Combining a Discrete-event Simulation Model of a Logistics Network with Deep Reinforcement Learning

Markus Rabe[1], Felix Dross[2], Alexander Wuttke[3]

[1, 3] Technical University Dortmund
Department IT in Production and Logistics
Leonhard-Euler-Straße 5, 44227 Dortmund, Germany
markus.rabe@tu-dortmund.de, alexander2.wuttke@tu-dortmund.de

[2] Technical University Dortmund
Graduate School of Logistics
Leonhard-Euler-Straße 5, 44227 Dortmund, Germany
felix.dross@tu-dortmund.de

Abstract

This paper presents a method to combine a discrete-event simulation model of a logistics network with a deep reinforcement learning agent. The agent applies different actions to the logistics network in order to learn a strategy to improve the cost structure and the performance of the logistics network. The return for the agent is derived from the costs and the performance of the logistics network which is measured in terms of logistics costs and β service level. Possible actions comprise the relocation of inventory and the adjustment of transport relations. The authors developed a method to represent the state of the simulation model with an image generated from the simulation input data and used a well-known deep Q-learning algorithm with a convolutional neural network to train the agent.

1 Introduction

Large logistics networks are complex systems that are very hard to manage. To cope with the complexity of logistics networks, several specific performance measurement systems with key performance indicators (KPIs) have been developed to provide managers with the background information they need to improve their area of accountability [1, 4, 5]. Besides specific KPIs, the systems usually also provide catalogues of possible actions to ease the decisions about the right corrective actions in the logistics network. Examples for such actions could be the relocation of stock from one site to another or the adjustment of transport relations within the network. Unfortunately though, the effects of all the actions in the catalogues and their interdependencies are very hard to predict for the managers. In many situations, they are overwhelmed with all the possibilities and uncertain about the right actions to take. Therefore, especially trading businesses are demanding for better solutions to plan their actions in the logistics network. For this research, the authors are cooperating with an international trading company with over 100 warehouses and an inventory of around 150,000 items on permanent stock.

To solve the described problem, the authors have previously proposed the design of a decision support system which uses a discrete-event simulation (DES) to predict the consequences of actions in the logistics network [6]. In order to offer a realistic prediction, special methods to measure real world data warehouse KPIs on the simulation output data have been developed [20]. The architecture of the system includes a data-driven DES model with an underlying database, so possible actions can be applied automatically to the simulation model in the form of SQL updates. Although already automated, the solution space of all possible combinations of actions is too large to be searched extensively. Therefore, the authors proposed to use a meta-heuristic which automatically applies the possible actions to the simulation model and finds the most promising action sets. This approach has previously been described as a simheuristic [10]. A simheuristic framework for the system has been extensively described in [6].

Continuing the research, the authors propose to use a reinforcement learning agent instead of a classical meta-heuristic. Reinforcement learning in general is targeted towards enabling an agent to learn what to do - how to map situations to actions - so as to maximize a numerical reward signal [27]. Using reinforcement learning, the system is now designed to use its idle time to experiment with the DES model and sequentially learn the most promising sequence of actions. The authors already extensively

described the software architecture for the realization of such a system in [19]. Since then, a corresponding prototype has been developed. Although the general idea is straightforward, the biggest challenge is to find the appropriate reinforcement learning method and a state representation for the simulation model to train a reinforcement learning agent with. The approach presented in this paper is inspired by recent research regarding the Arcade Learning Environment (ALE) [2] and deep reinforcement learning.

