated goods or hazardous materials.
- When driving from and to a sea terminal, several containers can be moved by one truck at the same time. Whether this is possible is dependent on the weight and size of the containers.
- There is a higher priority of getting export containers on time as international transport ships depart on fixed moments.
- Trucks can drive containers directly from a customer to a sea terminal or back without stopping at an inland terminal.
- In between stops for situations like customs declaration or gas measurement is not taken into account.
- Truck drivers have specific working times and need to take mandatory breaks according to Dutch law[3].

Historic data

To be able to make a proper simulation environment which represents the truck planning problem, data of van Berkel is extracted and analysed [31]. Van Berkel Logistics uses the software of Modality[4] to keep track of their container logistics. This data only contains information on what historically happened and does not contain any information on what the original planning was. Planning data is made per day in separate excel sheets and are not parsable in an automated matter.

Modality database exports for containers handled in 2016 up to 2019 are used to extract single orders (single container movements) for the simulation environment. Every row in this file depicts all information of a container; from the moment it first arrives at a sea terminal to be picked up till the

[3] https://www.rijksoverheid.nl/onderwerpen/werktijden/vraag-en-antwoord/rijtijden-en-rusttijden-wegvervoer
[4] https://www.modality.nl

moment it is dropped off again at a sea terminal after being handled by an inland customer. Per year, on average approximately 57.000 containers are handled by van Berkel Logistics. As these events per container generally span over several days, or even weeks, the information in the row of one container is split up into separate container orders as shown in Figure 2. Not every row contains a full journey of a container from the sea terminal to inland and back. This can for example happen when an empty container at one customer is directly brought to another customer to be filled, yielding multiple rows for one container. In such cases less than 4 orders will be extracted from the data.

Sea terminal orders

Transport from/to a sea terminal (orders 1 and 4) is typically done by barge, however it happens as well that a truck needs to transport a container from/to a sea terminal. As we are interested in truck logistics for this thesis, only orders which have historically been done by trucks are extracted from the data. In the Modality export there is some logging information present on when orders were done. However, the order log data is not always present or complete. For these orders there is an earliest time and deadline when the truck needs to be at the sea terminal. Sometimes only dates were present as moments the trucks should be on location and no specific times were present in the data. For such orders, 00:00 (earliest time) and 23:59 (latest time) were used as time window.

Inland customer orders

For orders 2 and 3 there is a single timestamp present when the truck needs to be at the customer. And as we only have this single timestamp, no time windows can be extracted. After discussion with van Berkel Logistics, it was determined that 30 minutes before and after the agreed timestamp would still be acceptable, yielding a time window of 1 hour wherein the order can be done.

Orders 2 can be both to decoupling and live-loading customer addresses. In case the address type is a live-loading address, the truck needs to directly bring back the live-loaded container to the inland terminal as part of the order. For this reason, orders 3 are only extracted from the data in case the inland customer address is not a live-loading address and the container needs to be moved back later. In some cases, when only a date (no timestamp) was present for inland addresses, it was assumed that the container could be transported to the location all day (from 00:00 till 23:59).

Time windows

From the data we can obtain either the moment the container needs to be picked up at the origin (orders 1 and 3) or the moment when it needs to be dropped off at the destination (orders 2 and 4). For simplicity, only the moment orders need to start are extracted or calculated. For orders 1 and 3 the original timestamps are taken over for earliest time and deadline. For orders 2 and 4, the destination times are used to calculate the time the orders need to be started on the origin location. The timestamps on the destination are shifted to an earlier time based on when the truck needs to depart from the origin address in order to arrive on time.

Trucks

Van Berkel Logistics has a permanent fleet of trucks that is available to use for container transport. This fleet is provided in a separate Excel file and is used to match the trucks present in the log files to be able to get insight into the general fleet composition including their types and main location. This main location is the location the trucks start and end their day.

Addresses

For all customers the addresses are known based on the street, postal code, city and country of the customer. Latitude and longitude coordinates of these customers are not present in the export and added via the use of the Google Maps API[5]. The addresses were filtered for inaccuracies like unknown or very far locations (like locations in Spain). The coordinates of the inland terminals in Veghel and Cuijk are shown in Table 3.

Table 3 Coordinates of the two inland terminal locations of van Berkel Logistics.

	Latitude	Longitude
Terminal Veghel	51.6184911	5.5086791
Terminal Cuijk	51.7570228	5.8460356

Address type

For inland locations, info on whether addresses can be visited by Cuijk or Veghel terminal trucks could be extracted from the exports. Whether a location is a decoupling or live-loading location is not readily present for all addresses though. For the key customers, i.e. customers which are frequently visited by van Berkel Logistics, this information is available and for the majority of these customers decoupling is performed.

For non-key customers the service type cannot be extracted from the data. For these locations live-loading is therefore assumed. This makes sense as non-key customers are less visited throughout the year and van Berkel Logistics prefers to directly take containers back to the inland terminal instead of having to pick up these containers another time.

Port and/or Craning addresses

Craning is almost exclusively done at the sea terminals. Import containers are coming from a ship and therefore trucks need to bring an empty chassis when picking up a container from a sea terminal. Vice versa happens when a container is dropped off at the sea terminal: the truck ends up with an empty chassis and can either pick up another sea terminal container or bring back the chassis to the inland terminal.

Whether addresses are port and/or craning orders cannot easily be determined from the addresses export. Instead, it is based on the historic container data described previously. The addresses which are found in the columns of orders 1 and 4 are considered as sea terminals with craning locations. In addition, in case the address is located near Rotterdam Maasvlakte, it is denoted as a port location and can only be visited by a port truck which has the appropriate Euronorm. Otherwise, the address is marked as a region address and can therefore be visited by both region and port trucks. Addresses in the port of Antwerpen can for example be visited by region trucks as there the emission rules are not as

[5] https://developers.google.com/maps

strict as on Maasvlakte Rotterdam. A visual plot of the extracted addresses, including their types are depicted in Figure 3.

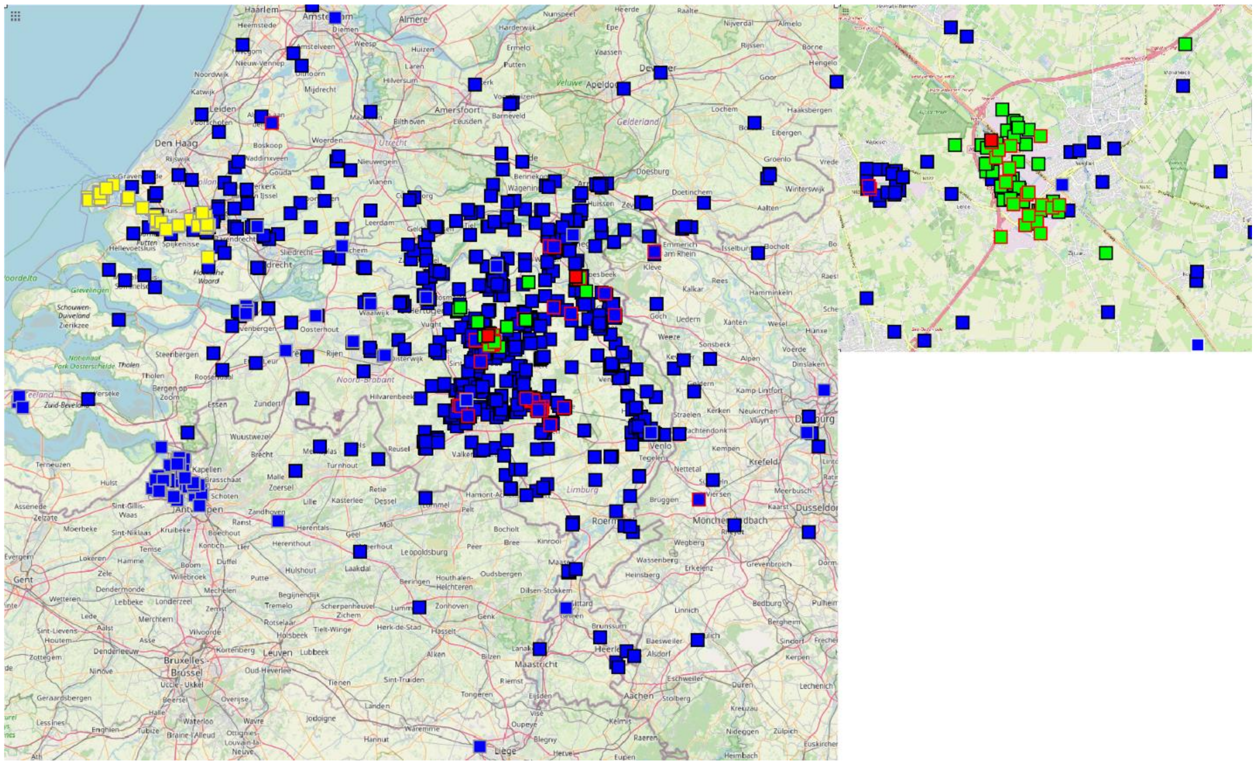


Figure 3 Addresses extracted from the exports. Inside colour represents the address types: inland terminal (red), terminal customer address (green), region address (blue) and port address (yellow). The outline represents the location type: decoupling (red), live-loading (black) and craning (grey). On the right a close-up of the terminal of Veghel is shown.

Driving times

The driving times between addresses is based on the measured distance between their coordinates. The distance between locations is calculated as the haversine distance between the coordinates multiplied by a factor of 1.2. When calculating the time it takes to drive this distance, different speeds are used for different parts of the route. The idea behind this is that one can more easily achieve a higher speed when driving a long distance due to the availability of more highways. When traversing only a few kilometres, lower average speeds will be achieved. The speed and duration calculation are shown in Table 4 and Figure 4. This approach has been discussed with van Berkel Logistics and deemed a realistic approximation.

Table 4 Speed calculation.

Kilometres (from-to)	Speed (km/h)
0 – 10	20
10 – 35	50
35 +	70

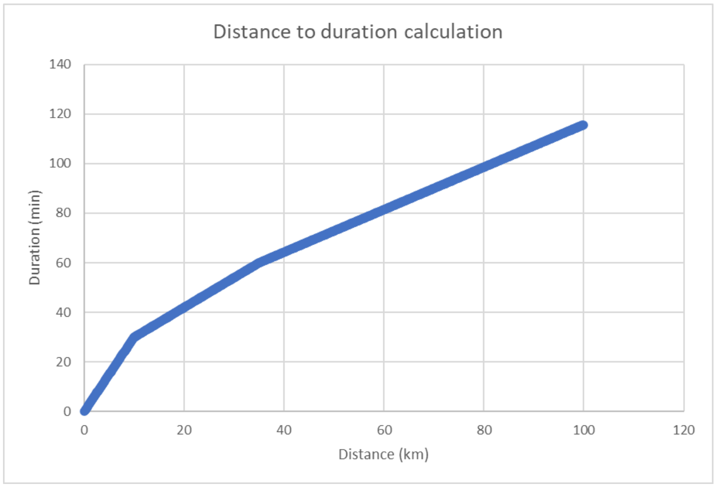


Figure 4 Drive duration calculation for increasing distances.

Leg list

Via an app, truck drivers indicate when they arrive and depart from a customer address. This data is provided in a leg list file and can be used to estimate the time it takes to process an order at a customer. A histogram of these stop durations is plotted in Figure 5. A high number of stops of only 0 and 1 minute was observed. These durations likely occurred due to truck drivers forgetting to indicate they arrived at a customer and log both arrival and departure at the same time. These durations are therefore excluded from the graph. Based on this data, an overhead time of 30 minutes is chosen while parsing the orders.

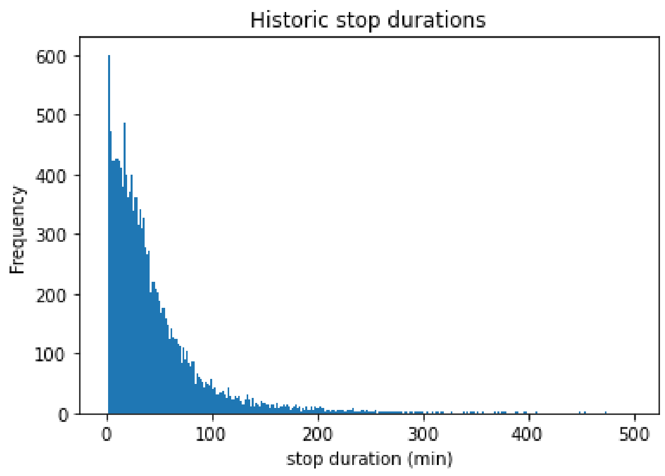


Figure 5 Historic stop duration extracted from the leg list.

Data insights

The number of extracted orders for the different years is shown in Table 5. The majority of the orders are from inland terminal to inland customer. Many of these orders are live-loading orders, meaning that the truck will bring back the container to the inland terminal as part of the order to be finished. For this reason, there are less orders from inland customer back to the inland terminal (order 3) as these only cover locations where the container is picked up later after decoupling.

Table 5 Numbers of orders per year and type.

	Order 1	Order 2	Order 3	Order 4	Total
From	Sea terminal	Inland terminal	Inland customer	Inland terminal	
To	Inland terminal	Inland customer	Inland terminal	Sea terminal	
2016	2579	37948	13508	2295	56330
2017	2518	42683	16670	2589	64460
2018	3071	41255	17725	2521	64572
2019	3209	45372	19551	3006	71138
Total	11377	167258	67454	10411	256500

Number of orders and trucks per day

The parsed orders are split over different days based on their deadline. Whenever a deadline before 05:00 is found, it is considered to be for the day before. In Figure 6 the orders are plotted for four separate days. Everyday there are quite a few orders which happen solely on the terminal of Veghel. Furthermore, the region orders are scattered around the terminals, most of these are in the Netherlands, but occasionally also a container needs to be moved to or from Germany. Every day also a few port orders need to be done and these are located on the Maasvlakte in Rotterdam.

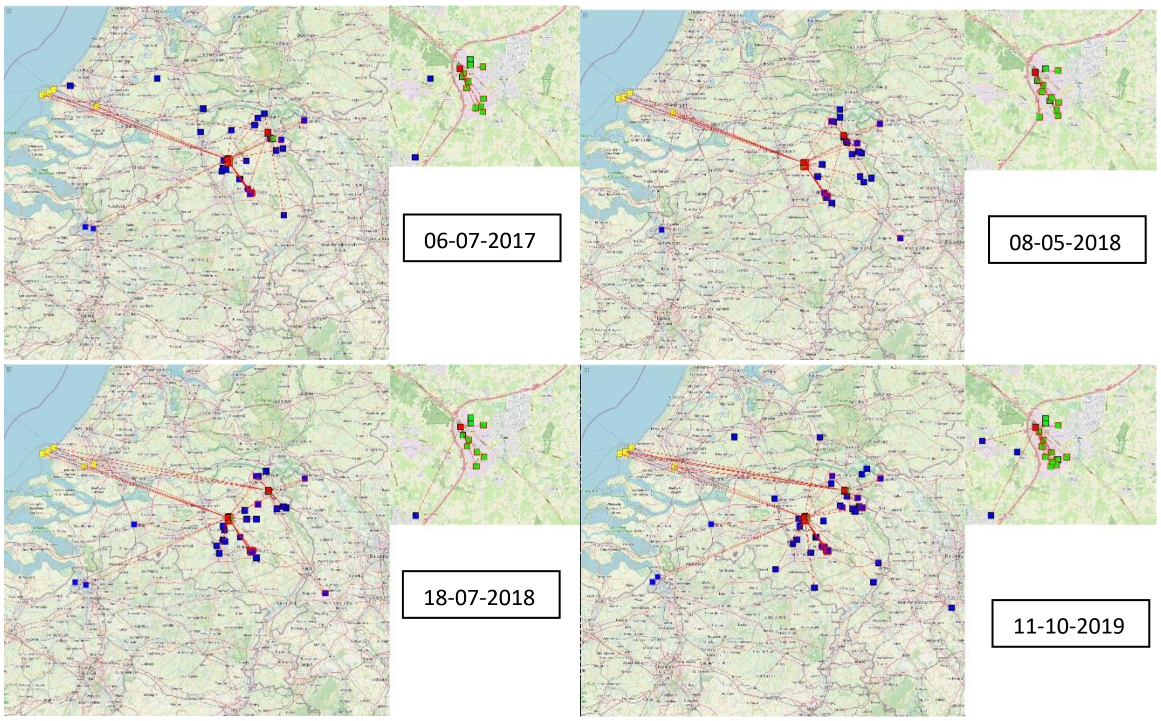


Figure 6 Orders for 4 different days. The red lines indicate the path over which the containers need to be moved. Inside colour represents the address types: inland terminal (red), terminal customer address (green), region address (blue) and port address (yellow). The outline represents the location type: decoupling (red), live-loading (black) and craning (grey).

For all the days in 2016 till 2019 the number of orders is counted per type and plotted in Figure 7. The same is done for trucks and their types in Figure 8. These figures give insight in how an average day for container planning by van Berkel Logistics looks like and will be used to design instances to test out later in a simulation environment. As all days, also the holidays and weekends, are included in the analysis, peaks around 0 are generally observed.

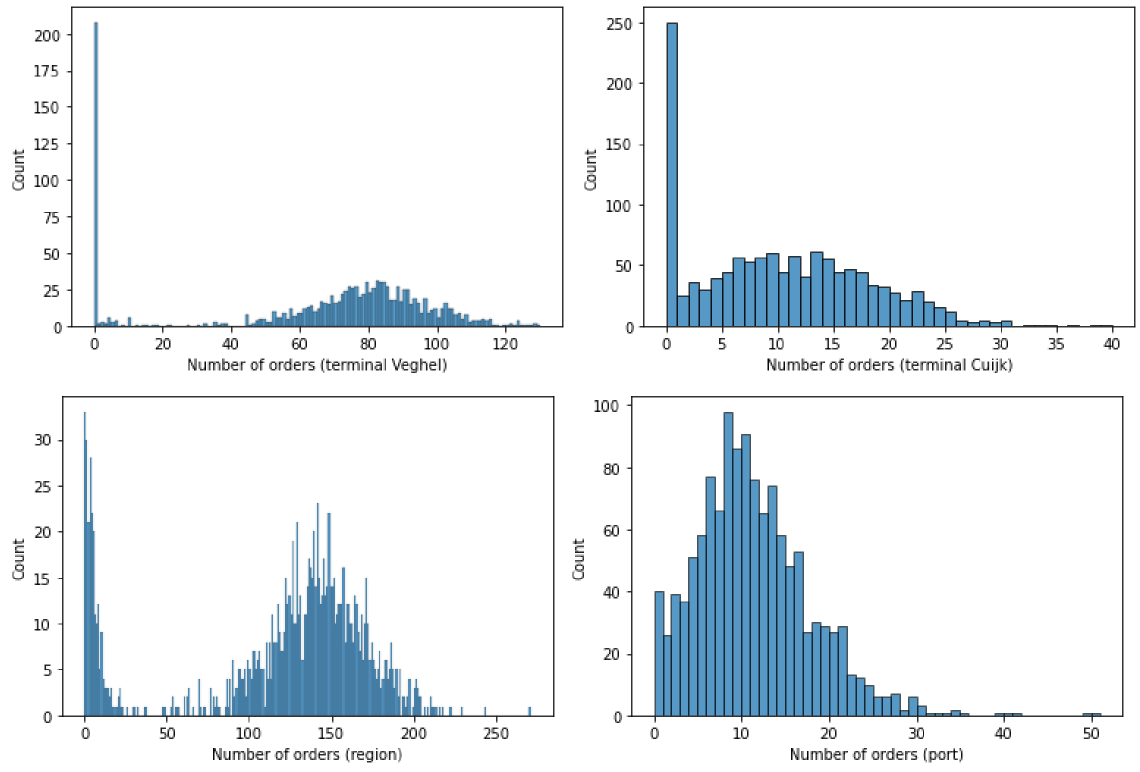


Figure 7 Histograms of number of orders on a day per order type.