The ALE provides an interface to hundreds of Atari 2600 game environments, each one different and designed to be a challenge for humans [2]. It therefore offers an environment for testing reinforcement learning agents. Breakthroughs regarding the development of an agent that teaches itself to play classic Atari games in the ALE by looking at pixels and learning actions that increase the game score were presented by Mnih et al. in 2013 [16]. Mnih et al. showed that their algorithm outperformed all previous approaches on six of seven tested games and surpassed a human expert on three of them. Following up on these results, Mnih et al. published a widely recognized article in 2015, where their deep reinforcement learning agent was able to surpass the performance of all previously presented algorithms and achieved a level comparable to that of a professional human games tester across a set of 49 games [17]. The research results were a remarkable example of the progress being made in artificial intelligence [23]. The approach of Mnih et al. combined reinforcement learning with a class of artificial neural networks known as deep neural networks, therefore it is called deep reinforcement learning [7, 15, 17]. The concrete neural network used by Mnih et al. is a convolutional neural network which is trained with a variant of Q-learning, a famous reinforcement learning approach [31]. The input to the network is a state representation and the outputs are estimates of the Q-values for each action. Mnih et al. therefore referred to the approach as a deep Q-network (DQN) [17]. Most interesting for the research presented in this paper is the fact that Mnih et al. showed the first artificial agent that was capable of learning to excel at a diverse array of challenging tasks, using the same algorithm, network architecture and hyperparameters, receiving only the pixels and the game score as inputs.

Inspired by these breakthroughs, the authors decided to design a graphical state representation for the problem described in this paper and let the agent presented by Mnih et al. try to solve the complex task of learning the most promising action sets. The system presented here can automatically apply actions to the database of the discrete-event simulation model and then obtain a reward by running and evaluating the simulation. The graphical state representation is generated from the simulation input data in the database. Finally, after some training, the system should be able to recommend the best action combinations for different logistics network situations.

First promising experiments have been conducted with a small simulation model of a segment of a larger logistics network. On the following pages, the authors present their current state of the research and some preliminary results.

The paper is structured as follows. Section 2 provides an overview of related work. Section 3 describes the general working principle of the system. Section 4 gives a detailed description of the current state representation, the actions used and the reward signal for the reinforcement learning agent. Section 5 provides some preliminary experimentation results. Section 6 closes the paper with a conclusion and an outlook.

2 Related Work

In order to get a complete understanding of the challenges faced in the research described here, and the setup of the solution approach, the reader is kindly referred to previous papers of the authors [6, 19, 20]. In this paper, the focus is entirely on the coupling of a discrete-event simulation model of a logistics network with a deep reinforcement learning algorithm, as briefly explained above.

2.1 The Simulation Software SimChain

The simulation tool used for the prototypical implementation of the system described here is called SimChain [8, 25]. It consists of generic building blocks for a logistics network simulation in Siemens Plant Simulation [24] and a corresponding data model stored in a MySQL database. The data model consists of 50 tables and holds the complete parameterization of the generic building blocks, including their placement on the map and therefore the structure of the network. The actual simulation model is dynamically instantiated from the data model at run time. Figure 1 illustrates the working principle.

Two design principles of SimChain were of importance for this research. First, since all modelling

is done with the parameters in the database, it offers a pure data-level interface to the simulation model. Changes in the simulation model can be realized by changing entries in the database. This allows to write software which can automatically apply changes to the simulation model. Second, it not only simulates the material flow, but also the information flow within the logistics network. The system load can be expressed by the customer orders coming into the system. The orders are processed through the system from customers to sites to suppliers, and the material flow is simulated from suppliers to sites to customers. It is therefore possible to change the logistics system configuration, e.g. the frequency of transports, while keeping the customer orders equal, in order to compare the performance of the two resulting system configurations. After each simulation run, the simulation output data and statistics are written to specific tables in the MySQL database, and from there they can be consumed by the controlling software. More details about this mechanism can be found in [19].

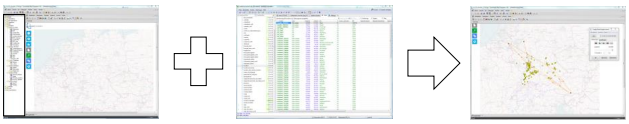

Figure 1: Working principle of the SimChain simulation software

SimChain can generate diverse simulation output data and statistics. The work presented in this paper uses the logistics costs and the performance of the logistics system. The logistics costs are calculated by summing up the inventory costs, the transportation costs and the handling costs. The performance of the logistics system is measured with the β service level. The β service level is a quantity-oriented performance measure. It expresses the percentage of customer orders which could be fulfilled without delay within a given timeframe [21]. It is an intuitive measure of the logistics system performance as it is perceived from a customer perspective. The formulae is given in (1).

$$\beta = 1 - \frac{delayed\ customer\ orders}{total\ amount\ of\ customer\ orders} \tag{1}$$

2.2 The Arcade Learning Environment

The Atari 2600 is a home video game console developed in 1977 and sold for over a decade [18]. A single game screen is 210 x 160 pixels with a 128-colour palette (RGB). 18 actions can be input to the game via a digital joystick: three positions of the joystick for each axis, plus a single button [2]. Screenshots of typical game screens are shown in figure 2. The ALE is built on top of Stella [26], an open-source Atari 2600 emulator, and adds a game-handling layer which transforms each game into a standard reinforcement learning problem. Through the ALE, researchers have access to several dozen games through a single common interface. It is a software framework designed to make it easy to develop agents that play arbitrary Atari 2600 games and therefore offers a method to evaluate the development of general, domain-independent artificial intelligence technology [2]. Its source-code is publicly available at [29].

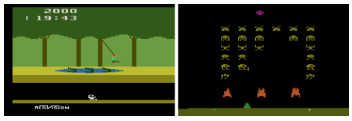

Figure 2: Screenshots of the Atari games *"Pitfall!"* and *"Space Invaders"* [2]

2.3 Playing Atari with Deep Reinforcement Learning

Creating a single algorithm that would be able to develop a wide range of competencies on a varied range of challenging tasks is a central goal of general artificial intelligence [14, 17]. To use reinforcement learning successfully in situations approaching real-world complexity, agents are confronted with a difficult task: they must derive efficient representations of the environment from high-dimensional

sensory inputs, and use these to generalize past experience to new situations [17]. Mnih et al. approached this challenge by creating an algorithm called DQN [16, 17]. They tested their DQN agent on the ALE and outperformed the best existing reinforcement learning methods on 43 of the 49 games without incorporating any of the additional prior knowledge about the Atari 2006 games used by other approaches.

Reinforcement learning in general considers tasks in which an agent interacts with an environment through a sequence of observations, actions and rewards. The general principle is illustrated in figure 3.

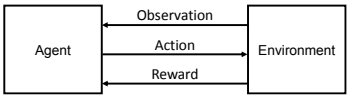

Figure 3: General principle of reinforcement learning

More specifically, the agent interacts with the environment at each of a sequence of discrete time steps, $t = 0, 1, 2, 3, \ldots$. At each time step t, the agent receives some representation of the environment's state $s_t \in S$, where S is the set of possible states. On the basis of s_t, the agent selects an action $a_t \in A(s_t)$, where $A(s_t)$ is the set of actions available in state s_t. One time step later, in part as a consequence of its action, the agent receives a numerical reward r_{t+1}. The goal of the agent is to select actions in a fashion that maximizes its cumulative future reward, also called return, R_t. At each time step t, the agent implements a mapping from states to probabilities of selecting each possible action. This mapping is called the agent's policy and is denoted π_t, where $\pi_t(s, a)$ is the probability that $a_t = a$ if $s_t = s$. Reinforcement learning methods specify how the agent changes its policy as a result of its experience [27].

Almost all reinforcement learning algorithms are based on estimating value functions [27]. Q-learning [31] is based on the Q-function, a function of state-action pairs that expresses how good it is for the agent to perform a given action in a given state regarding the expected return. More formally, the value of taking an action a in state s under a policy π, denoted $Q^\pi(s, a)$, expresses the expected return starting from s, taking the action a, and thereafter following policy π.