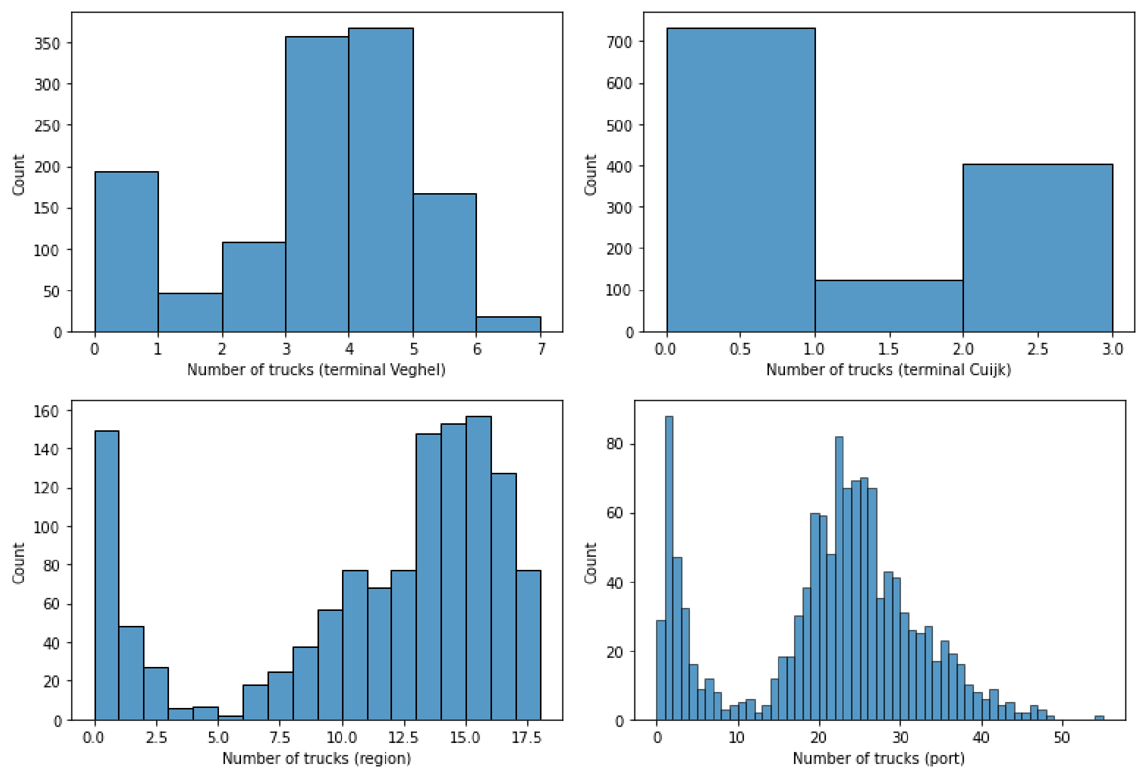


Figure 8 Histograms of number of trucks on a day per truck type.

Order times analysis

An overview of the times of the orders are depicted in Figure 9, Figure 10 and Figure 11 for order lengths, time window lengths and deadlines respectively. Order lengths vary between 30 minutes to 3 hours. The high peak around 30-40 minutes is due to many orders with a short driving distance. For these orders the overhead time of 30 minutes is mainly determining the order lengths. The time windows wherein orders should be done are often 1 hour as explained earlier. In addition, there are also orders present which can be done all day (from 00:00 till 23:59, which equals 1440 minutes). The deadlines extracted from the data show a peak around 10:00. Furthermore, there are also many orders which have their deadline at the end of the day (around 24 hours).

Next steps

In this chapter, the container planning problem of van Berkel Logistics is described as well as the simplifications that have been made to make the problem manageable for this thesis. In the following chapter, Chapter 4, it will be described how this problem is converted into a simulation environment where can be interacted with. For this simulation environment, the data insights above will be used to make the environment more accurate. In Chapter 5, descriptions are given of the reinforcement learning models which learn to optimize choices made in this environment.

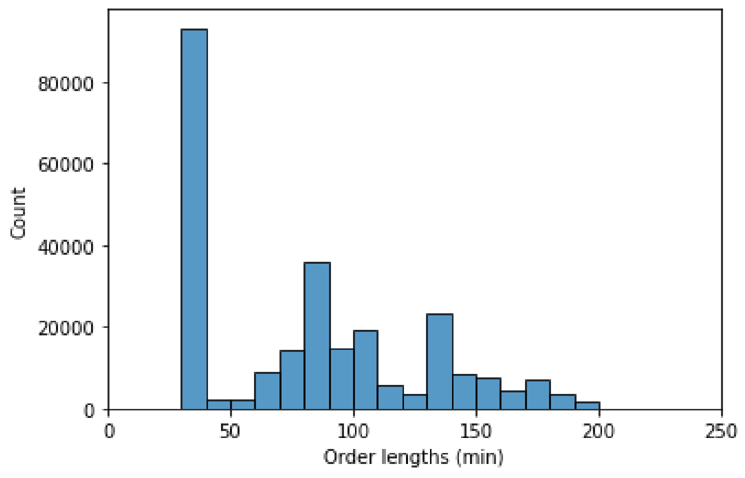


Figure 9 Histogram of the orders and the time it takes to do these.

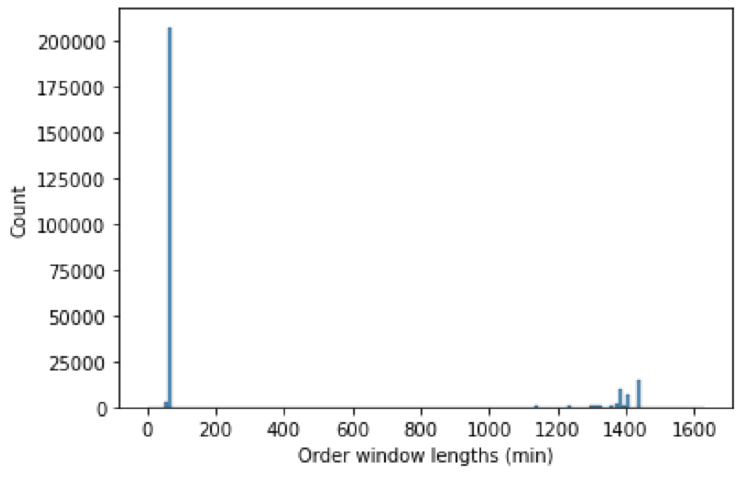


Figure 10 Histogram of the length of the time windows of the orders.

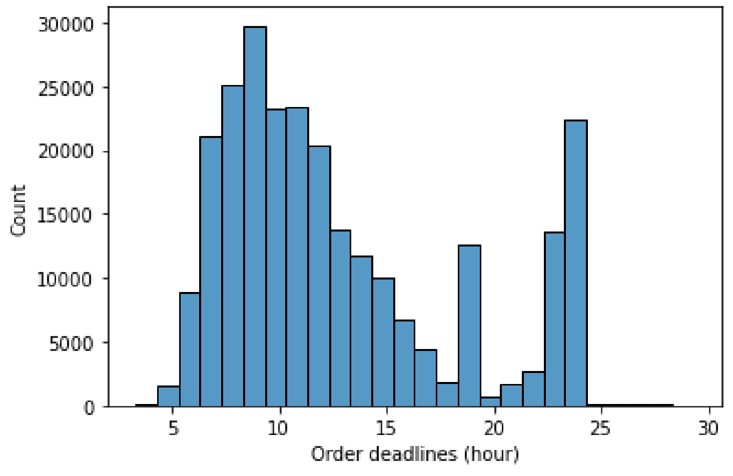


Figure 11 Histogram of deadlines of orders throughout the day.

Chapter 4. Simulator for reinforcement learning

The aim of this research is to investigate to what extent reinforcement learning can be used as a tool to aid in order distribution for self-organizing logistics. Reinforcement learning aims to optimize sequential decisions in a dynamic environment. A simulator is built which represents the real-life environment for container logistics planning van Berkel Logistics is dealing with. This simulation environment is based on the problem definition as described in previous chapter.

A general view of the interaction with the simulator is depicted in Figure 12. At every timestep the environment, consisting of trucks and orders, is observed. A truck which is first available can choose an order which it wants to do. This choice can be made in several ways; from random choice to a simple heuristic or an advanced decision model. This chosen order is subsequently communicated to the environment and executed there. This changes the environment, the truck which chose the action will be occupied till later in the day and the chosen order will be marked as done. The effect of doing this order can be measured in whether and how late the order was executed compared to its deadline. This lateness is communicated back to the truck agents and can be used for optimization. The environment is in its final state when no orders need to be done anymore and all trucks have moved back to their main location.

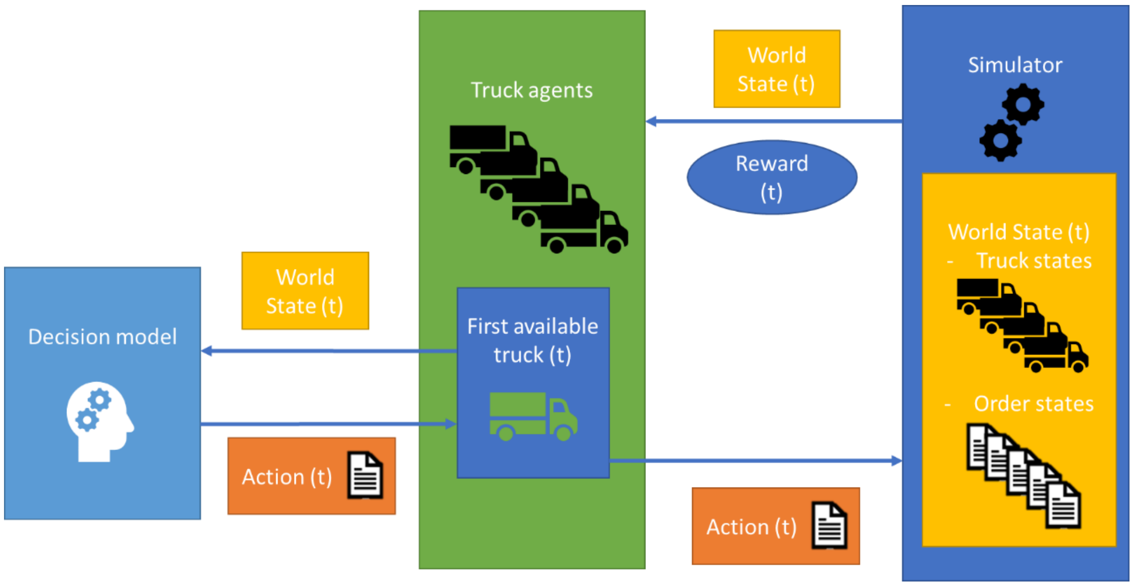


Figure 12 General view of the container logistics simulator, the steps which happen at every decision at time t are depicted with arrows. The environment gives back the reward observed due to the action at the previous decision moment as well.

No planning ahead

A logistics planner typically does not assign orders to trucks linearly from the start till the end of the day. In reality, a planner can plan orders ahead and subsequently assign orders which happen earlier on the day when trucks are still idle. In this way the planner can also estimate whether he should rent additional trucks or not. While such an approach is feasible for a planner, it is not easily translated to a reinforcement learning setting.

For reinforcement learning, preferably a close connection is desired between the performed action in an environment and the feedback obtained as effect of doing a certain action. Therefore, the direct execution of an order when assigning to a truck is deemed the most logical approach.

When an order can be planned ahead, the effect of planning that order is not observed until the order is actually executed. Such discrepancy makes it harder for decision models to learn the effect of their actions. Furthermore, it will be a challenge to make the action space such that rescheduling is possible as well. And an additional question is when orders will be executed as orders can be (re)scheduled at any time within this approach.

Simulator details

A detailed description of the simulation environment and its elements will be given here. The environment is defined by a set of orders, i.e. container movements which need to be done, and the trucks which can do these orders. As trucks do orders which have different lengths, the current time of the environment is represented by the earliest time a truck is available and an action can be taken.

Order

Orders are typed by their origin (where the container is located) and destination (where the container needs to be moved to). In addition, orders have an earliest time the order can be started as well as a deadline before it needs to be started. Furthermore, every order contains a duration which represents overhead time it takes to do this order apart from time it takes to move the container. Not all orders can be done by all trucks due to their origin and destination address type. The determination of the order type can be found in Table 2.

Address

For orders, addresses are used to indicate where from and to a container needs to be moved. An address has latitude and longitude coordinates, typically located in the Netherlands or just over the border in Germany or Belgium. Addresses have a type which indicate which truck can visit these. An overview of truck-address combability can be found in Table 1. Furthermore, addresses have a service type; either craning, decoupling or live-loading. This service type determines how containers are handled on these locations. This is further explained in the paragraph Reinforcement learning details.

Truck

Trucks contain basic information on their current location (as a latitude and longitude) and time they are available to do a next order. Another property of trucks is their truck type and can be either terminal-Veghel, terminal-Cuijk, region or port. Not all orders can be done by all trucks though and this is based on the order's origin and destination address. When both addresses are located on the same terminal, Veghel or Cuijk, terminal trucks of that terminal can do these orders. Region trucks are allowed to do these orders as well, but can also visit addresses which are further away from the inland terminal (so called region addresses). Region truck are not allowed to go to port addresses due to emission restrictions. Port trucks are allowed on these locations and can effectively do all orders. While not moving a container, trucks can have an empty chassis with them which they can use for picking up orders from a craning location. Further information is explained in the paragraph Reinforcement learning details. Lastly, trucks have a main address where they start and end their days.

Reinforcement learning details

An essential part of a reinforcement learning environment are the actions that can be taken and their effect on the environment. Any of the trucks that are available first can take an action based on the available orders which need to be done. In addition, a truck can choose to wrap up its day and go back to its main location. Whenever a truck drives to a location distance and duration is calculated as shown in Figure 4.

Actions

A truck can do an order if its compatible with the type of the truck. The truck first makes sure it has appropriate chassis status; either no chassis or an empty chassis. When the order starts at a craning address, it needs to have an empty chassis and in other cases it should make sure it does not have a chassis with it. If there is a mismatch, the truck goes via an inland terminal and picks up or drops of an empty chassis. This change takes 15 minutes.

The truck thereafter drives to the origin location. Depending on the time window of the order, different things can happen. If the truck is too early, it needs to wait until the earliest time of doing the order before it can start the order. If the truck arrives within the time window, the truck will do the order right away and no lateness is encountered. When the truck arrives at the origin location after the deadline, the order can also start directly but there will be a lateness penalty.

The truck then transports the container to the destination and takes time to process (based on the order duration). When the destination is a live-loading location, the truck drives back the processed container to the origin address before the truck is available again. When the destination accepts decoupling, the container will be decoupled and the truck is directly available for the next order. If the service type of the destination address is craning, the truck ends with an empty chassis. It either needs to pick up another container from a craning location, or drop off the chassis at a van Berkel inland terminal before it can do another order.

Finishing the day

When no compatible orders are available for a truck to do, a truck decides to wrap up its day. The truck needs to bring back its empty chassis to an inland terminal if needed and can then go to its main address (likely an inland terminal as well).

Reward

An environment used for reinforcement learning needs to return a reward which can be used for learning. Reinforcement learning aims to maximize the long-term cumulative reward. We would like to minimize the total lateness of the orders at the end of the day as this corresponds to what van Berkel Logistics optimizes while planning its trucks. As reward the negative of the lateness observed is used, meaning that maximizing this reward minimizes the lateness.

Split lateness

After an order has been done in the environment, the lateness can be calculated for the order which has just been finished. The formula is depicted in Equation 5.

Equation 5 Lateness calculation for order O when started at t_{start}.

$$\mathrm{L(O)} = \max\left(0, t_{start} - O_{deadline}\right)$$

An alternative approach is to already calculate partial lateness for all orders which were already due by the moment the next action can be taken, see also Equation 6. In this way partial lateness is calculated for every open order. For example, when an order needs to be done by 10:00 and it is not done yet by 11:00, already 1 hour of lateness is returned from the environment for that order. The rationale is that it is easier to optimize for lateness this way as there is more direct feedback when orders are running late.

Equation 6 Split lateness calculation between the moment the truck takes an action ($t_{current}$) and the next moment a truck can take an action (t_{next}). t_{start} is the time the chosen order will be started at the origin of the order.

$$\begin{aligned} L_{split} = {} & \min\left(\max\left(0, t_{start} - O_{chosen_{deadline}}\right), t_{next} - t_{current}\right) \\ & + \sum_{\substack{O \in Orders \\ \mid O \neq O_{chosen}}} \min(\max(0, t_{next} - O_{deadline}), t_{next} - t_{current}) \end{aligned}$$

Chapter 5. Deep Q learning models

Q learning is a version of reinforcement learning which aims to predict the quality (Q) values of doing actions in certain environmental states. In our setting this corresponds to a truck predicting the quality of choosing an order based on available orders and the states of the other trucks. In deep Q learning neural networks are used to predict these Q values. This decision mechanism is often referred to as the policy. Deep Q learning has proven success in learning to do good actions in environments based on obtained rewards [9] and this approach can be applied to our simulation environment as well.

Main challenges

When comparing existing applications of deep Q learning to our environment of trucks needing to choose orders, there are several challenges to overcome. Well known examples of applications of Q learning are learning Atari games and the game Go. These games can be represented in a fixed feature space, namely the pixels on the screen and stones on the Go board respectively. In addition, there are a fixed set of actions that can happen. For Atari games these are limited number of button-presses and for the game Go there is a fixed number of locations stones can be placed in.

However, an easy translation of our container distribution environment to a fixed feature and action space cannot be made. Between episodes, as well as during an episode, the number of available orders and active trucks can be highly different. This makes it hard to put all these states into a single feature space. Furthermore, as we are working with sets of trucks and orders, we do not care about the order of these elements. However, when we represent these elements in a feature space the order matters for the network and several different representations can be constructed for the same environment state. The aim now is to construct Deep Q networks which are invariant for the order of the elements for their prediction outcome. Another challenge is how to represent the actions that can be taken in a Deep Q network. We look here at a set of orders which can be chosen from, and this changes for every environmental state that is occurred. This cannot be easily generalized as can be done with button presses in an Atari game.

The problem of such a variable action space is encountered more often in literature, but a straightforward approach to tackle this is not easily found. One solution is to embed the action in the state itself [32]. The output of the network can then be interpreted as Q value. Alternatively, it is found that one can also predict a fictional action with a DQN and use something like nearest neighbour search to select the best matching action based on this fictional action [33]. Lastly, one can also categorize the actions and predict a category instead of an action [34]. In this case a random action from the chosen category is picked. In the following paragraphs we tried to tackle the problem of variable actions space during the construction of the three deep Q learning models.

State representation

The container distribution environment is defined by the set of trucks and orders. For both a truck (Table 6) and an order (Table 7) a vector representation is made such that these can be used as building blocks to construct full representations for environmental states. These environmental state representations are then used for the deep Q networks to predict actions to take. A key design choice in the representation is that certain values are shifted in respect to the truck which is currently making a decision. The location and time of the currently choosing truck is considered the centre of the universe and all values are calculated relative to these values. For this reason, when the choosing truck is explicitly referenced in a feature representation, its time available and location is therefore omitted. Lastly, for both trucks and orders, a Boolean is added which indicates that it is not a dummy input. As will be explained when describing the learning models, sometimes dummy values are needed to fill up the inputs of the network.

*Table 6 General truck representation as feature vector. Elements with an asterisk * are calculated relative to the choosing truck.*

	Example value	Note
type_terminal_veghel	0 or 1	One-hot encoding for the type of truck
type_terminal_cuijk		
type_region		
type_port		
has_empty_chassis	0 or 1	Whether the truck has an empty chassis when it is next available
time_available*	0.0 or higher	Time and location information. Omitted when the currently choosing truck is described.
latitude*	Any	
longitude*	Any	
is_valid_truck	1	Used to distinguish real trucks from dummy inputs

*Table 7 General order representation as feature vector. Elements with an asterisk * are calculated relative to the choosing truck.*

	Example value	Note
type_terminal_veghel	0 or 1	One-hot encoding for the type of order
type_terminal_cuijk		
type_region		
type_port		
start_latitude*	Any	Coordinates of the origin address of the order
start_longitude*	Any	
start_craning	0 or 1	Indication whether the start location is a craning address or not (decoupling)
end_latitude*	Any	Coordinates of the end address of the order. Can be the same as the origin location in case of a live-loading order
end_longitude*	Any	
end_craning	0 or 1	Indication whether the end location is a craning address or not (decoupling)
earliest_start_time*	Any	Earliest time the order can be started
deadline*	Any	Latest time the order should be started before a lateness penalty will occur
time_window	0.0 or higher	Time between earliest_start_time and deadline
order_time	0.0 or higher	Time and distance it takes to do the order
order_distance	0.0 or higher	
origin_time	0.0 or higher	Time and distance it takes before order can be started as the truck needs to drive to the origin
origin_distance	0.0 or higher	
expected_lateness	0.0 or higher	Expected lateness occurred when the order is chosen
is_valid_order	1	Used to distinguish real orders from dummy inputs

Models

Three different deep Q learning model have been developed; the OnePass, Iterative and Attention model. Each model aims to predict Q values for doing certain orders in a state by a truck. These models are also referred to as the policy and whenever the network is used to make a choice, the order with the highest Q value is chosen from the compatible orders. When no compatible order exists, the truck wraps up its day and goes back to its main location.

The OnePass model closely represents the models used by Mnih et al. [9] where all outputs of the network correspond to individual actions. Due to the many outputs in the OnePass model, it might be hard for the model to individually evaluate all these actions in one forward pass. The Iterative model is proposed to overcome this problem. It evaluates one order at the time and therefore has just one output. Both these models have the problem that their input space is of a fixed length and not invariant as one environmental state can be represented by a multiple number of vectors. The Attention network is proposed to overcome this problem as it works on sets of arbitrary size instead.

OnePass model

Of the models, the OnePass model most closely resembles the approach deep Q learning is applied in Atari games and Go. The whole environmental state in represented as a feature vector to the network and there are as many Q value outputs as there are order inputs in the network, see also Figure 13. This means that all orders are evaluated with one pass through the network.

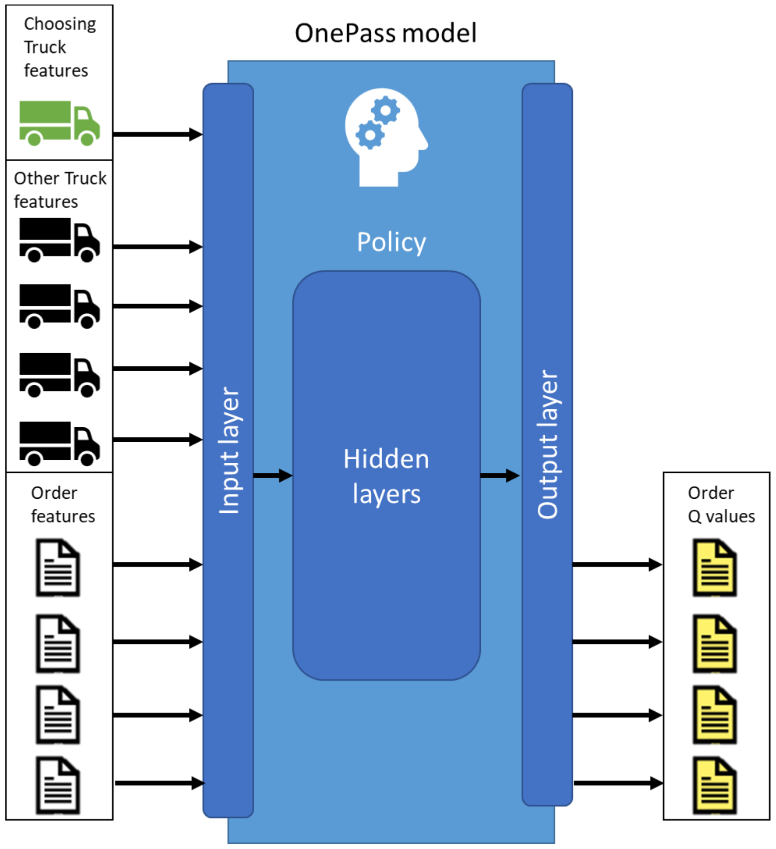


Figure 13 The OnePass model. For all orders Q values are calculated at the same time.

The network can be initialized for an arbitrary order number-truck number combination, which is represented as one big feature vector to the network. The input consists of; the current truck that is choosing an order, all other active trucks that are present in the environment and all orders that still need to be done. Due to the nature of neural networks, there is a fixed number of orders and truck that can be part of the input. When the input is bigger, the network cannot be used for predicting Q values. When there are less orders or trucks than there are corresponding inputs for the network, the remaining inputs will be filled by “dummy” inputs, consisting of only 0 values. The order outputs corresponding to these inputs will be ignored when the best order is determined. The idea is that the network is able to recognize that it should not pay attention to these dummy inputs and base the Q values on the non-dummy inputs.

Due to the way this network is built up, the order of the trucks and orders in the state representation can influence the by the network estimated Q values. However, we would like to have a network which does provide the same Q values independent on how these trucks and orders are ordered in the state representation. In an attempt to make the network so-called permutation invariant, the non-choosing and dummy trucks are always randomly assigned to the truck inputs of the network. The same is done for the (dummy) orders. In this way it is less likely bias for certain outputs will be created in the network while training.

The information goes to 0 or more hidden layers via a specified hidden representation. The output layer has as many outputs as number of order inputs present. All forward passes except the output layer are activated via a Rectified Linear Unit (ReLU) activation function[6]. ReLU was chosen due to its wide popularity in deep learning [35].

Iterative model

The iterative network is built up similar to the OnePass network. However, as the name implies, the orders are not evaluated in one pass, but iteratively for their Q values. For this reason, the network only has one output; the Q value for the evaluated candidate order, see also Figure 14.

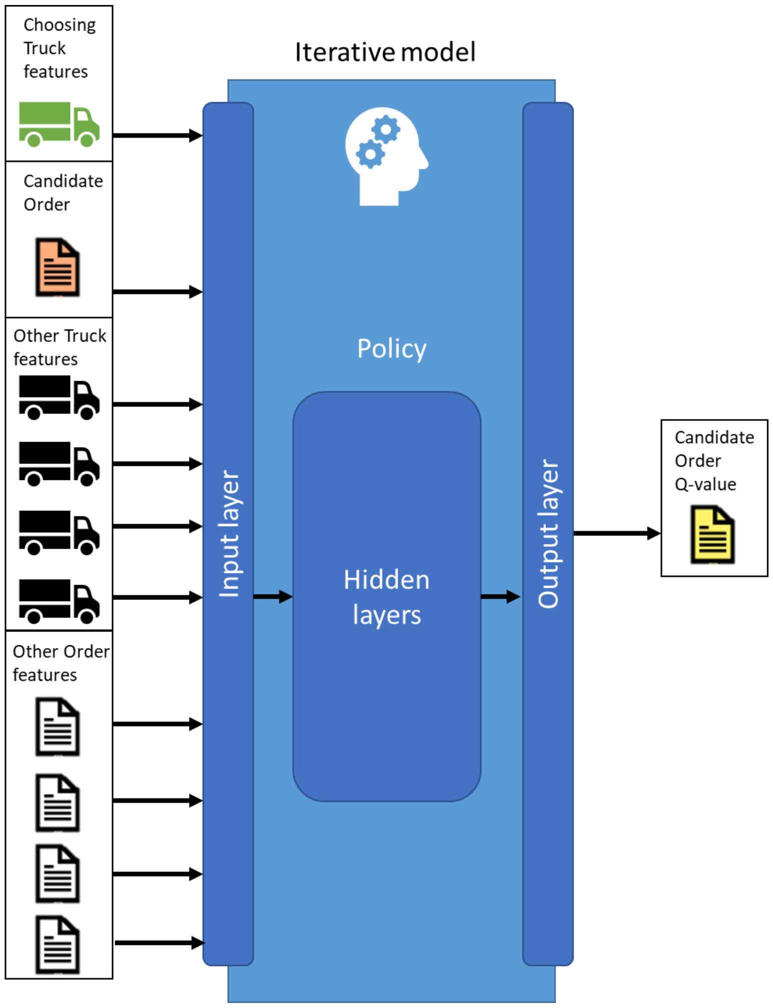


Figure 14 The Iterative model. One order is evaluated for its Q value within one forward pass.

[6] https://en.wikipedia.org/wiki/Rectifier_(neural_networks)

Similar to the OnePass model, this network can be initialized for an arbitrary number of trucks and orders. The input of the network now contains explicitly the candidate order as input for which the Q value will be calculated. In addition, the other orders and trucks are part of the input as reference. Again, like the OnePass model, if there are less trucks or orders in the current environment, dummy values need to be used. Furthermore, hidden layers and activation function in the network are the same as in the OnePass network. Again, the other trucks and other orders are randomly assigned to inputs for every evaluation to circumvent bias to make the network permutation invariant.

The rationale for this model is that the network can more easily match the order at the dedicated input to the output than it can in the OnePass model, where the network needs to figure out by itself which order input corresponds to which output. However, a downside is that for every decision, all compatible orders for a truck need to be checked with the network, which makes decision making slower.

Attention model

Both the OnePass and Iterative model are not flexible in input size. They can be initialized for a certain number of orders and trucks and are not compatible for larger instances than the chosen size. Furthermore, when environment states with a smaller number of orders or trucks are evaluated, dummy orders need to be used, which might confuse the network.

As alternative, the Attention network is proposed as a third model. Attention networks work on arbitrary sets of elements which can exchange data via transformer encoder layers. For this reason, we are not bounded by the predetermined number of inputs of the network and do not need to use dummy values either. The model architecture is shown in Figure 15.

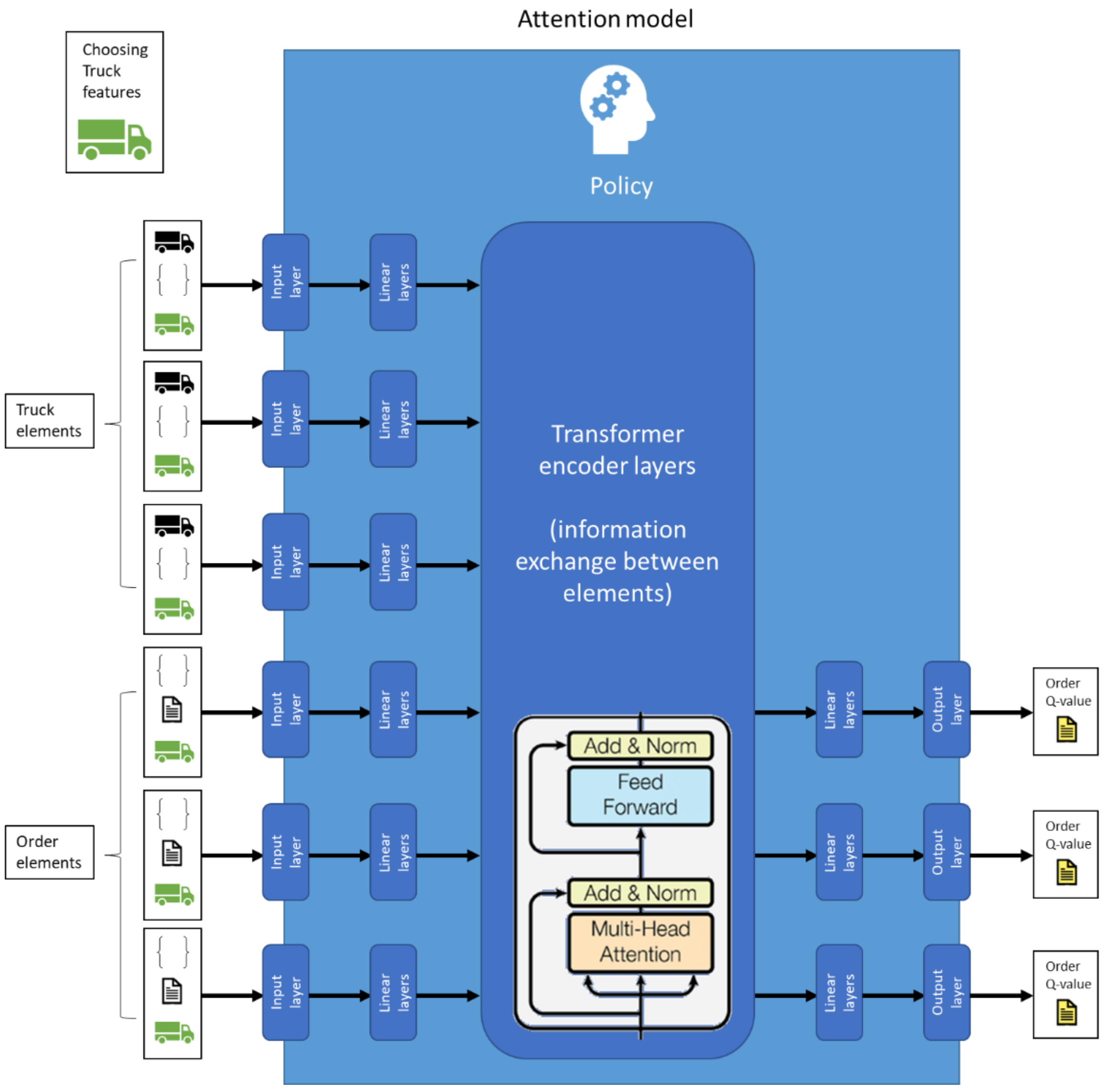


Figure 15 The Attention model. An arbitrary number of orders can be evaluated at the same time for their Q values. The transformer encoder layer diagram is taken over from Vaswani et al. [17].

For every element (available order or active truck) a separate input is created in the same feature space. An element has therefore input features for both an order and a truck. In case the element is an order, the first features of the node represent the order as described in Table 7. For the inputs which are present for describing a truck, the inputs are set to 0. Vice versa happens for input elements which represent trucks. These have a feature vector which start with 0s on the places where order information should be present. Hereafter the truck information is put as shown in Table 6. To make it clear to the network which truck is choosing currently, the features of this truck are appended to every element as well.

Every input element is first put through an input layer which converts it to a hidden representation. The element can thereafter go through 0 or more hidden layers before it is put in the transformer encoder layers. In the transformer encoder layers the elements can exchange information. In this way the orders become aware of the other orders and trucks in the system. For the transformer encoder layers a default of 8 heads and 0 dropout is used. The internal representation of the transformer encoder layers can have a variable size. After the transformer encoder layers the outputs of the orders are fed through some linear layers before they are converted to one value in the output layer for every element. This output value is interpreted as the Q value for the order elements. Except for the transformer encoder layers and the output layer, the ReLU activation function is used within the network.

Model Training

Although the three proposed models differ in their architecture, the way these are trained all follow the same method (apart from some implementation details). In the simulator instances are generated and interacted with by doing actions. These actions are chosen via an ε-greedy mechanism, meaning whenever a truck is available first and needs to choose an order, it chooses a random compatible order with chance ε. In other cases, it chooses an order based on the choice of the policy network.

While doing this, experiences are obtained. These experiences consist to the environmental state at time t, including the truck choosing at that time, the actual order chosen and the next environmental state, including next choosing truck. These experiences are saved and can be used to learn from. In case a truck cannot do an order and needs to wrap up its day, the reward obtained for that action is added to the next moment a truck can actually do an order. In this way all experiences are between moments that orders are chosen. In Figure 16 is shown how the Q values of the network are trained. A target network is used as a delayed copy to make the training more stable.

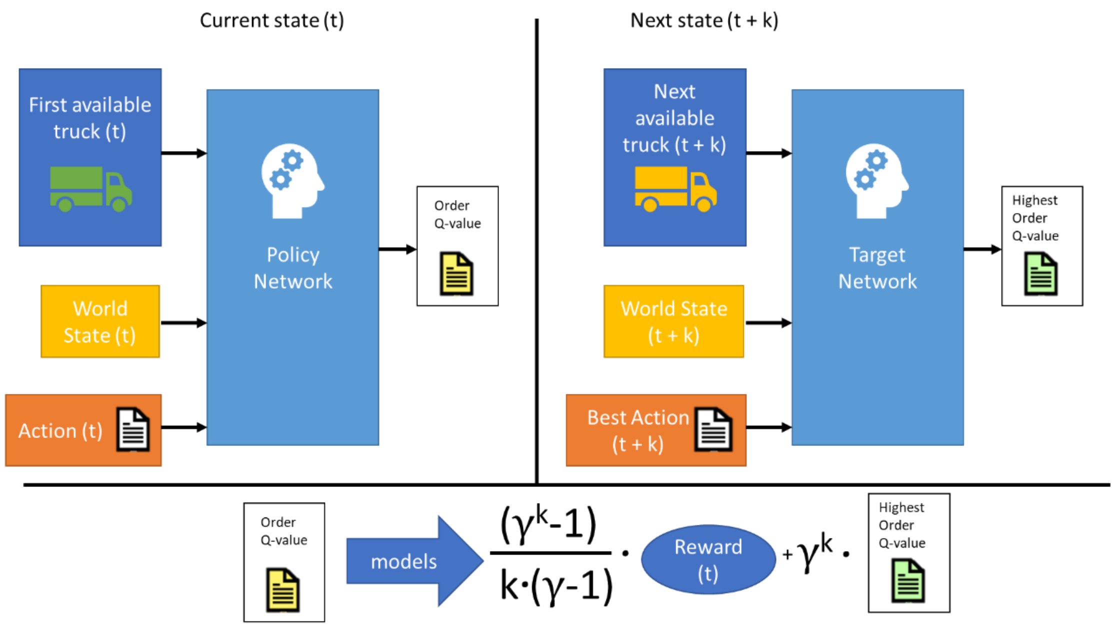


Figure 16 Model training scheme. The aim is to train the policy network such that the predicted Q value corresponds to the discounted reward plus the discounted Q value of the best action that can be taken by the next truck in the next state.

In Markov Decision Processes (MDPs) where every timestep is equal to 1, the Q value of the next state is discounted by a factor γ as indicated by the Bellman equation (Equation 3). However, due to the nature of our problem, we do not have fixed timesteps and cannot just directly multiply the Q values of the next state with γ. As we are working with a Semi-MDP (semi-Markov Decision Process) wherein the timesteps are irregular between actions, we should scale the obtained rewards accordingly as described in [28].

We consider the state at time t, s_t, the reward R which is being received after doing an action in that state, the next moment an order can be chosen, time t + k and the state s_{t+k}. The reward must be discounted appropriately over that period of k timesteps. For simplicity, we assume that the reward R is obtained evenly within that period of k timesteps:

$$R = \frac{R}{k} + \frac{R}{k} + \cdots + \frac{R}{k} + \frac{R}{k} \ (k\ times)$$

If we discount the obtained reward for every timestep by γ, the total discounted received reward becomes:

$$R_{discounted} = \frac{R}{k} + \gamma\frac{R}{k} + \gamma^2\frac{R}{k} + \cdots + \gamma^{k-2}\frac{R}{k} + \gamma^{k-1}\frac{R}{k}$$

If we apply the Bellman equation to this discounted cumulative reward, we obtain then obtain:

$$Q(s_t, a_t) = R_{discounted} + \gamma^k V(s_{t+k})$$

When we consider that 0 < γ < 1 and k ≥ 1, we can rewrite the equation of $R_{discounted}$:

$$R_{discounted} = \frac{R(\gamma^k - 1)}{k(\gamma - 1)}$$

Yielding the final Bellman equation which can be used for training in this Semi-MDP.

Equation 7 Bellman Equation for a Semi-Markov Decision process.

$$Q(s_t, a_t) = \frac{R(\gamma^k - 1)}{k(\gamma - 1)} + \gamma^k V(s_{t+k})$$

Chapter 6. Experimental setup

The performances of the models mentioned in Chapter 5 are tested with the simulation environment. Every test run consists of 10000 episodes wherein in every episode a newly generated instance is used. An episode number of 10000 was chosen as it was experimentally verified that the models generally converge within these number of episodes. Actions are chosen by the models via an ε-greedy approach where the model chooses an order either randomly (with chance of ε), or an order based on the model's output (chance 1-ε). The value of ε is slowly decreased during the training process and is calculated via Equation 8. A visualization of this function is shown in Figure 17.

Equation 8 epsilon (ε) calculation based on number of episodes.

$$\varepsilon = 0.9 * e^{-\frac{n_{episode}}{5000}} + 0.1$$

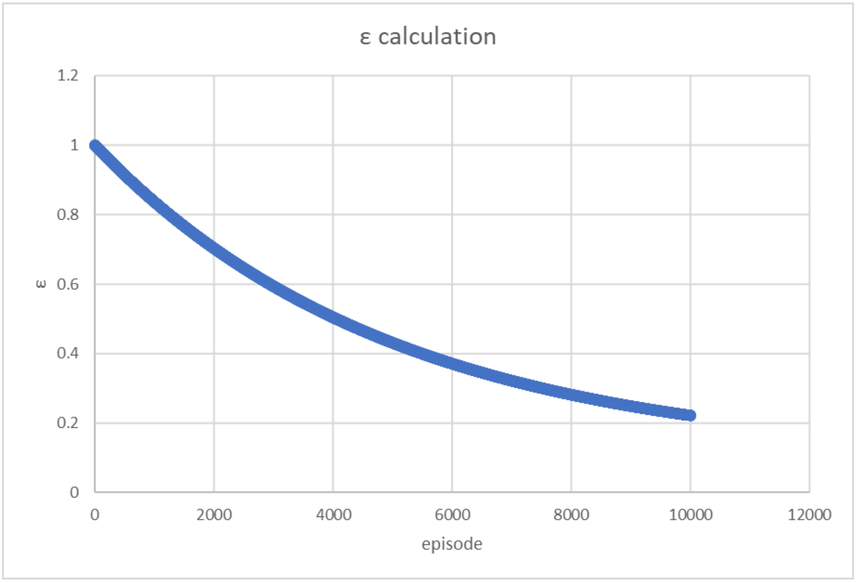


Figure 17 ε calculation based on number of episodes.

While interacting with the environment, experiences are obtained which can be used for training the model. A maximum of 2^{16} experiences are saved in a replay buffer. After every episode 128 experiences are samples from this replay buffer and used to train the models on. Every 50 (OnePass and Iterative model) or 250 (Attention model) episodes, the network parameters of the policy network are copied over to the target network as these values were found to work best for these networks. As the OnePass model and the Iterative model have a maximum number of truck and order inputs, these models will be initialized with as many inputs for trucks and orders as are as defined by the scenarios at the start of the day. All experiments were run in 10-fold on the High Performance Cluster (HPC) of the TU Delft[7].