For small problems, the Q-function can be stored as a table, but for larger problems this table quickly gets too large to be stored in memory. Moreover, the time and data needed to fill the table accurately would be too high. In many tasks to which one would like to apply reinforcement learning, most states encountered will never have been experienced exactly before. The only way to learn anything at all on these tasks is to generalize from previously experienced states to ones that have never been seen before. Therefore, for larger problems, the key issue is that of generalization and some sort of function approximation is generally needed [27].

Advances in deep neural networks [3, 9, 11] in which several layers of nodes are used to build up progressively more abstract representations of data, have made it possible for artificial neural networks to learn concepts such as object categories directly from raw sensory data [17]. In order to approximate the Q-function, Mnih et al. used one particularly successful architecture of such deep neural networks, a convolutional neural network (CNN) [13]. A CNN uses hierarchical layers of tiled convolutional filters to mimic the effects of receptive fields [17].

Furthermore, they addressed known instabilities of using a nonlinear function approximator such as a neural network to represent the Q-function [30] with a novel variant of Q-learning, which uses two key ideas. First, they used a biologically inspired mechanism termed experience replay that randomizes over the training data for the neural network, thereby removing correlations in the observation sequences and smoothing over changes in the data distribution. Second, they used an iterative update that adjusts the action-values (Q) towards target values that are only periodically updated, thereby reducing correlations with the target. For details about these improvements see [17].

When using a neural network to approximate the Q-function, there are several possible ways to implement the architecture of the neural network. Because the Q-function maps state–action pairs to scalar estimates of their Q-value, some previous approaches used the state and the action as inputs to the neural network [12, 22]. The main drawback of such architectures is that a separate forward pass is required to compute the Q-value of each action, resulting in a cost that scales linearly with the number of actions [17]. Mnih et al. instead used an architecture in which there is a separate output unit for each possible action, and only the state representation is an input to the neural network. The outputs correspond to the predicted Q-values of the individual actions for the input state. The main advantage of this type of architecture is the ability to compute Q-values for all possible actions in a given state with only

a single forward pass through the neural network [16, 17].

Regarding the input to the CNN, Mnih et al. applied a preprocessing to the Atari 2600 frames, because working directly with the raw images would have required too much computation and memory utilization in their case. They extracted the Y channel, the luminance, from the RGB frame and rescaled it to a size of 84 x 84 pixels [16, 17]. They stacked 4 of those grayscale images, so that in total the input to their neural network consisted of an 84 x 84 x 4 matrix.

In the CNN, the first hidden layer convolves 32 filters of 8 x 8 with stride 4 with the input image and applies a rectifier nonlinearity. The second hidden layer convolves 64 filters of 4 x 4 with stride 2, again followed by a rectifier nonlinearity. This is followed by a third convolutional layer that convolves 64 filters of 3 x 3 with stride 1 followed by a rectifier. The final hidden layer is fully-connected and consists of 512 rectifier units. The output layer is a fully-connected linear layer with a single output for each action. The number of actions varied between 4 and 18 depending on the game [17].

Regarding the reward signal, Mnih et al. clipped all positive rewards at 1 and all negative rewards at -1, leaving 0 unchanged, since the scale of scores varies greatly from game to game [17].

To sum up, Mnih et al. successfully demonstrated that a single architecture can successfully learn control policies in a range of different environments receiving only the pixels and the game scores as inputs. They were using the same algorithm, network architecture and hyperparameters for different games. The games in which DQN exceled are extremely varied in their nature, from side-scrolling shooters (River Raid) to boxing games (Boxing) and three-dimensional car-racing games (Enduro).

3 Architecture and General Working Principle of the System

The general working principle of the system presented in this paper is illustrated in figure 4. Everything except the discrete-event simulation software has been implemented in the Python programming language. As mentioned in section 2.1, the discrete-event simulation of the logistics network is realized with SimChain, which is connected to a MySQL database to store the data model. The implementation of the DQN described in 2.3 has been built with the Python API for TensorFlow [28], an open source software library for numerical computations using data flow graphs.

In order to automatically apply actions to the discrete-event simulation model, a concept to generate specific actions from generic action types has been created. Each generic action type corresponds to a Python method with parameters. The parameterization of the method creates the specific action. Within the method, diverse SQL statements can be performed to change different tables of the MySQL database.

A model generation tool has been implemented which can generate data models for SimChain from the raw data provided by the cooperating company. A filter can generate data models depending on a specified amount of stock-keeping units (SKU). For the research presented here, an artificially created simulation model with 30 SKU, 5 sites, 3 suppliers, 103 customers, and 176 orders is used. The simulation horizon is always one year and the simulation time for one simulation run is approximately 10-15 seconds. Currently, the random number stream is always the same for each simulation run. Therefore, the presented model is in fact a deterministic model. For the future it is planned to run multiple replications with different random numbers for statistical correctness.

An action is actually a change in the configuration of the logistics system, while the system load – the customer orders – stays the same. Therefore, each simulation run after the application of an action evaluates how well or badly the new system configuration can handle the system load. The quality of a logistics system configuration can be expressed by different KPIs, as briefly mentioned in the introduction. Ultimately, a weighted sum of the KPIs would be the ideal measurement of the configuration quality. For this research, the authors followed a more simplistic approach, where the quality of a configuration is measured by the change in costs and performance compared to the initial configuration.

Because the number of possible actions depends on the current state of the system, the number of possible actions varies from state to state. Therefore, a method to assign an index to each action has been developed. First, the system evaluates the number of all possible actions, depending on the generic action types available. Second, a specific action index is assigned to each action, corresponding to a specific outlet of the CNN. In the current implementation, 150 possible actions are tested.

After an action is applied, the simulation model is instantiated and the simulation is run. The reward calculation generates the reward signal from the simulation output data by generating a scalar reward signal using the changes in costs and performance. The reward signal is then routed back to the DQN.

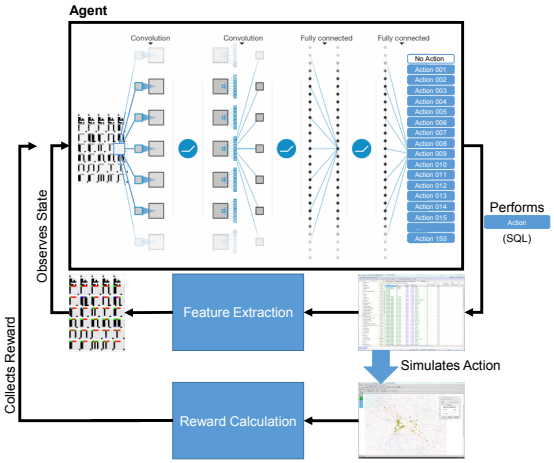

Figure 4: General working principle of the combination of the discrete-event simulation with the DQN algorithm. Convolutional network graphic adopted from [17].

The feature extraction selects features from different tables in the MySQL database in order to express the logistics system configuration, or the state of the system, as an image. The general idea behind the design of the state representation as an image is to profit from further upcoming research regarding the development of domain-independent agents. The image is designed to look similar to an Atari game screen in order to make the problem accessible for various agents.

4 State Representation, Actions and Reward Signal

Clearly, the performance of the overall system described here heavily depends on the quality of the state representation. The authors aimed to display necessary information as structured as possible, since a CNN excels at identifying structures in images. Furthermore, the state representation should also scale for larger logistics network simulations. Although the image is eventually converted to a grayscale image, i.e. a 2-dimensional matrix, as described in 2.1, the authors decided to create a colored state image, i.e. a 3-dimensional matrix, to make the state representation more readable for humans. This approach should only be used for the design phase of the system for research purposes, due to its computational overhead and the performance requirements in a production system.