[7] https://login.hpc.tudelft.nl

Different scenario sizes

Four scenarios have been designed, as shown in Table 8, based on what is observed in historic data. Scenario 4 is corresponding to a typical real-life day for van Berkel Logistics and the other scenarios are scaled down versions of this scenario. For evaluation purposes within this research, we mainly focus on scenario 2, yielding 5 trucks and 38 orders. This scenario is of a size such that the planning problem is still challenging while we could train the models in manageable time. Unless mentioned otherwise, instances with synthetic orders based on scenario 2 are used for evaluation of the model.

Table 8 The composition of the four scenarios tested in this research.

	Trucks				**Orders**			
Type	Terminal Veghel	Terminal Cuijk	Region	Port	Terminal Veghel	Terminal Cuijk	Region	Port
Scenario 1	0	0	1	1	4	1	7	1
Scenario 2	1	0	2	2	12	2	21	3
Scenario 3	2	2	8	8	40	8	70	10
Scenario 4	4	3	15	15	80	15	140	20

Instance generator

A random new instance is created for every episode to enable the models to generalize for arbitrary instances. The way the trucks and orders for these instances are created is based on what is found in historic data and explained below;

Trucks

Different truck types can be generated and their number is based on the scenario size (Table 8). Generated trucks always start their day at 360 minutes (6:00) on one of the terminal locations (Table 3).

Orders

Orders can be either generated (synthetic orders) or sampled from historic data (real orders).

Real orders

Instances with real orders can be created by sampling orders from historic data. Historic orders of van Berkel Logistics have been extracted from database exports (see also Historic data) and can be used to sample from to create instances.

Synthetic orders

Orders can be generated with the use of generated addresses. The latitude and longitude are sampled uniformly between minimal and maximal values depending on the address type. The same counts for the service types, which can be live-loading, decoupling or craning. These distributions correspond to the historic data observed and deemed to be a proper representation of reality.

Table 9 Generated addresses including coordinates. For terminal locations, the addresses are generated relative to the locations of the inland terminals Veghel (Veg) and Cuijk (Cui), see also Table 3. Port addresses are generated on the Maasvlakte in Rotterdam.

Address type	latitude		longitude		Service type (fraction)		
	min	max	min	max	Live-loading	Decoupling	Craning
Terminal Veghel	Veg_{lat} - 0.025	Veg_{lat} + 0.025	Veg_{long} - 0.05	Veg_{long} + 0.05	0.2	0.8	0.0
Terminal Cuijk	Cui_{lat} - 0.025	Cui_{lat} + 0.025	Cui_{long} - 0.05	Cui_{long} + 0.05	0.9	0.1	0.0
Region	51.30	52.00	4.90	6.25	0.5	0.45	0.05
Port	51.80	52.00	4.00	4.70	0.0	0.0	1.0

A visual representation of generated addresses from this distribution is shown in Figure 18. One of the locations of the order will be an inland terminal (Table 3). The other location is generated based on the order type (Table 9). When the other location is not a live loading location, the inland terminal can be either the origin and destination of the order with a 50 percent chance.

For the deadline a moment on the day between 400 (6:40) and 1200 (20:00) minutes is chosen. The earliest time an order can be done is based on a time window which is sampled between 30 and 90 minutes. The earliest time will then be that number of minutes before the deadline. Lastly, the overhead of doing an order (processing time while not driving) is uniformly sampled between 15 and 45 minutes.

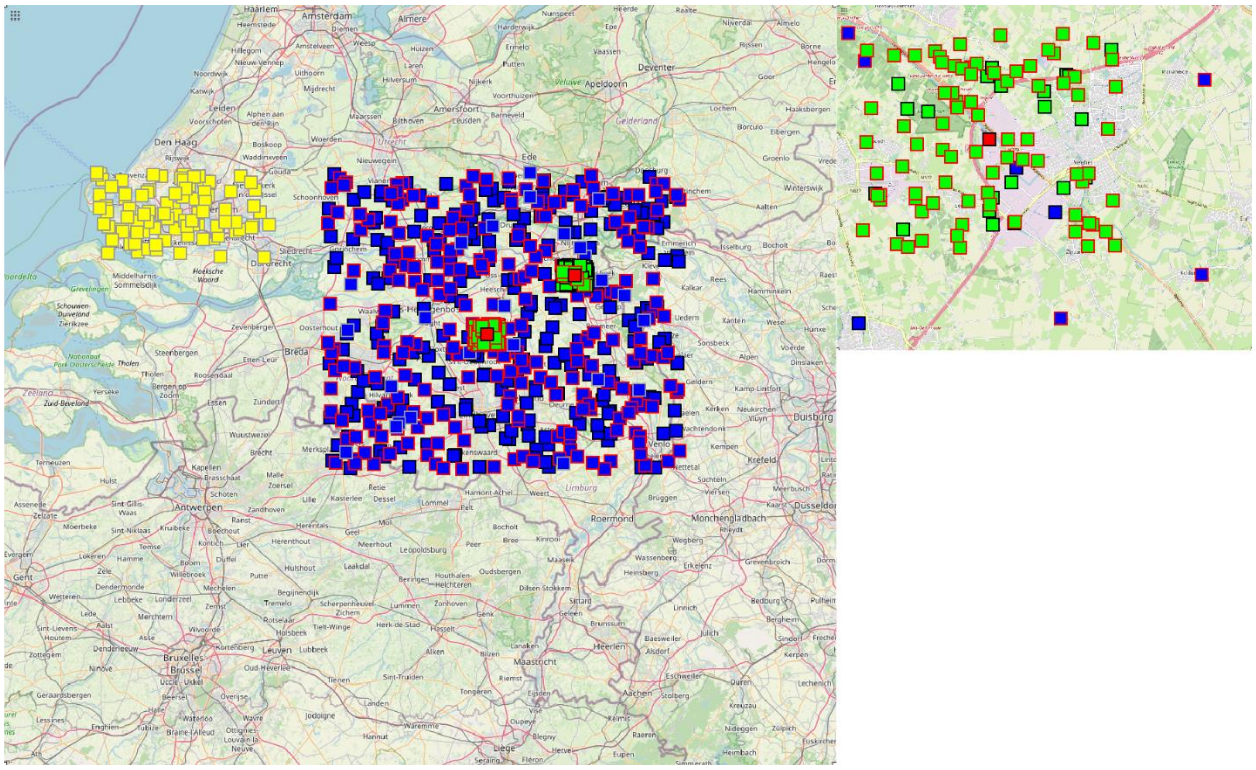


Figure 18 Synthetic generated addresses. Inside colour represents the address types: inland terminal (red), terminal customer address (green), region address (blue) and port address (yellow). The outline represents the location type: decoupling (red), live-loading (black) and craning (grey). On the right a close-up of the terminal of Veghel is shown.

Alternative scenarios

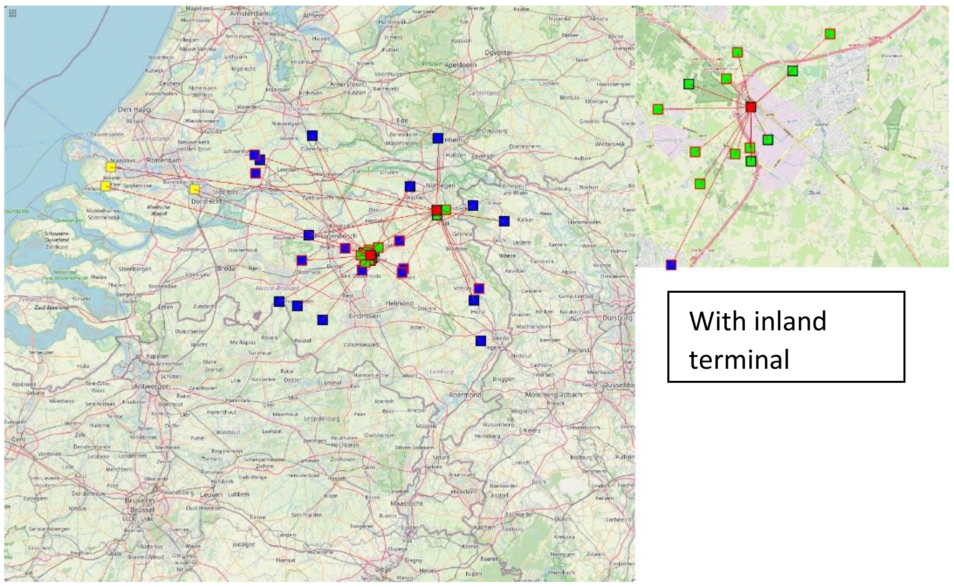
Our default problem setting is quite restrictive for the trucks. The time windows when an order should be done is usually rather small (around 1 hour) and the orders are always from or to an inland terminal. This aspect makes the problem more a scheduling problem than a vehicle routing or traveling salesman problem. To enable some more freedom for the models to optimize, 2 relaxations have been designed.

No earliest start times

For the standard environment, we have times orders can be done earliest. Whenever a truck arrives earlier than this time, the truck normally needs to wait before the earliest time has started. With this relaxation, we remove these earliest times such that, while planning, the truck only needs to take the deadlines of orders into account and does not have to wait for doing an order. This provides more flexibility for trucks when orders can be done.

No inland terminals

With this relaxation, synthetic orders are generated without having the inland terminal Veghel or Cuijk as start or end location. Instead, another location in the region is sampled (Figure 19). For terminal orders, a random location on the respective terminal is taken. The idea is that in this way, trucks can make more sophisticated decisions by planning their orders such that they minimize time they are driving to origin addresses. In this way the problem is more like a vehicle routing problem, although we still have coupled origins/destinations.



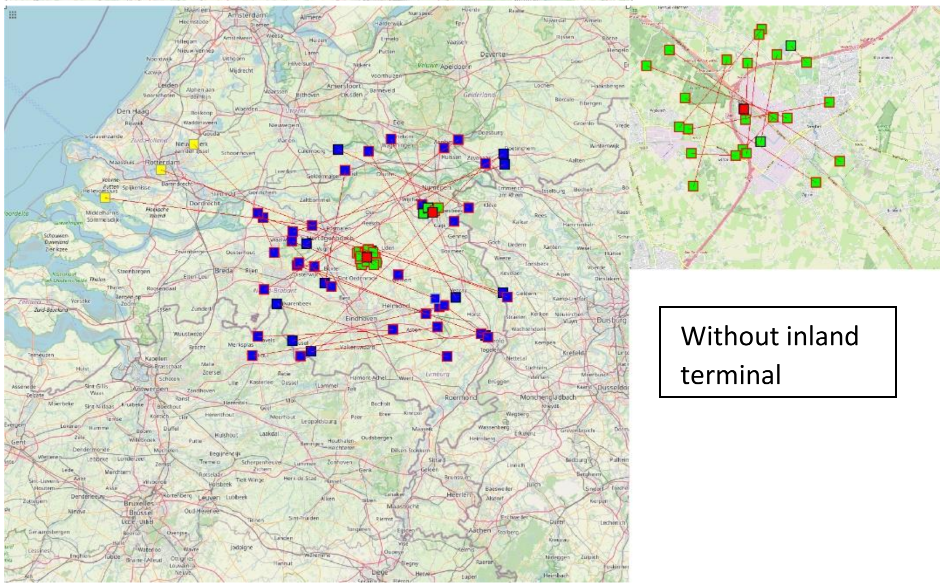


Figure 19 Orders generated with (upper) and without (lower) inland terminals. The red lines indicate the path over which the containers need to be moved. Inside colour represents the address types: inland terminal (red), terminal customer address (green), region address (blue) and port address (yellow).

Model hyperparameters

The reinforcement learning models can be tweaked by several hyperparameters to improve the performance. These hyperparameters are shown in Table 10 and the final values used for the models are depicted as well. In the appendices of this thesis (Chapter 10), you can find the plots of the different tested out hyperparameters to see the effect of changing these.

Two different loss criteria have been tested; the Mean Squared Error (MSE) Loss and the Smooth L1 loss [36]. For the optimizer, Stochastic Gradient Decent (SGD) [37] with a momentum of 0.9 and a learning rate of 0.001, RMSprop [38] and Adam [39] have been tested.

Table 10 Model hyperparameters. The values used for the final experiments are indicated as well. TEL is short for the Transformer Encoder Layer used in the Attention model.

hyperparameter	Tested values	Value used		
		OnePass	Iterative	Attention
Split lateness	[Yes, No]	Yes		
Use DDQN	[Yes, No]	No		
Loss criterion	[MSE, SmoothL1]	MSE	MSE	MSE
Optimizer	[SGD, RMS, ADAM]	ADAM	ADAM	ADAM
Gamma	[0.5, 0.9, 0.99, 0.999, 0.9999, 1.0]	0.999	0.999	0.999
Target update	[10, 25, 50, 125, 250, 500]	50	50	250
Dimension hidden representation	[64, 128, 256, 512]	256	256	64
Number hidden layers	[0, 1, 2]	0	1	N/A
Number hidden layers before TELs	[0, 1]	N/A	N/A	0
Number transformer encoder layers (TELs)	[0, 1, 2]	N/A	N/A	1
Number hidden layers after TELs	[0, 1]	N/A	N/A	0
Dimension TEL hidden representation	[128, 256, 512]	N/A	N/A	128

Heuristic baselines

To evaluate how well the models perform, the performance of the models (without random ε-greedy choices) is compared against four different heuristics.

Random

As baseline to compare against, a random heuristic is implemented. Whenever a truck needs to make a decision, a random compatible order is chosen from the available orders.

Earliest deadline

A straightforward heuristic to use is to do the order of which the deadline is the soonest. The truck does not take into account the order type as long as it is compatible with its own truck type.

Closest order

With this heuristic the truck chooses the order of which it can reach the soonest. It does not take into account the time window of the order though, meaning that the truck might need to wait often before starting an order.

Latest departure

Instead of solely looking at the deadline, we have another heuristic which also takes into account the time before it can be started. The urgency is determined by subtracting the time it takes to start the order from the deadline when the order needs to be started.

Key Performance Indicators

The quality of the models is measured via different Key Performance Indicators (KPIs). A description of these KPIs is given in Table 11. The KPIs lateness, distance and total time worked are divided by the result of random choices as these values are otherwise hard to compare among episodes with different instances. Furthermore, for the results a 50-episode moving average is used to even values out.

*Table 11 Key Performance Indicators (KPIs) measured during the experiments. KPI with an asterisk * are in the results normalized by dividing it by the result random choices yield.*

KPI	Description
Lateness*	Cumulative lateness of all orders done
Distance*	Cumulative distance which all trucks have driven together
Fraction orders on time	Fraction of the orders which were done on time and therefore did not lead to lateness
Total time worked*	Cumulative hours worked by all trucks combined
End time of the day	Last moment a truck has wrapped up its day
Fraction worked hours while loaded	Fraction of hours worked while trucks actually are handling or moving a container
Fraction driven kilometres while loaded	Fraction of driven kilometres while trucks actually are moving a container

Chapter 7. Results and Discussion

The three models have been evaluated by analysing their learning curves (averaged with a 50 episode rolling average). The learning curves for the OnePass, Iterative and Attention model can be found in Figure 20, Figure 22 and Figure 24 respectively. The KPIs other than the lateness are shown in Figure 21, Figure 23 and Figure 25. The performance values for the heuristics are shown as well in the figures to put the reinforcement learning models' performance in perspective.

The earliest deadline heuristic is very powerful and works well for reducing the lateness. The latest departure heuristic performs slightly worse than the earliest deadline heuristic and has the tendency to have a bias for orders which take longer to start, yielding more time driving without moving a container. As expected, the choose closest heuristic does not work that well in reducing the lateness as it does not take any deadlines into account in its decisions. The heuristic does give the best results in terms of distance as it drives around 80% of the distance random choices would yield.

All three models are able to learn from the environment and reduce the total lateness compared to what is observed when making random choices. The Iterative model is able to learn the environment in the fewest episodes among the models (Figure 22). The OnePass model is able achieve a similar performance, but takes more episodes to converge to that value. This difference can be explained by the point that the output of the network is more explicit in the Iterative network. There is a special input for the evaluated order and only one output corresponding to the Q value of that order. For the OnePass model there are many inputs and each input corresponds to their own output for their Q value. This needs to be learnt and tends to be more of a challenge. It should be noted though that the Iterative model evaluates every compatible order separately while making a decision, making it take longer to do an episode than the other models. An episode for the Iterative model takes roughly 4 to 10 times as long (depending on the scenario) as an episode while using the other models. Both the Iterative model and OnePass model converge to around the same values for all KPIs, indicating that these two models likely optimize similarly. The eventual performance of both models is similar to the best-found heuristic; choosing the earliest deadline. The models are however more efficient in the number of driven kilometres the average truck drives on a day. In addition, both models show an improvement in the fraction of the orders that are on time. This indicates that the models are not just looking for the earliest deadlines but also take other factors into account.

The Attention model (Figure 24) is able to learn from the environment as well and is able to reduce the obtained lateness compared to random choices. However, it is not able to achieve the performance which is observed in the other models. Furthermore, the learning is less stable as can be seen from the confidence interval in the graphs. In contrast to the other two models, the Attention model did not show a reduction in driven distance. It does show a better performance in the fraction of on time orders compared to the heuristic, indicating that the model does not just look at the deadline. The advantage of this network is that it scales better with the input size, something the other models as no dummy inputs are needed within this network.

OnePass model

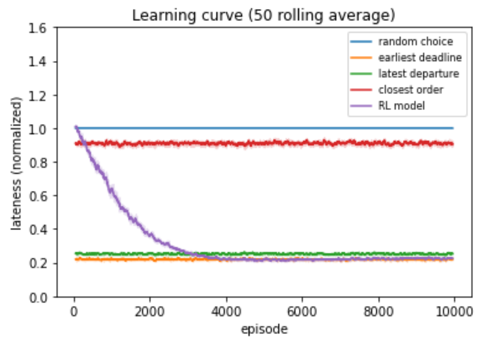


Figure 20 Learning curve of the OnePass model. Lateness is normalized based on the result from the random choice.

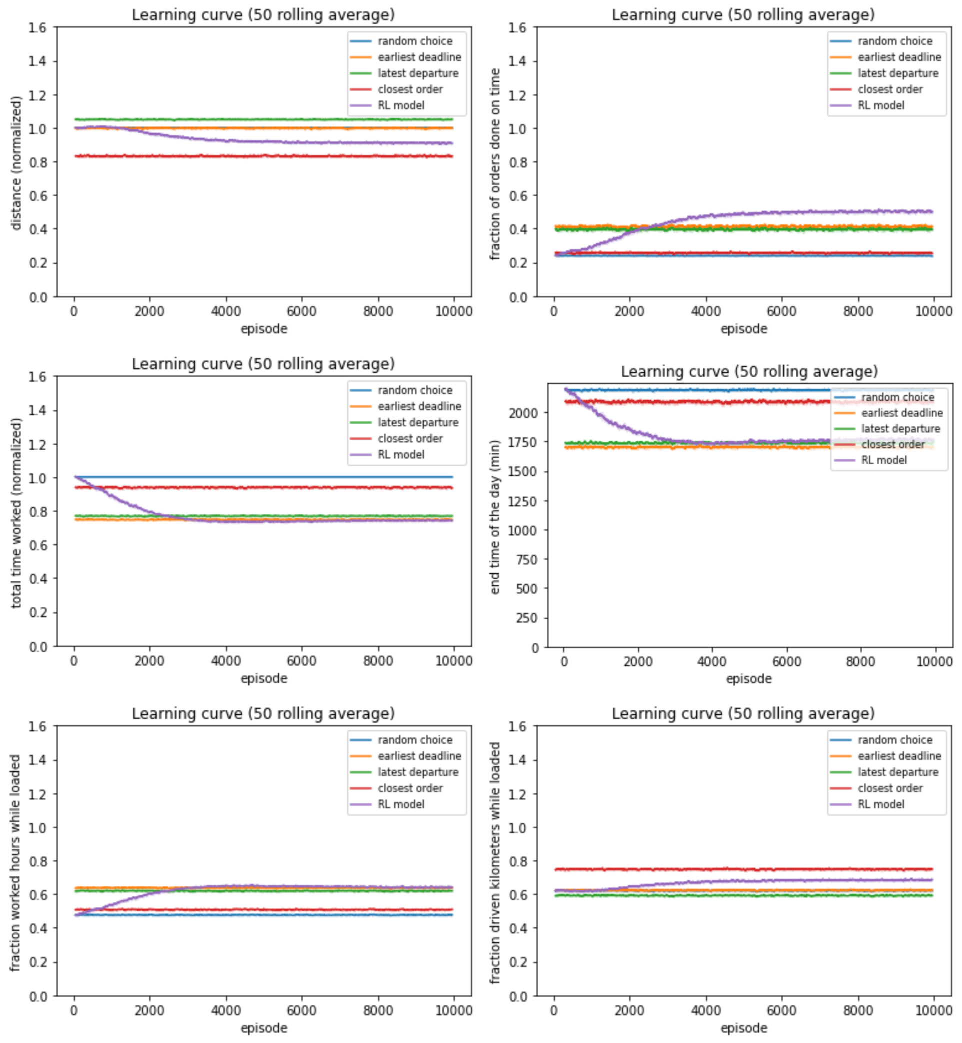


Figure 21 Other KPIs measured during the learning curve of the OnePass model.

Iterative model

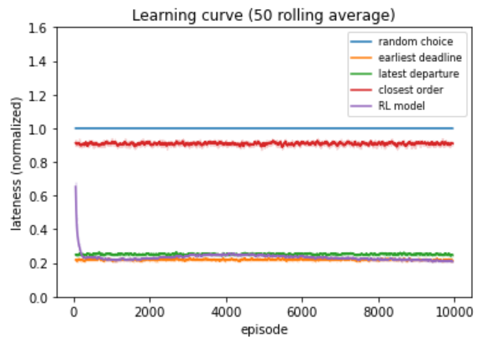


Figure 22 Learning curve of the Iterative model. Lateness is normalized based on the result from the random choice.

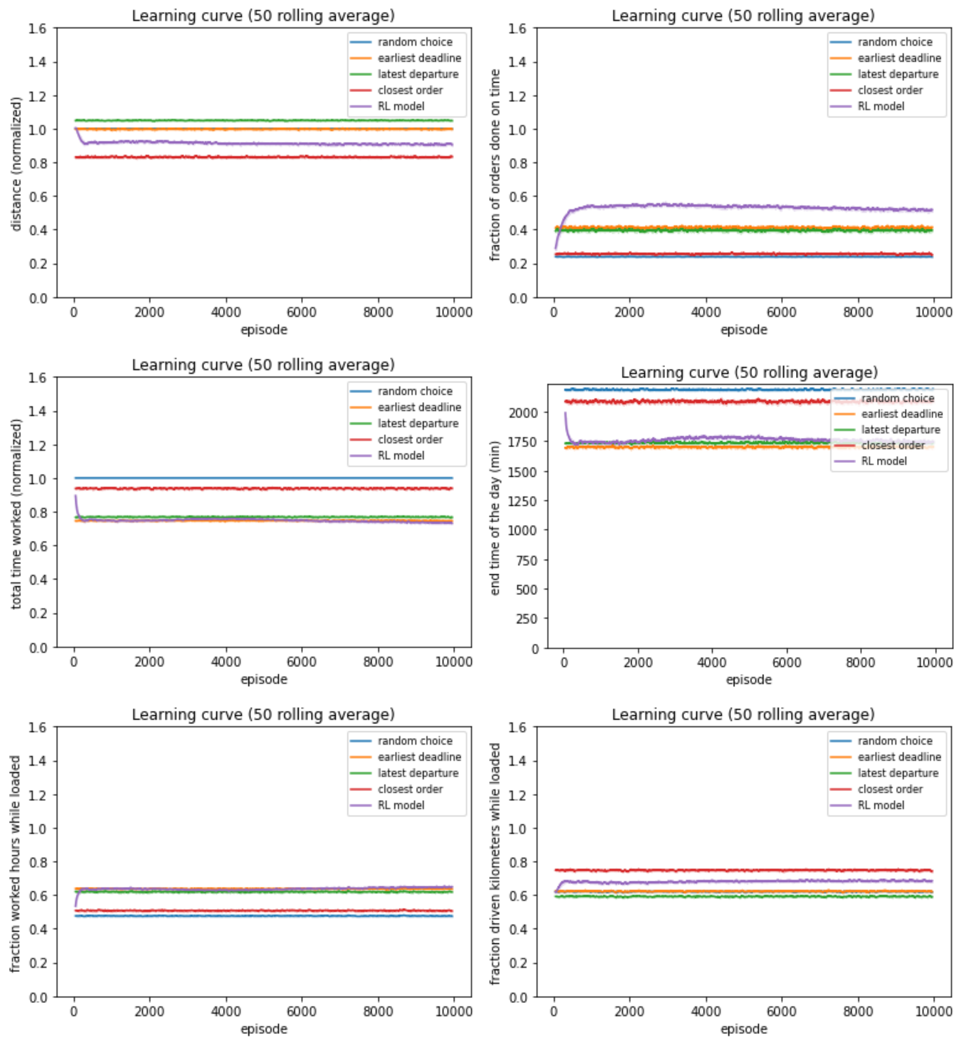


Figure 23 Other KPIs measured during the learning curve of the Iterative model.

Attention model

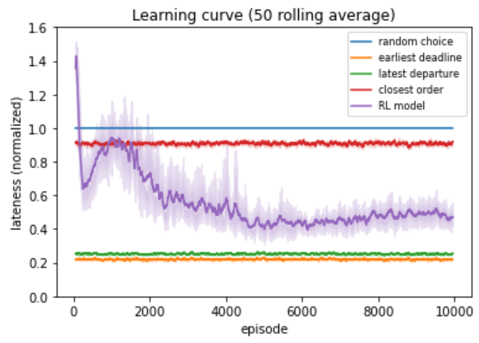


Figure 24 Learning curve of the Attention model. Lateness is normalized based on the result from the random choice.

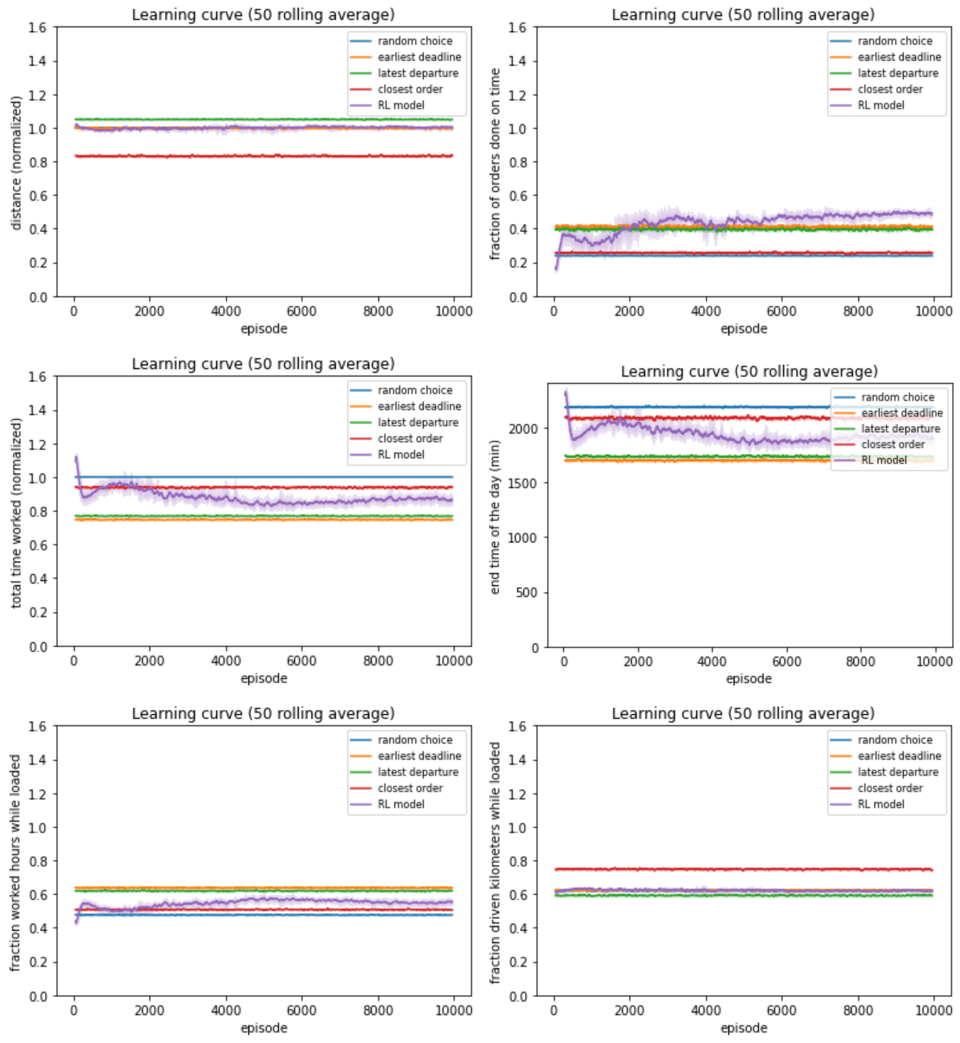


Figure 25 Other KPIs measured during the learning curve of the Attention model.

Alternative scenarios

The default scenarios tested out are quite restrictive due to all orders having just an order window of around an hour and one of the locations is always an inland terminal. This makes the earliest deadline heuristic very powerful and leaves less space for optimization in other directions. As extension of the base scenario, two relaxations have been made. Firstly, time windows are removed so orders can be done at any time of the day; the truck does not need to wait till the earliest time of an order anymore. In the second relaxation, no inland terminals are used for orders anymore, giving more room for spatial optimization. These scenarios are tested for both synthetic orders and real orders. No fixed inland terminals for orders could only be done with synthetic data. The results for the OnePass, Iterative and Attention models can be found in Figure 26, Figure 27 and Figure 28 respectively.

The OnePass model shows similar performance to the heuristics when orders are not fixed as origin or destination. However, when the time windows are relaxed, it is observed that the OnePass model optimizes less fast to the performance observed by the heuristics. This is a bit of surprise as we would expect that more freedom in doing orders would lead to a higher performance. It is postulated that the results are due to the random choice and other heuristics becoming more powerful as well (as the truck do not need to wait anymore before starting orders). The comparison between synthetic data and real data shows similar performance, indicating that the synthetic data properly approximates real data.

The Iterative model is able to optimize well for the different scenarios. The performance when the orders are not fixed to an inland terminal location are again similar like we saw as well for the OnePass model. This indicates that its either harder or not as beneficial for the models to optimize spatial distance than thought beforehand. Interestingly, when we relax the time the truck can do orders, we see for the Iterative model that it is able to beat the heuristics, showing added value of the reinforcement learning model. Possibly the model is able to make a better distinction between the truck and order types and is so able to make a better order allocation. When we compare the results to real data, we see that it is able to learn similarly, though it is not able to beat the heuristics as well as it does for synthetic data.

The Attention network shows again that it can learn the different alternative scenarios by beating the performance of random choices, both for synthetic as well as real data. It is however not able to properly beat the heuristics yet.

OnePass model

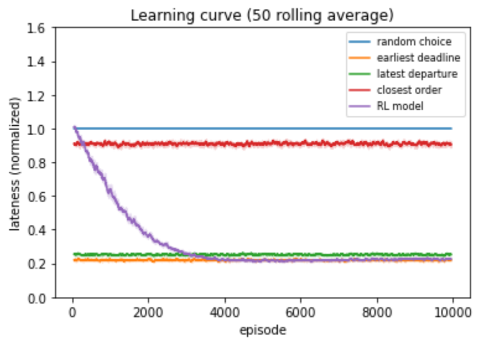
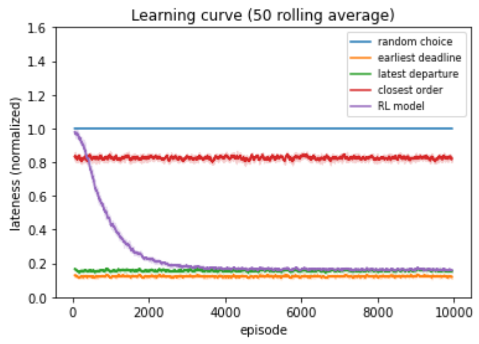
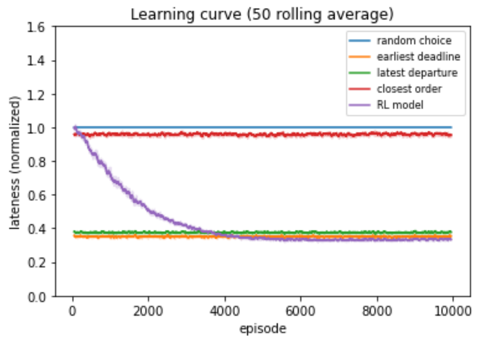
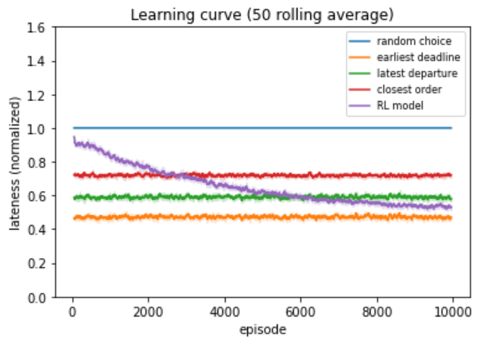
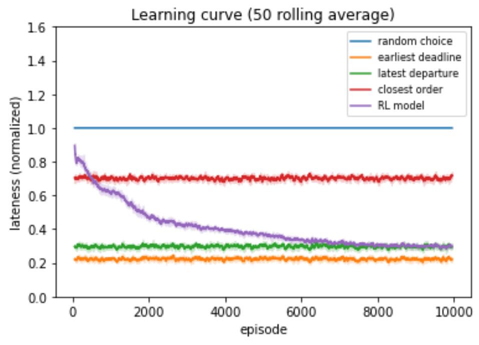
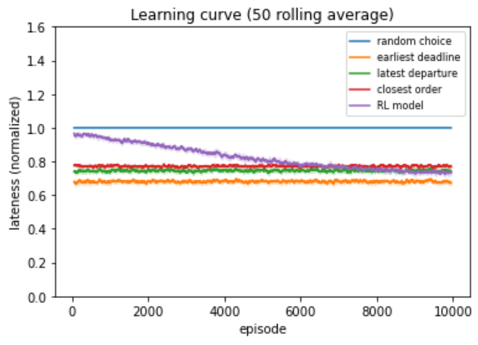
OnePass				Type data	
				Synthetic	Real
Time window	True	Fixed inland terminal	True		
			False		
Time window	False	Fixed inland terminal	True		
			False		

Figure 26 Performance of the OnePass model on alternative scenarios where time windows and inland terminals are relaxed.

Iterative model

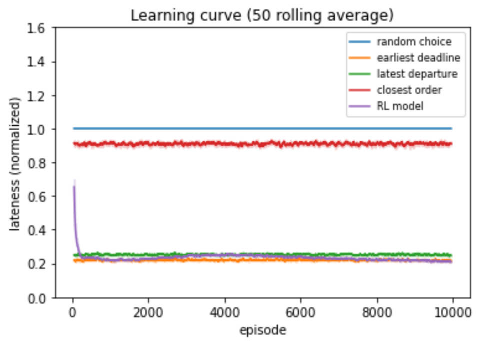
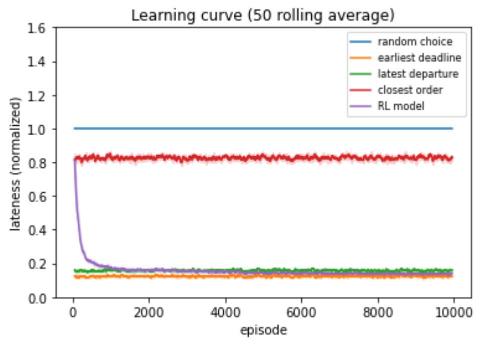
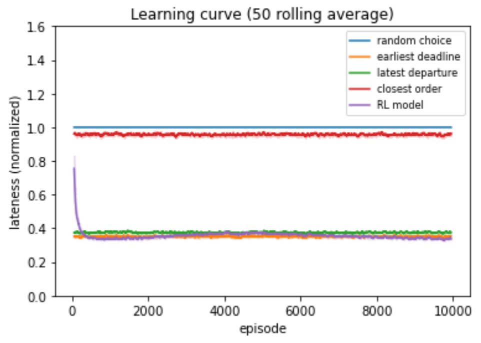
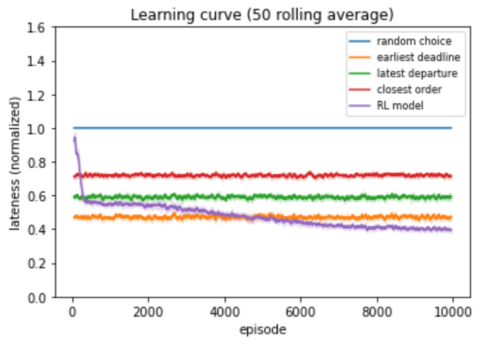
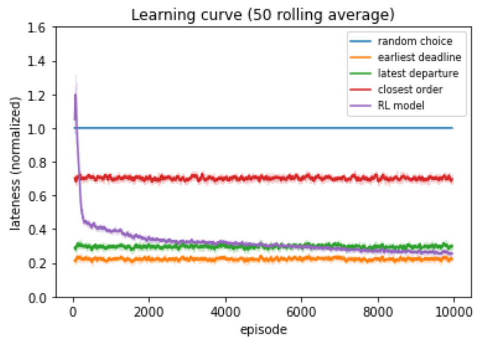
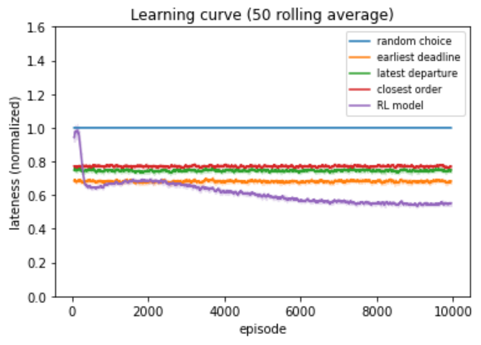
Iterative				Type data	
				Synthetic	Real
Time window	True	Fixed inland terminal	True		
			False		
Time window	False	Fixed inland terminal	True		
			False		

Figure 27 Performance of the Iterative model on alternative scenarios where time windows and inland terminals are relaxed.

Attention model

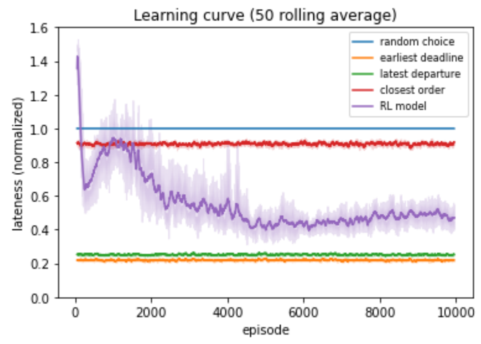
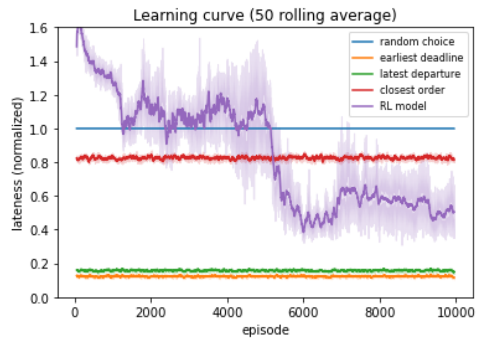
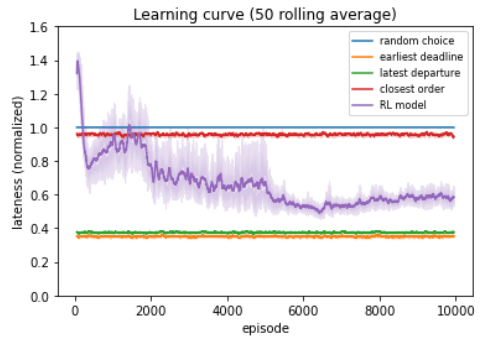
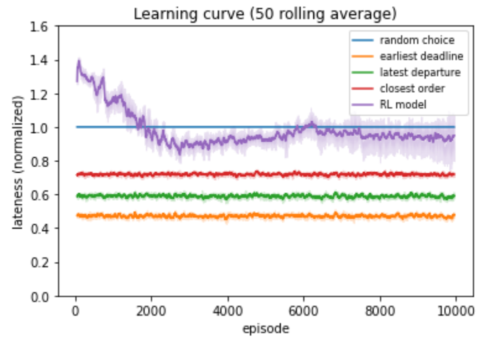
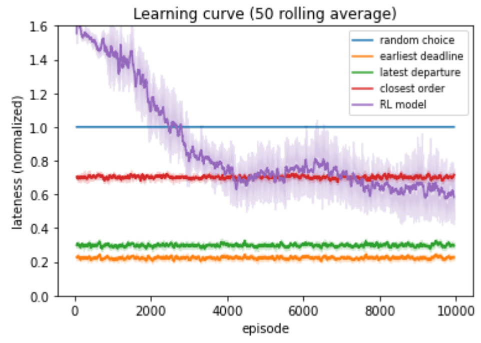
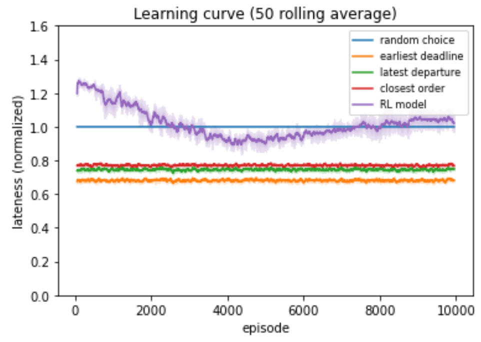
Attention				Type data	
				Synthetic	Real
Time window	True	Fixed inland terminal	True		
			False		
Time window	False	Fixed inland terminal	True		
			False		

Figure 28 Performance of the Attention model on alternative scenarios where time windows and inland terminals are relaxed.