Since the agent should learn a mapping from states to actions, the state representation should encode information to conclude from states to actions. E.g., if the agent should learn to make a decision regarding the inventory, useful information to make such a decision, e.g. inventory levels and customer demands, should be included in the state representation. For actions regarding e.g. machines, other information might be needed in the state representation. Thus, the information needed in the state representation heavily depends on the generic action types in the system. Consequently, the features which have to be selected from the MySQL database can be derived from the action types used.

The generic action type which is currently implemented in the system is illustrated in figure 5. The problem which this action type addresses is the classical logistics problem regarding whether it is better to keep an SKU on stock (and to replenish it periodically from a supplier) or to forward the SKU on demand from other sites in the logistics network. The example illustrates how complex an action type can get. As the reader can see in figure 5, the action type requires changes in the database tables for the transport relations, the site-SKU relationships and the supplier-site relationships. Furthermore, an action type can also include logic, e.g. the search for other sites in the scenario that are currently holding stock. Therefore, as can be seen in figure 5 on the right side, the replenishment from the supplier at site 2 has been replaced with two transport relations to the other sites in the scenario, site 1 and site 3, with a supplier split, not with only one single connection to one of these sites.

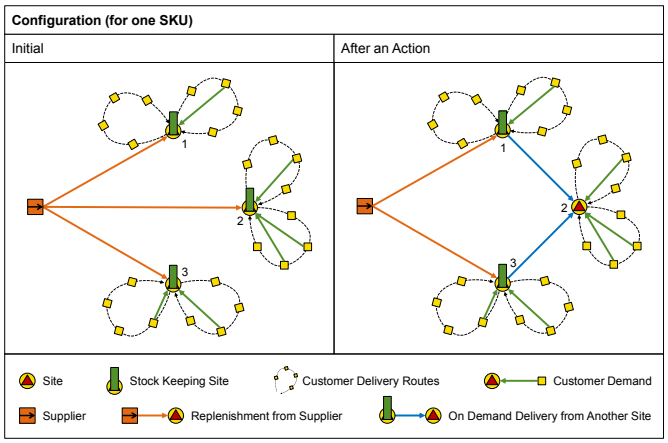

Figure 5: Currently implemented action type, illustrated as an action for one SKU

Figure 5 also demonstrates the important difference between a specific action and an action type. An action type is the generic description of the changes that need to be made in the data model of the simulation. A specific action describes the concrete changes that need to be done in the data model.

The currently implemented action type has the signature `get_replenishment_from_other_sites(sku, site)`. It requires two parameters, SKU and site, to result in a specific action. Since there are currently 30 SKU and 5 sites in the simulation model, and the action type is theoretically possible for each SKU at each site, this results in 150 possible specific actions. In addition, there is the artificial specific action "No Action", which leads to 151 outlets for the CNN described above.

In order to address the requirements regarding the structuredness and the scalability of the state representation, the state image is built from different image segments. Each segment corresponds to a segment type, similar to the previously explained relationship between actions and action types. The size of the specific segments is fixed, which helps to keep the resulting image symmetric. This should help the CNN to identify patterns within the data. The actual segment size is derived from the size of the largest segment type. Figure 6 shows the two segment types used in the state representation and examples of segments. Currently, the transport relation segment type is the largest and, therefore, determines the size of the other segments. The transport relation segment type essentially symbolizes a classical transport matrix. Its size is determined by the locations in the logistics network. The logistics network in the research configuration has 8 locations (5 sites and 3 suppliers), consequently the size is 8 x 8 pixels.