Larger problem sizes

Four scenarios of increasing size have been designed to test whether the models scale well with this increased problem size (Table 8). The previous experiments were all performed with Scenario 2. The results of running of the other scenarios are shown in Figure 29, Figure 30 and Figure 31 for the OnePass, Iterative and Attention model respectively.

The OnePass is able to optimize for instances up to the size of Scenario 2. Scenario 3 takes very long to learn and it does not show any learning in Scenario 4. This indicates that this model is less suitable for large instance sizes. The likely reason is due to this model having many inputs and outputs for all the orders at the same time. When not all inputs can be filled, dummy orders need to be put at these inputs. This gives the network a hard time to make proper predictions.

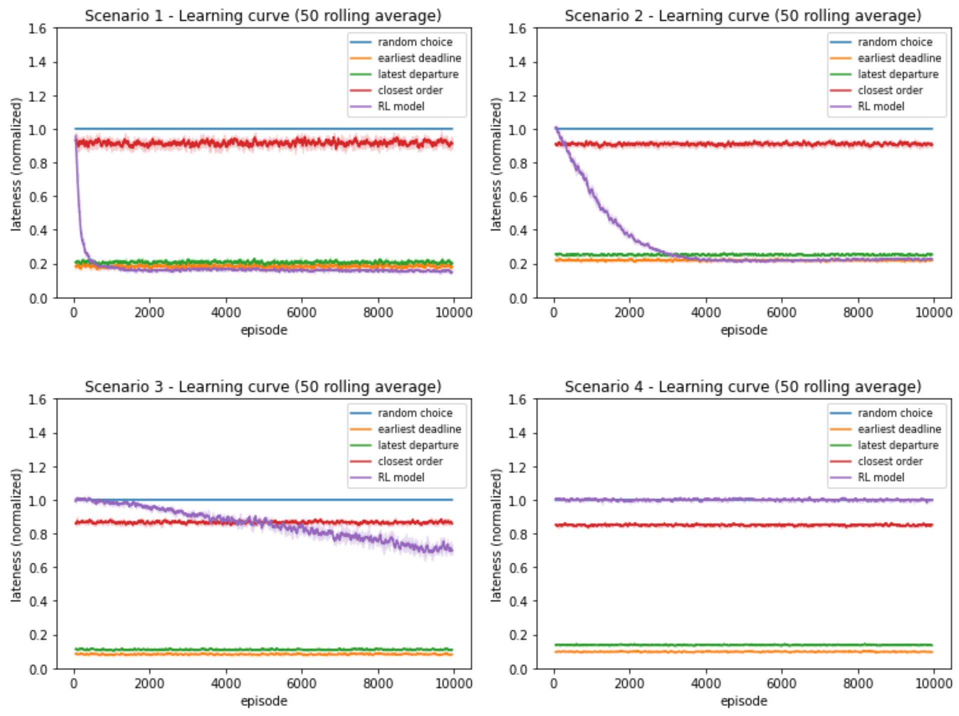


Figure 29 Learning curve for the OnePass model for increasing problem sizes (Scenario 1-4).

The Iterative model shows that it scales better with increasing scenario sizes. For all 4 scenarios it is able to reach a performance similar to the heuristics. The more direct representation of the evaluated order and the Q value as output likely helps the network to generalize better than the OnePass model. However, it should be noted that, due to the iterative way the orders are evaluated, episodes can take quite long to perform as more than one pass through the network are needed. Episode durations 10 times as long compared to the other models have been observed.

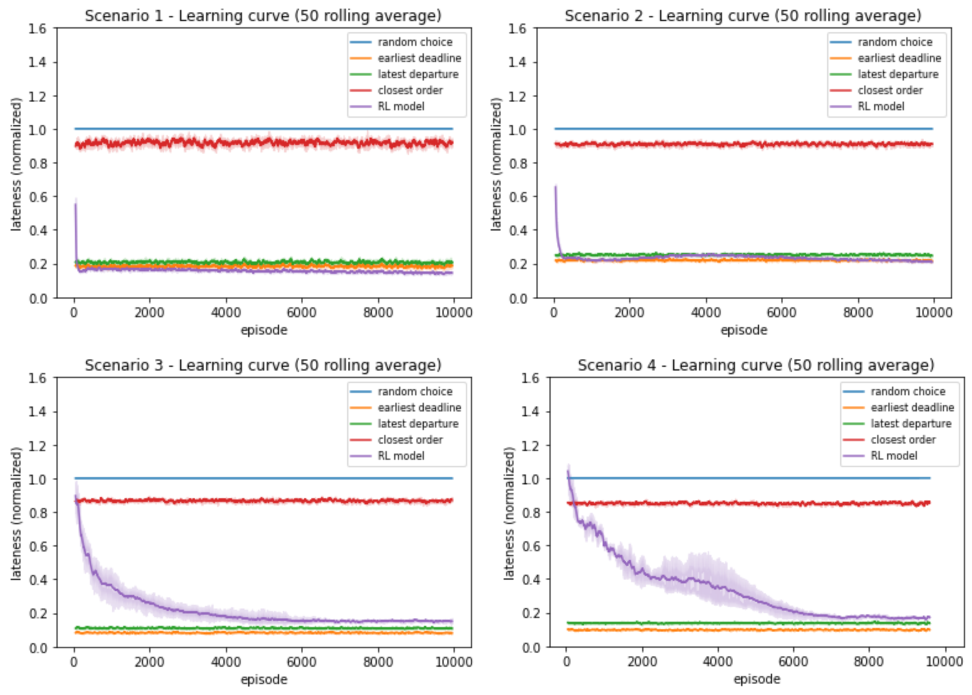


Figure 30 Learning curve for the Iterative model for increasing problem sizes (Scenario 1-4).

The Attention model is able to scale with an increasing scenario size and shows better scaling behaviour than the OnePass network, but it is not as good as the Iterative model. It shows a better performance than random choice, even for scenarios as big as Scenario 4. The scaling nature of this network with its input size likely facilitated this performance, indicating that there is potential in such approaches which are using attention networks.

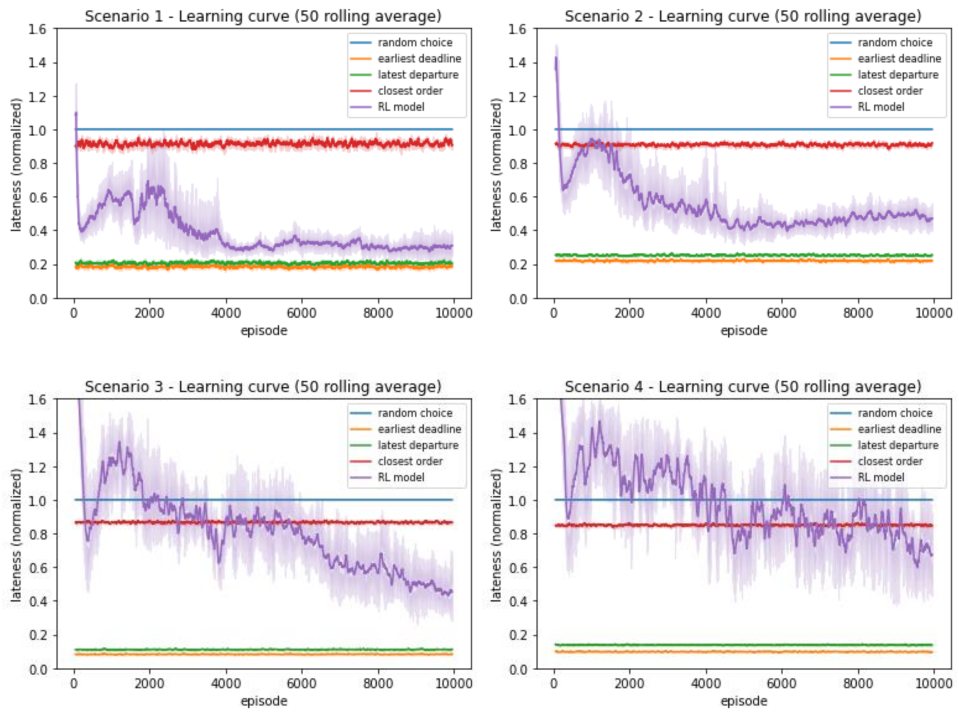


Figure 31 Learning curve for the Attention model for increasing problem sizes (Scenario 1-4).

Transfer learning

To test how well the trained models on synthetic data perform on real data, transfer learning has been tested. In these experiments, we solely train the models on experiences from synthetic data, but evaluate their performance on scenarios consisting of real data as well (Figure 32).

It is observed that the lateness on real data decreases together with the lateness on synthetic data during training. However, the performance does not reach as optimal values as observed in the synthetic data. Furthermore, it is observed that, once the lateness of the synthetic data is not going lower anymore, the lateness on real data is becoming worse. The reason for this is unknown, but it could be that the models are overfitting the synthetic data and generalize less well for real data. Altogether, the concurrent optimization of both types of data indicates that transfer learning could be applied in this problem setting.

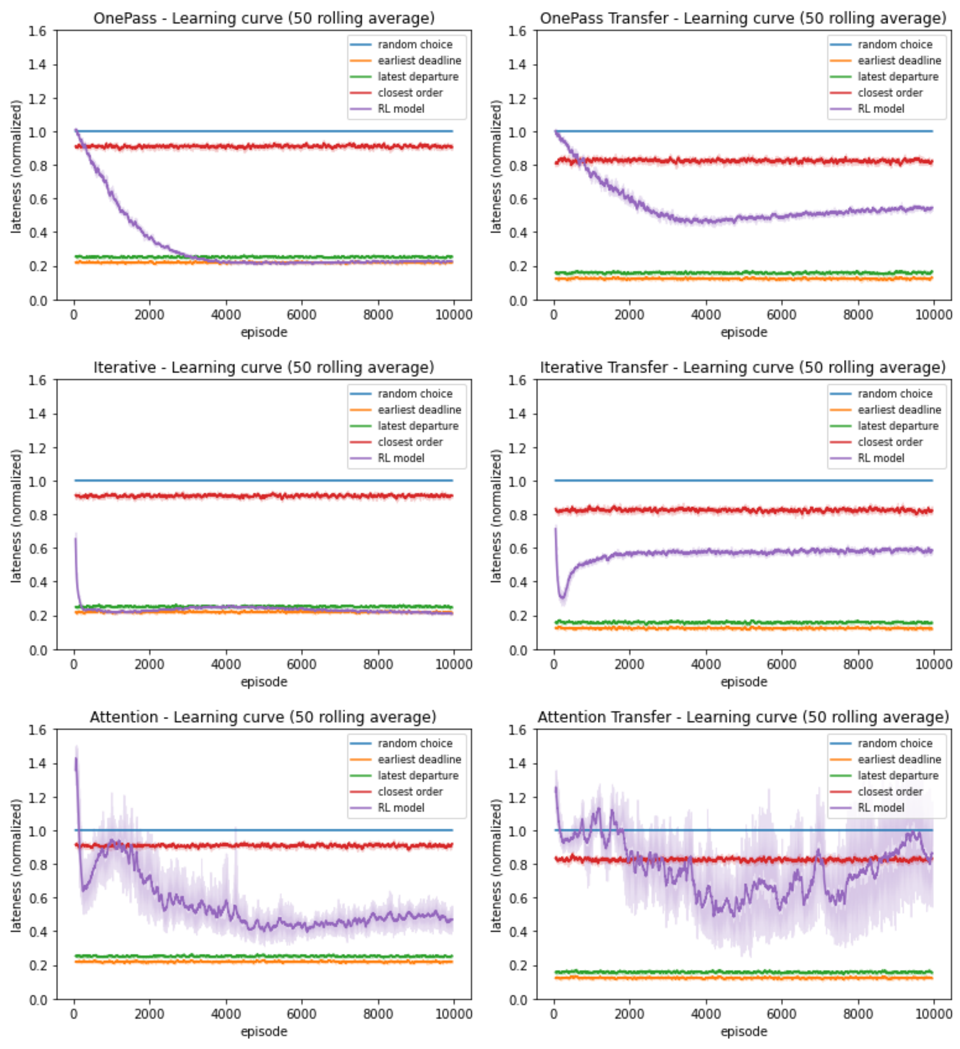


Figure 32 Transfer learning measured for the different models. Left are the learning curves for synthetic data, right are the reinforcement learning models evaluated on real historic order data.

Concluding remarks

Of the three reinforcement learning models, the Iterative model was found to give the best performance. The Iterative model is also able to beat the heuristics in scenarios wherein time windows are more relaxed. In comparison to the other models, an episode for the Iterative model takes quite a bit longer due to the iterative nature of the network. The Attention model is able to learn the environments as well, but tends to achieve less performance and is also less stable than the other models. Additionally, in contrast to the other models, the Attention model does not show any improvement in total driven distance.

When the problem is scaled up to larger instances, it was found that the OnePass model is not able to learn the environment anymore. The Iterative model is able to scale to larger instances in observed performance, however it takes quite long to train. In contrast, the Attention model shows that is able to scale to larger instances and does not take as long to train.

Chapter 8. Conclusions and Recommendations

The aim of this thesis was to identify the advantages of reinforcement learning for order distribution in self-organizing logistics. Firstly, a simulation environment for truck container logistics has been designed (Subquestion 1). This simulator is based on historic data extracts of van Berkel Logistics and made as accurate as reasonably possible. The trucks can be of one of four truck types and are in this way restricted to which orders can be done by those trucks. Additional dynamics like live-loading, decoupling and craning of containers are incorporated as these are key aspects in real planning as well. To enable learning from this environment, three different Deep Q learning models have been developed (Subquestion 2); the OnePass, Iterative and Attention model. In the design of these models the variable state- and action space due to a changing number of orders have been taken into account (Subquestion 3). It was found that shifting the features with respect to the choosing truck helped in learning the environment. The reward function in the environment could solely be based on the lateness of the order (Subquestion 4). To make a better connection between the chosen order and observed lateness, the lateness of the orders which were already running late was included in the form of split lateness. This was effective for learning the environment.

The heuristic decision mechanisms the models were compared against worked already well in planning orders (Subquestion 5). Among the models, both the OnePass as well as the Iterative model were able to learn the environment to the degree such that it performs similar to the earliest deadline heuristic when concerning total lateness. In addition, it was found that these models perform better in terms of driven distance and fraction of on time orders. In alternative scenarios without time windows, the Iterative model has shown that it can beat the heuristics in terms of lateness as well. The Attention model showed that it can learn to reduce lateness to some extent, but tends to be less stable during training. It was not be able to beat the earliest deadline heuristic in our test cases.

When scale up analysis was performed, both the Iterative and Attention model were able to scale up to bigger instances without losing much performance. The OnePass model was not able to learn in scenarios as big as occur in real days. The Iterative model however, takes considerably longer for training as it evaluates all orders one at the time before making a decision. The Attention model might scale better, and might also learn the environment better if other hyperparameters for the model are used. To our insight there is quite some potential for improvement for this model with further tweaking.

Altogether, this thesis has shown that reinforcement learning can help in making a step towards a more self-thinking automated logistical supply chain. However, due to the simplifications made and the near-heuristic performance achieved, this approach still needs further development. One could look into training separate networks for the different truck types as in reality their drivers make their decisions differently as well. Another logical step is to shift the focus from Q learning and look into whether other approaches like actor-critic approaches yield better performance. The actor-critic architecture, uses two deep neural networks (actor and critic) of which the actor predicts which action to take and the critic evaluates the value of that action [8]. Furthermore, the permutation invariance of the state space can be addressed further; An additional regularization factor in the training scheme might help in achieving a better performance and has not been tried out yet within this thesis.

Instead of using reinforcement learning for planning out a whole day of container logistics, one can also look at using reinforcement learning for solving local logistical disruptions. During the day, when trucks are delayed or ad-hoc orders need to be done, quick decisions need to be made. These decisions tend to be local and are therefore not prone to a large action space and might therefore be easier automatically solved via reinforcement learning than other methods.

When looking at the future for self-organizing logistics, we might need to shift our focus more to multi-agent settings wherein different agents work together without central coordination for a good supply chain. There have not been many real-life successes with using reinforcement learning for many agents cooperating in logistical settings. This might be due to that with reinforcement learning the choices are made by a neural network and can therefore not be easily explained to humans. When a reinforcement learning choice is not logical to a planner, it might be easily decided to override the action, limiting its effectiveness.

A more technical reason might be that it is infeasible to train all agents individually. A promising approach to tackle this is to do central training and decentral execution. The actor-critic based counterfactual multi-agent (COMA) architecture shows that such approach is feasible [40]. Additionally, there have been several variants of DQN for a multi-agent setting. Nguyen et al. [10] reviewed these approaches and the Deep repeated update Q network (DRUQN) and Deep loosely coupled Q network (DLCQN) seem most applicable.

In a decentral multi-agent setting, the agents are concurrently learning the environment and therefore the next state is not only dependent on the actions of an individual agent, but also on the actions of the others. The actions of the other agents make the environment stochastic and more difficult to learn [41]. Furthermore, every agent will have a local view of their environment which makes the decision of the action for the highest expected reward harder to determine [42]. A key challenge will be to design the system such that the agents can work with just local information while optimizing the system as a whole. When going decentral, a global reward function might not be easily measured and therefore achieved. In addition, there will be a challenge in how and what information the trucks will communicate with each other and how this information is processed. When each agent is learning to optimize their individual goals, the system might end up in a suboptimal local equilibrium. In such equilibria, changing an action of an individual agent will not be beneficial due to the decisions of other agents. Such equilibria are depicted as Nash equilibria in game theory of which the Prisoners Dilemma is one of the most famous examples. In logistics this has also been observed in the form of the Wardrop equilibrium [43] and Braess paradox [44] where the best actions of individual drivers do not result in a global optimum of the system.

Chapter 9. References

[1] M. Janic, "Modelling the full costs of an intermodal and road freight transport network," *Transp. Res. Part D Transp. Environ.*, vol. 12, no. 1, pp. 33–44, 2007.

[2] W. J. van Schijndel and J. Dinwoodie, "Congestion and multimodal transport: A survey of cargo transport operators in the Netherlands," *Transp. Policy*, vol. 7, no. 4, pp. 231–241, 2000.

[3] B. Montreuil, "Toward a Physical Internet: meeting the global logistics sustainability grand challenge," *Logist. Res.*, vol. 3, no. 2–3, pp. 71–87, 2011.

[4] S. Pan, D. Trentesaux, and Y. Sallez, "Specifying self-organising logistics system: Openness, intelligence, and decentralised control," *Stud. Comput. Intell.*, vol. 694, pp. 93–102, 2017.

[5] M. Frisk, M. Göthe-Lundgren, K. Jörnsten, and M. Rönnqvist, "Cost allocation in collaborative forest transportation," *Eur. J. Oper. Res.*, vol. 205, no. 2, pp. 448–458, 2010.

[6] C. R. van Ommeren, R. W. Fransen, G. L. J. Pingen, C. J. van Leeuwen, J. P. Paardekooper, and J. C. van Meijeren, "Exploring Possibilities of Letting Vehicles Plan and Organise Transportation Themselves," *https://www.tno.nl/en/focus-areas/traffic-transport/roadmaps/smart-and-safe-traffic-and-transport/smart-mobility-and-logistics/first-steps-towards-self-organizing-logistics/pilot-autonomous-algorithms/*.

[7] C. J. Van Leeuwen and P. Pawełczak, "CoCoA: A non-iterative approach to a local search (A)DCOP Solver," *31st AAAI Conf. Artif. Intell. AAAI 2017*, pp. 3944–3950, 2017.

[8] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*. MIT press, 2020.

[9] V. Mnih *et al.*, "Playing Atari with Deep Reinforcement Learning," pp. 1–9, 2013.

[10] T. T. Nguyen, N. D. Nguyen, and S. Nahavandi, "Deep Reinforcement Learning for Multiagent Systems: A Review of Challenges, Solutions, and Applications," *IEEE Trans. Cybern.*, vol. 50, no. 9, pp. 3826–3839, 2020.

[11] D. Silver *et al.*, "A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play," *Science (80-.).*, vol. 362, no. 6419, pp. 1140–1144, 2018.

[12] R. Bellman, "A Markovian Decision Process," *Indiana Univ. Math. J.*, vol. 6, no. 4, pp. 679–684, 1957.

[13] R. S. Sutton, D. Precup, and S. Singh, "Between MDPs and Semi-MDPs: A Framework for Temporal Abstraction in RL," *Statew. Agric. L. Use Baseline 2015*, vol. 1, no. 1, pp. 181–211, 1999.

[14] C. J. C. H. Watkins and P. Dayan, "Q-Learning," *Mach. Learn.*, 1992.

[15] C. J. C. H. Watkins, "Learning from delayed rewards," 1989.

[16] H. Van Hasselt, A. Guez, and D. Silver, "Deep reinforcement learning with double Q-Learning," *30th AAAI Conf. Artif. Intell. AAAI 2016*, pp. 2094–2100, 2016.

[17] N. S. Vaswani, Ashish *et al.*, "Attention Is All You Need," *Adv. Neural Inf. Process. Syst.*, vol. 30, no. Nips, pp. 5998–6008, 2017.

[18] R. V. Kulkarni and P. R. Bhave, "Integer programming formulations of vehicle routing problems," *Eur. J. Oper. Res.*, vol. 20, no. 1, pp. 58–67, 1985.

[19] A. Agra, M. Christiansen, R. Figueiredo, L. M. Hvattum, M. Poss, and C. Requejo, “The robust vehicle routing problem with time windows,” *Comput. Oper. Res.*, vol. 40, no. 3, pp. 856–866, Mar. 2013.

[20] M. Hoogeboom, Y. Adulyasak, W. Dullaert, and P. Jaillet, “The robust vehicle routing problem with time window assignments,” *Transp. Sci.*, vol. 55, no. 2, pp. 395–413, 2021.

[21] M. Nazari, A. Oroojlooy, M. Takáč, and L. V. Snyder, “Reinforcement learning for solving the vehicle routing problem,” *Adv. Neural Inf. Process. Syst.*, vol. 2018-Decem, pp. 9839–9849, 2018.

[22] W. Kool, H. Van Hoof, and M. Welling, “Attention, learn to solve routing problems!,” *7th Int. Conf. Learn. Represent. ICLR 2019*, pp. 1–25, 2019.

[23] Y. Hu, Y. Yao, and W. S. Lee, “A reinforcement learning approach for optimizing multiple traveling salesman problems over graphs,” *Knowledge-Based Syst.*, vol. 204, p. 106244, 2020.

[24] N. Sultana, J. Chan, A. K. Qin, and T. Sarwar, “Learning to Optimise General TSP Instances,” 2020.

[25] N. Sultana, J. Chan, A. K. Qin, and T. Sarwar, “Learning Vehicle Routing Problems using Policy Optimisation,” pp. 1–16, 2020.

[26] K. Lin, R. Zhao, Z. Xu, and J. Zhou, “Efficient Collaborative Multi-Agent Deep Reinforcement Learning for Large-Scale Fleet Management,” *arXiv Prepr.*, vol. 1802.06444, 2018.

[27] Z. Xu *et al.*, “Large-scale order dispatch in on-demand ride-hailing platforms: A learning and planning approach,” *Proc. ACM SIGKDD Int. Conf. Knowl. Discov. Data Min.*, pp. 905–913, 2018.

[28] X. Tang *et al.*, “A deep value-network based approach for multi-driver order dispatching,” *Proc. ACM SIGKDD Int. Conf. Knowl. Discov. Data Min.*, pp. 1780–1790, 2019.

[29] N. N. Sultana, V. Baniwal, A. Basumatary, P. Mittal, S. Ghosh, and H. Khadilkar, “Fast Approximate Solutions using Reinforcement Learning for Dynamic Capacitated Vehicle Routing with Time Windows,” 2021.

[30] X. Li, J. Zhang, J. Bian, Y. Tong, and T. Y. Liu, “A cooperative multi-agent reinforcement learning framework for resource balancing in complex logistics network,” *Proc. Int. Jt. Conf. Auton. Agents Multiagent Syst. AAMAS*, vol. 2, pp. 980–988, 2019.

[31] C. van Ommeren, L. Oudenes, J. van Meijeren, and J. de Bes, “Waarde van Data,” *https://repository.tno.nl/islandora/object/uuid%3Aa358735f-0aa1-4b1e-99af-09622284e128*, 2020.

[32] C. Boutilier *et al.*, “Planning and learning with stochastic action sets,” *IJCAI Int. Jt. Conf. Artif. Intell.*, vol. 2018-July, pp. 4674–4682, 2018.

[33] G. Dulac-Arnold *et al.*, “Deep Reinforcement Learning in Large Discrete Action Spaces,” 2015.

[34] V. Kumar, S. Bhambri, and P. G. Shambharkar, “Multiple Resource Management and Burst Time Prediction using Deep Reinforcement Learning,” *Proc. Eighth Intl. Conf. Adv. Comput. Commun. Inf. Technol. - CCIT 2019*, vol. 9, no. 2, pp. 51–58, 2019.

[35] B. Zoph and Q. V Le, “Searching for activation functions,” *6th Int. Conf. Learn. Represent. ICLR 2018 - Work. Track Proc.*, pp. 1–13, 2018.

[36] R. Girshick, “Fast R-CNN,” *Proc. IEEE Int. Conf. Comput. Vis.*, vol. 2015 Inter, pp. 1440–1448, 2015.

[37] G. E. Dahl, T. N. Sainath, and G. E. Hinton, “Improving deep neural networks for LVCSR using rectified linear units and dropout,” *ICASSP, IEEE Int. Conf. Acoust. Speech Signal Process. - Proc.*, no. 2010, pp. 8609–8613, 2013.

[38] T. Tieleman and G. Hinton, “Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude,” *COURSERA Neural networks Mach. Learn.*, vol. 4, no. 2, pp. 26–31, 2012.

[39] D. P. Kingma and J. L. Ba, “Adam: A method for stochastic optimization,” *3rd Int. Conf. Learn. Represent. ICLR 2015 - Conf. Track Proc.*, pp. 1–15, 2015.

[40] J. N. Foerster, G. Farquhar, T. Afouras, N. Nardelli, and S. Whiteson, “Counterfactual multi-agent policy gradients,” *32nd AAAI Conf. Artif. Intell. AAAI 2018*, pp. 2974–2982, 2018.

[41] P. Hernandez-Leal, M. Kaisers, T. Baarslag, and E. M. de Cote, “A Survey of Learning in Multiagent Environments: Dealing with Non-Stationarity,” *arXiv Prepr.*, vol. 1707.09183, 2017.

[42] S. Omidshafiei, J. Pazis, C. Amato, J. P. How, and J. Vian, “Deep decentralized multi-task multi-agent reinforcement learning under partial observability,” *34th Int. Conf. Mach. Learn. ICML 2017*, vol. 6, pp. 4108–4122, 2017.

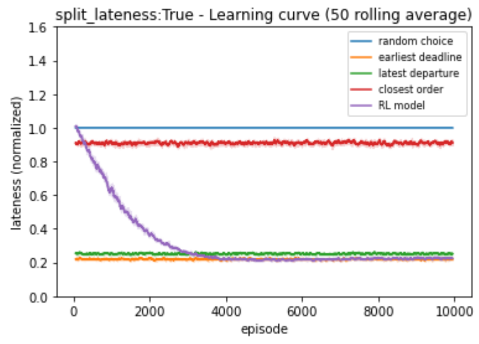
[43] J. G. Wardrop and J. I. Whitehead, “Correspondence. Some theoretical aspects of road traffic research.,” *Proc. Inst. Civ. Eng.*, vol. 1, no. 5, pp. 767–768, 1952.

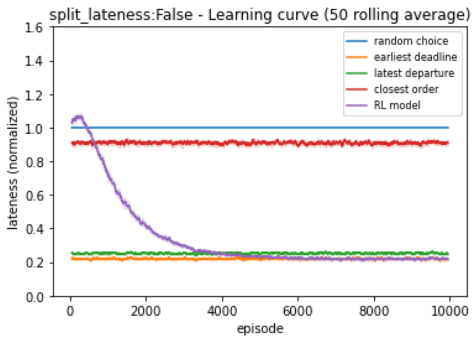
[44] D. Braess, “Uber ein paradoxen der verkehrsplanung,” *Unternehmensforschung*, vol. 12, pp. 258–268, 1968.

Chapter 10. Appendices

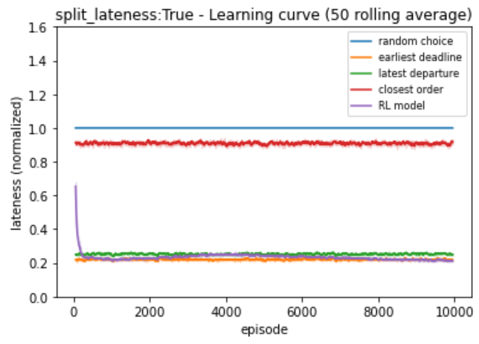
Appendix A - Split lateness

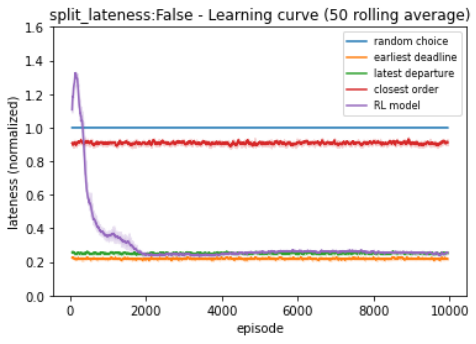
OnePass



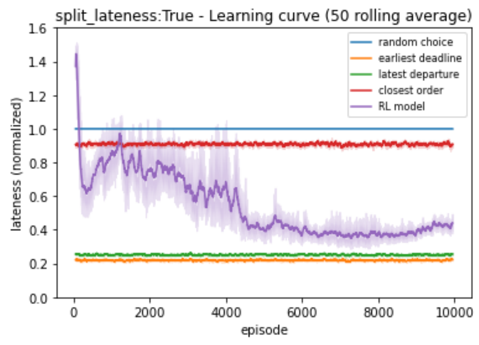


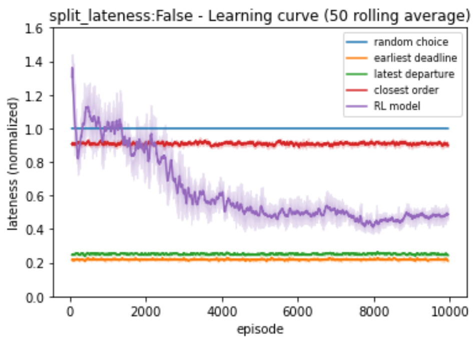
Iterative





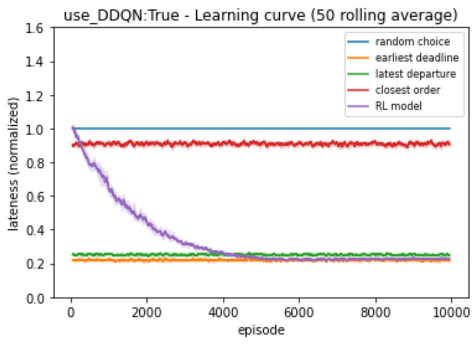
Attention

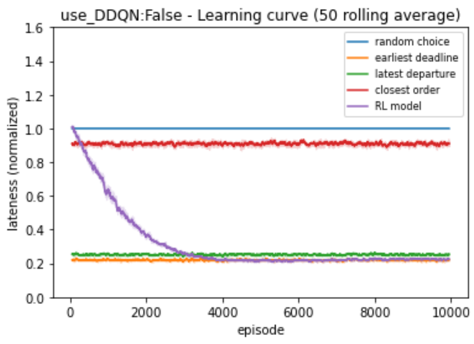




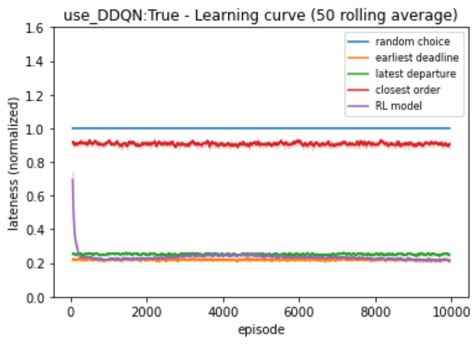
Appendix B - DDQN

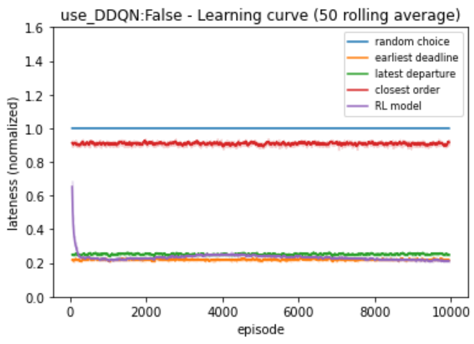
OnePass



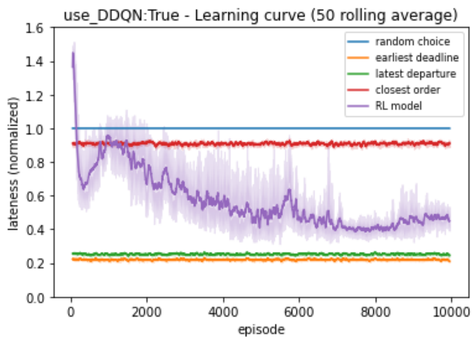


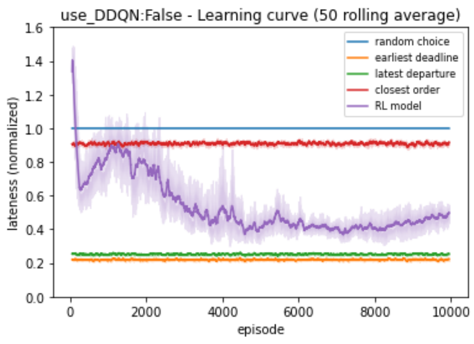
Iterative





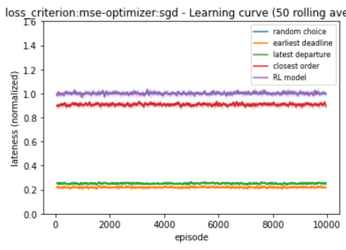
Attention

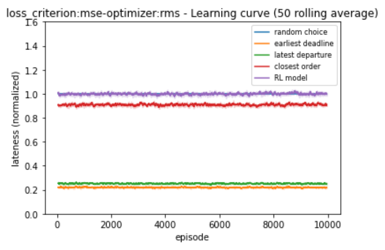


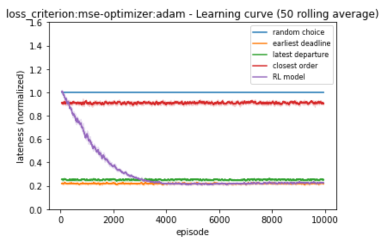


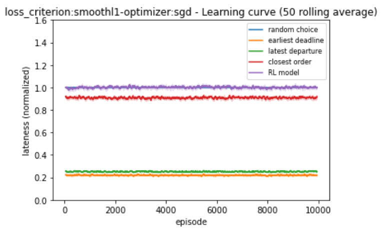
Appendix C - OnePass model hyperparameters

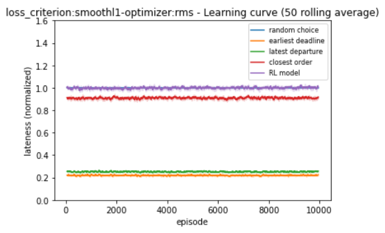
Loss criterion / optimizer

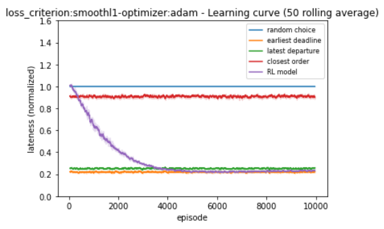




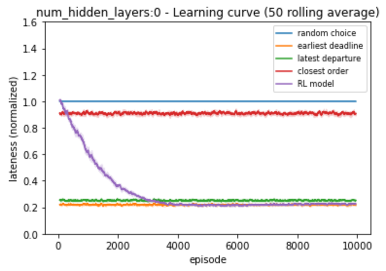


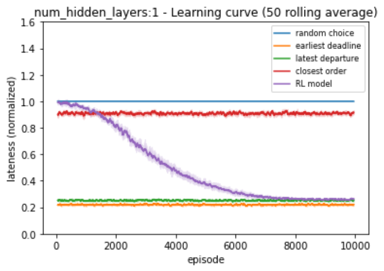


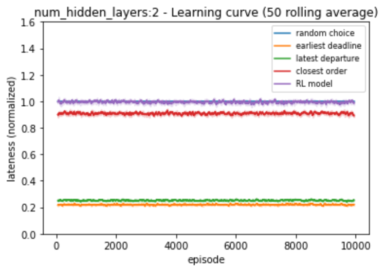




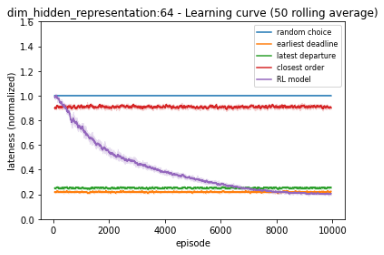
Number of hidden layers

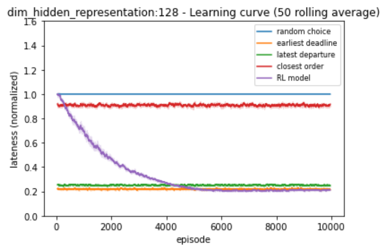


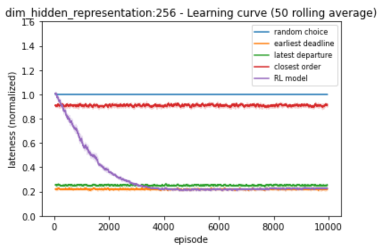


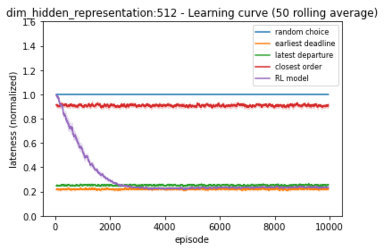


Dimension hidden representation

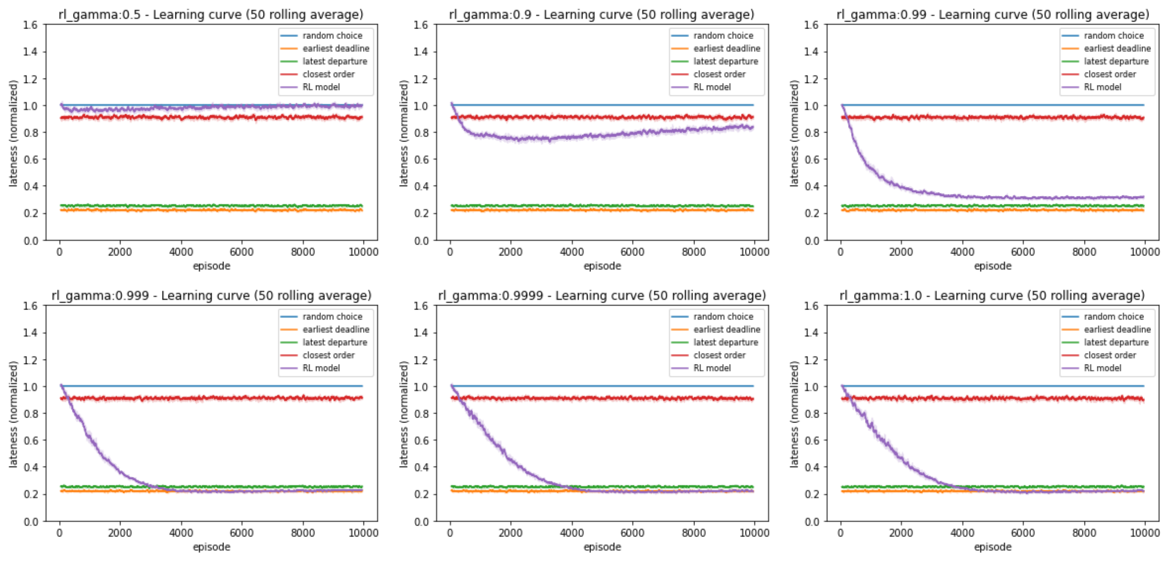




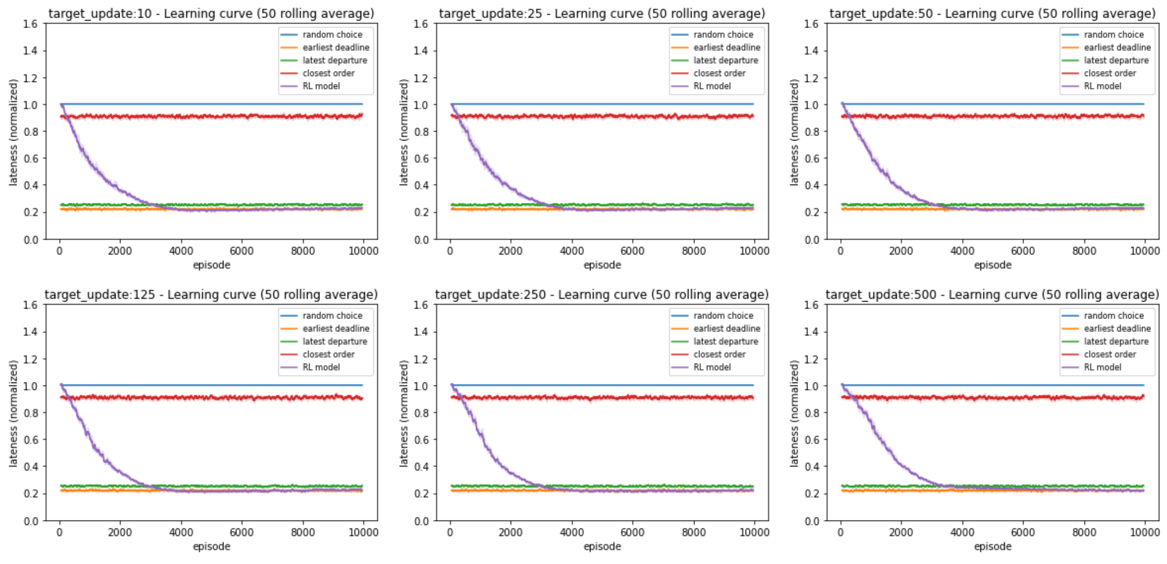




Gamma

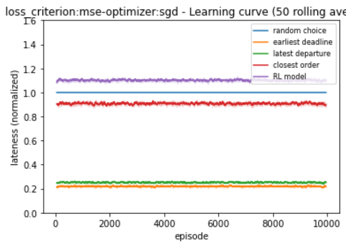


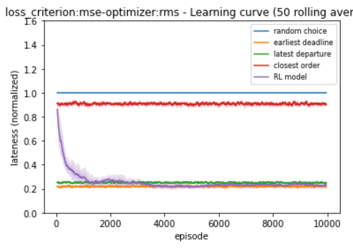
Target network update frequency

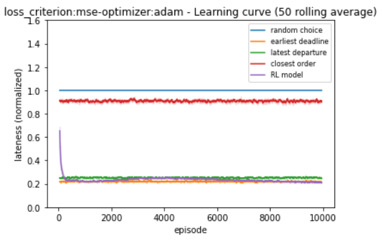


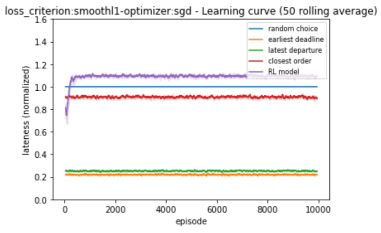
Appendix D - Iterative model hyperparameters