The second segment type, the SKU information type, holds information for each SKU, so one segment is generated for each SKU in the logistics network. The first information extracted from the database is whether a site is keeping the respective SKU on stock or not. Green indicates that the corresponding site is stock-keeping. Blue means that the SKU is ordered on demand from other sites, and red means that the SKU is not available at the site at all. Furthermore, some numerical values are extracted for each SKU, such as the initial stock at each site, the amount of customers for the SKU at each site, the amount of orders for the SKU at the location, the total demand for the SKU at the location and the minimum replenishment size for the SKU at the location. Numerical values are currently expressed as RGB colors, covering a range of 256 x 256 x 256 = 16777216 values. Unfortunately, in the current example, the numerical values are so low that all the pixels look black. In fact, many of the pixels are dark gray. While this can be confusing for humans, the CNN consumes the states as matrices of numerical values, so even small differences in the numerical values should be noticeable. It is an issue which is currently investigated in the ongoing research regarding the state representation. Finally, the SKU information type expresses which location in the logistics network is supplying the respective SKU.

The size of the segment type sizes gets calculated dynamically and, therefore, also the image size

varies for each logistics network. For the first segment row of the state representation, 5 transport relation segments are generated, one transport relation segment for each day of the workweek. Thus, not only transport relations are extracted, but also transport frequencies. This is especially important regarding the effects on the performance of the logistics network. The rest of the state representation is filled with SKU information segments, one segment for each SKU in the simulation model. In the small logistics network presented here, there are 30 SKUs and therefore 30 SKU information segments. These segments are placed in a 5 x 6 grid below the transport relation segments. Therefore, the state representation of the example contains 5 x 7 segments, each with a size of 8 x 8 pixels, which results in a concrete images size of 40 x 56 pixels.

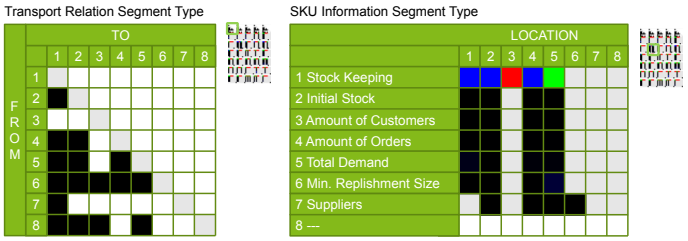

Figure 6: Segment types used in the state representation and examples of segments

As briefly mentioned above, the features represented in the state image can be directly derived from the action types used. Therefore, there is a strong correlation between action types and segment types. When introducing new action types to the system, such as actions regarding the machines or the working staff, one would also have to extend the state representation with new segment types.

Regarding the reward, one would expect that the changes in the logistics system configuration described above would cause several effects on the cost structure and performance of the logistics system. If the customer demand, i.e. the system load, stays the same, the changes described above would probably influence the inventory costs, the transportation costs, the handling costs and also the delivery performance, e.g. expressed by the β service level.

In order to generate a scalar reward signal for the reinforcement learning agent, a decrease of the total costs after taking an action is translated into a positive reward, an increase is translated into a negative reward. Furthermore, the difference in performance has to be incorporated into the rewards. The authors decided to define a penalty cost which is multiplied with the percental change in the β service level. If the service level increases, a bonus instead of costs is generated. The service level costs should express the loss of customer orders in the future due to unsatisfied customers or the increase of orders from satisfied customers.

The difference in the logistics costs and the costs associated to the β service level are summed up and interpreted as the total costs caused by an action. Finally, these total costs are scaled between -1 and 1 to generate the final reward signal which is sent back to the agent. The scaling is done to get as close as possible to the parameters used in the original DQN implementation.

5 Experimentation Scope and Preliminary Results

First experiments have been conducted with the implementation and the results look promising. For the example presented in this paper, the authors assumed that one episode for the reinforcement learning agent consists of taking three actions. Once the three actions have been taken, the simulation is reset to its initial state and the agent can start a new episode. Therefore, the agent should gradually learn to take the best three actions possible for the initial state of the logistics network. Parameter-wise the authors stayed with the same parameters presented in the papers of Mnih et al. for these initial experiments.

The initial costs generated by the logistics network simulation were 73,897 € and the initial β service level was 79.89 %. The reinforcement learning agent was trained with 1000 episodes, each consisting of 3 actions, which resulted in up to 3000 evaluation runs. The total runtime of the experiment was 7 h. As expected for a simheuristic approach, the bottleneck in terms of computing performance was not the backpropagation through the CNN, but the simulation time needed for each evaluation run.