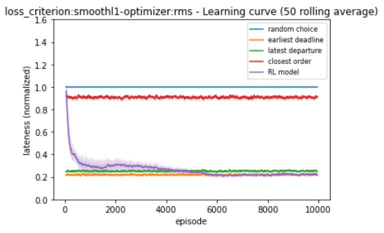
Loss criterion / optimizer

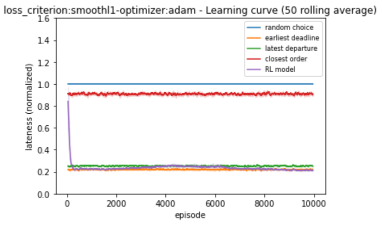




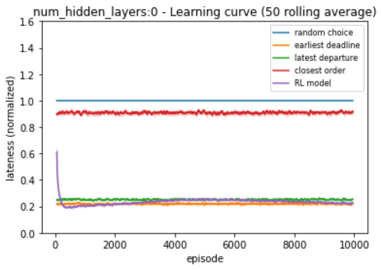


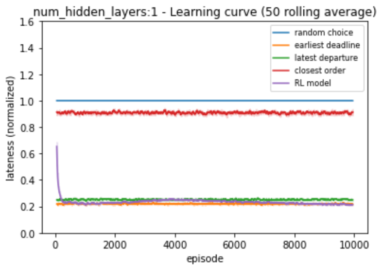


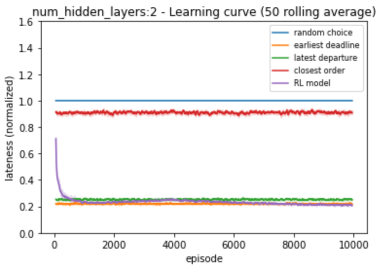




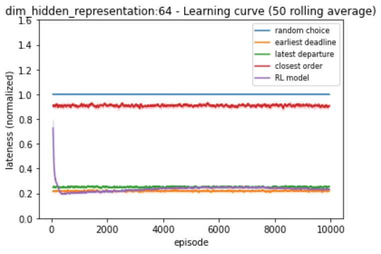
Number of hidden layers

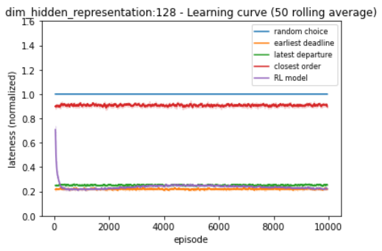


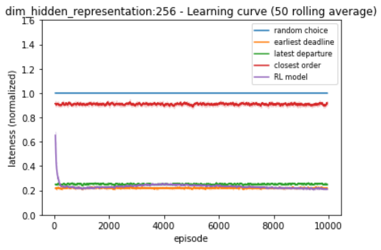


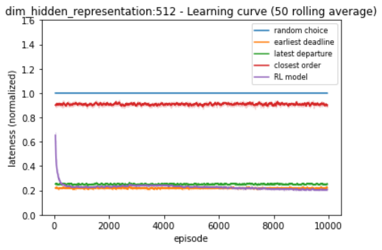


Dimension hidden representation

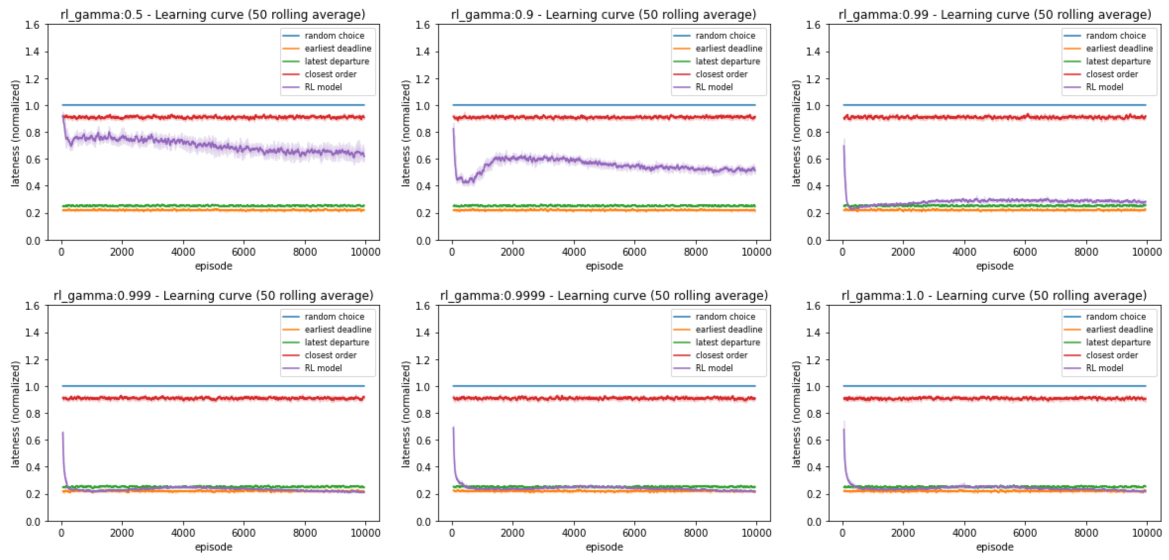




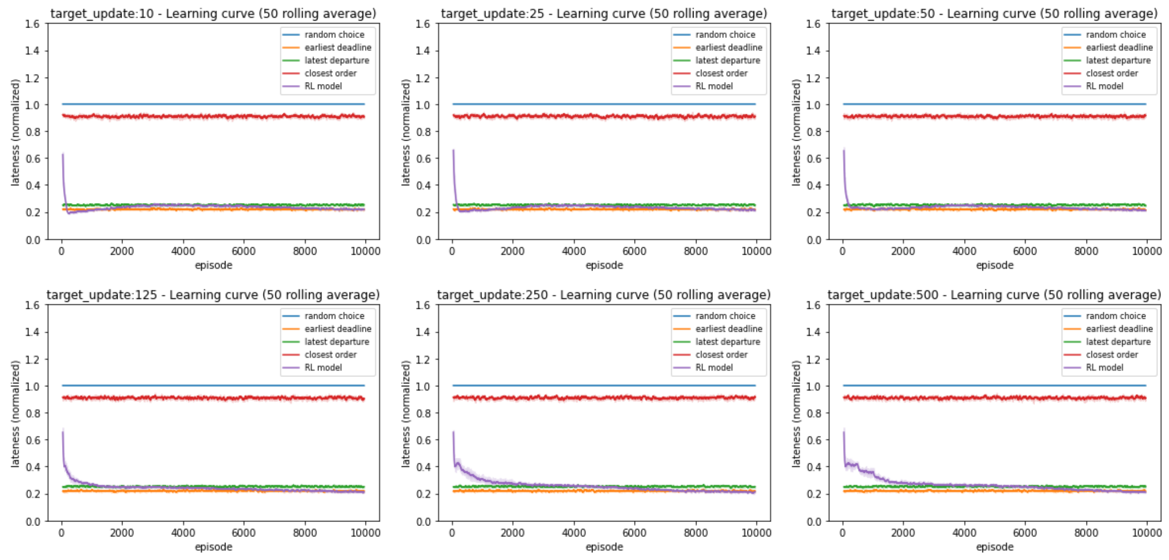




Gamma

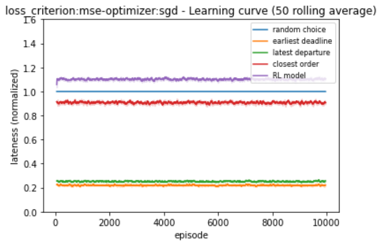


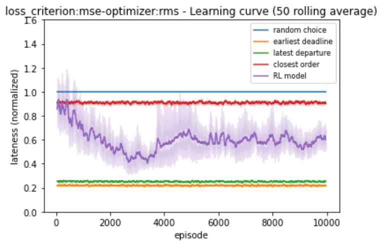
Target network update frequency

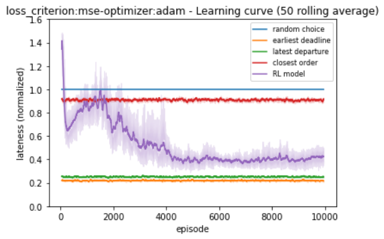


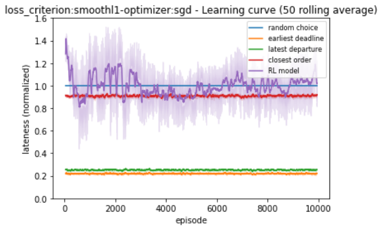
Appendix E - Attention model hyperparameters

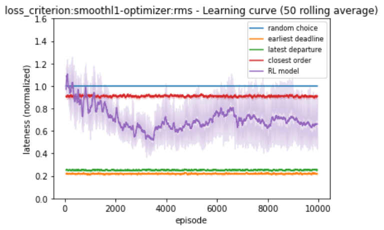
Loss criterion / optimizer

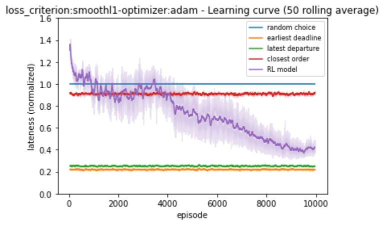




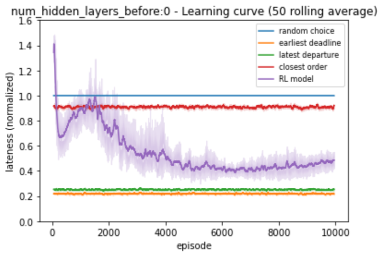


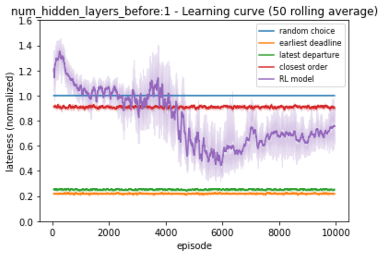




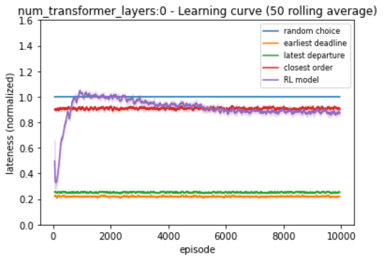


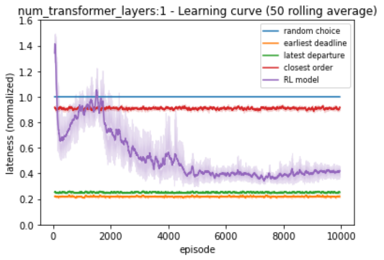
Number of hidden layers before the transformer encoder layers

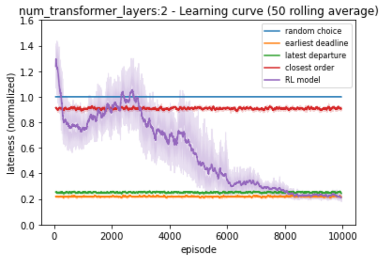




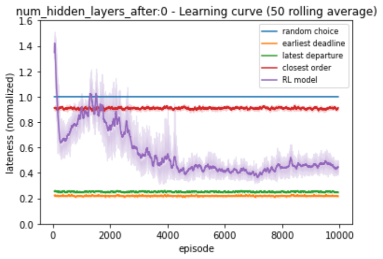
Number of transformer encoder layers

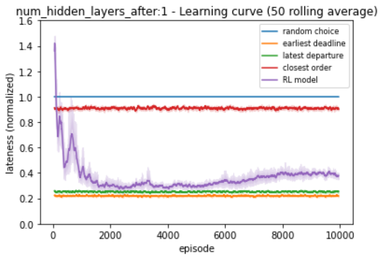




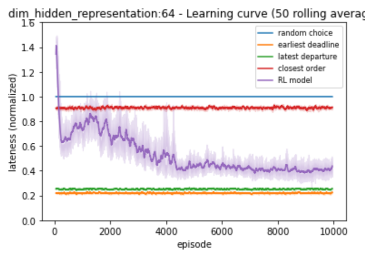


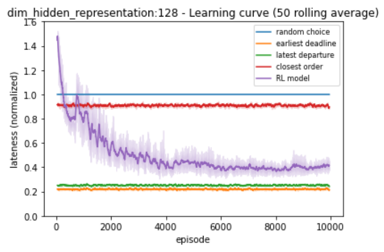
Number of hidden layers after the transformer encoder layers

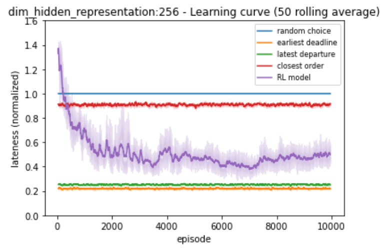


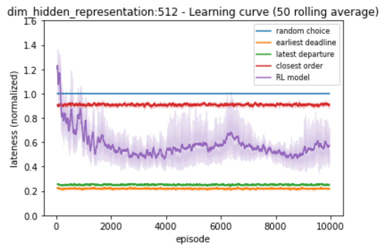


Dimension hidden representation

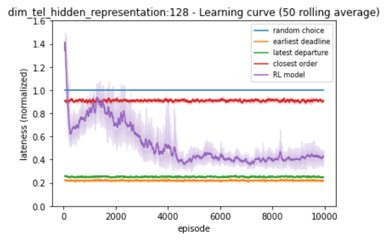


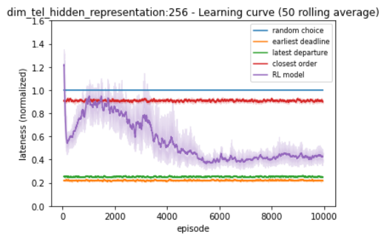


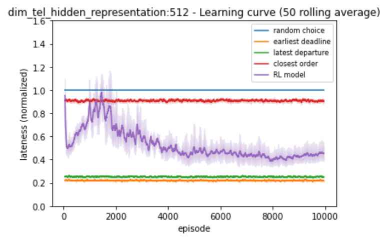




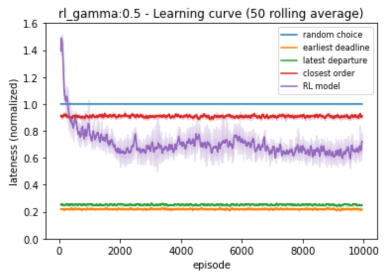
Dimension transformer encoder layer hidden representation

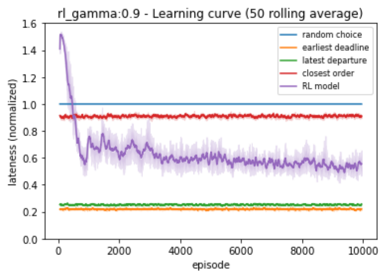


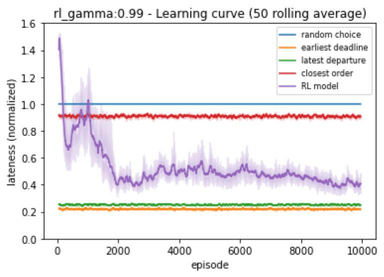


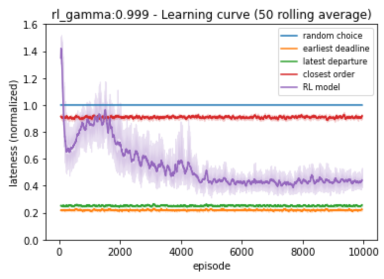


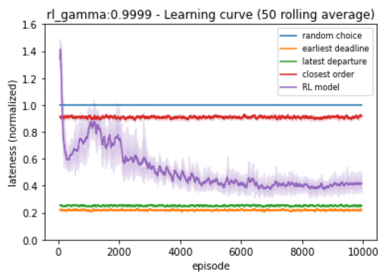
Gamma

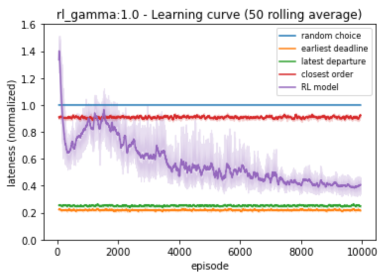




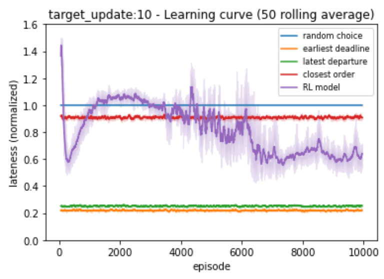


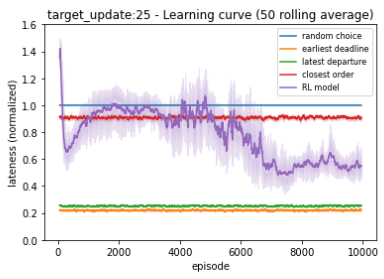


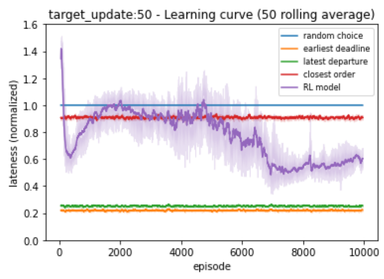


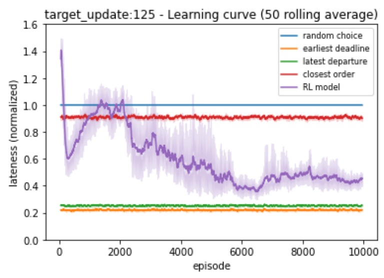


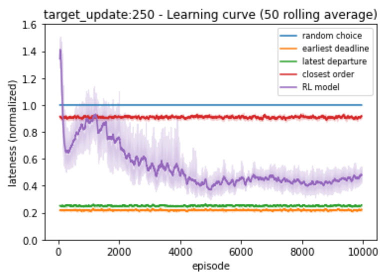
Target network update frequency

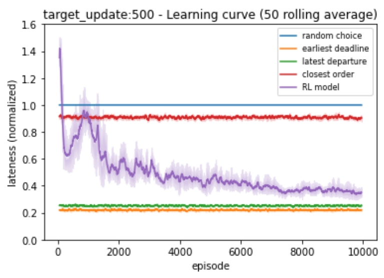

target_update:10 - Learning curve (50 rolling average)
lateness (normalized)
episode
random choice
earliest deadline
latest departure
closest order
RL model


target_update:25 - Learning curve (50 rolling average)
lateness (normalized)
episode
random choice
earliest deadline
latest departure
closest order
RL model


target_update:50 - Learning curve (50 rolling average)
lateness (normalized)
episode
random choice
earliest deadline
latest departure
closest order
RL model


target_update:125 - Learning curve (50 rolling average)
lateness (normalized)
episode
random choice
earliest deadline
latest departure
closest order
RL model


target_update:250 - Learning curve (50 rolling average)
lateness (normalized)
episode
random choice
earliest deadline
latest departure
closest order
RL model


target_update:500 - Learning curve (50 rolling average)
lateness (normalized)
episode
random choice
earliest deadline
latest departure
closest order
RL model