The results of the experiments are shown in figure 7. As the reader can see from the average result per episode, the agent was able to improve its strategy over the episodes. It finally developed a strategy that reduced the simulated logistics network costs by 2,157 € and increased the simulated logistics service level by 0.57 %. While the absolute improvements are not of major significance, the experiments showed that the implementation of the system in general seams feasible.

Figure 7: Preliminary experimentation results

6 Conclusion and Outlook

The work presented in this paper is an important milestone towards the development of a decision support system which uses a discrete-event simulation and reinforcement learning to propose integrated action combinations to managers of logistics networks. The implementation shows promising results, while some aspects still remain open.

First of all, the authors aim to further research the scaling of the state representation for larger logistics networks, gradually working towards networks with thousands of SKUs. Furthermore, it needs to be investigated how the system performs once more action types are integrated into the system, which could also affect the size of the state representation.

Another important aspect to be researched is the architecture of the CNN. Since the images fed to the CNN are hand-crafted, a custom adjustment of the filter sizes could help to break down the dimensions of larger state representations.

As mentioned before, the bottleneck in terms of computation time probably still remains the discrete-event simulation, which also becomes slower for larger logistics networks. It is expected that, even if the CNN becomes larger, the computation time needed for the backpropagation in the CNN will not exceed the computation time of the simulation.

Another minor aspect which will be integrated in the future is to assign implementation costs to the action types. These costs should be integrated into the reward for the agent to reflect the costs that would be necessary to implement the action in the real logistics network.

References

[1] D. Achenbach. Process Model for Short Term Booking of Customized Products on Online Market Places. *JOEBM*, vol. 3, 10:977–983, 2015.

[2] M.G. Bellemare, Y. Naddaf, J. Veness, M. Bowling. The Arcade Learning Environment: An Evaluation Platform for General Agents. *Journal of Artificial Intelligence Research*, vol. 47:253–279, 2013.

[3] Y. Bengio. Learning Deep Architectures for AI. *FNT in Machine Learning*, vol. 2, 1:1–127, 2009.

[4] M. Biesen, A. Tavakoli, T. Hegmanns. Managing Logistics Performance of Numerous Facilities under Consideration of Facility-individual Preconditions. In Jones, C., editor, *Proceedings of the 6th International Colloquium on Business & Management (ICBM),* Prince of Songkla University, Bangkok, Thailand, 2013.

[5] C. Bruns, T. Hegmanns. Performance Measurement System for a Changeable Stocking Strategy in Distribution Systems of Commercial Enterprises. In Jones, C., editor, *Proceedings of the 6th International Colloquium on Business & Management (ICBM),* Prince of Songkla University, Bangkok, Thailand, 2013.

[6] F. Dross, M. Rabe. A SimHeuristic Framework as a Decision Support System for Large Logistics Networks With Complex KPIs. In Wittmann, J; Deatcu, C., editors, *Proceedings of the 22nd Symposium Simulationstechnik (ASIM 2014)*, pages 247–254, HTW Berlin, Germany, 2014.

[7] I. Goodfellow, Y. Bengio, A. Courville. Deep Learning, 1. ed. MIT Press, Cambridge, Massachusetts, USA, 2016.
[8] K. Gutenschwager, K. Alicke. Supply Chain Simulation mit ICON-SimChain. In T. Spengler, S. Voß, H. Kopfer, editors, *Logistik Management*, pages 161–178, Physica, Heidelberg, Germany, 2004.
[9] G.E. Hinton, R.R. Salakhutdinov. Reducing the Dimensionality of Data with Neural Networks. *Science (New York, N.Y.)*, vol. 313, 5786:504–507, 2006.
[10] A.A. Juan, M. Rabe. Combining Simulation with Heuristics to Solve Stochastic Routing and Scheduling Problems. In Dangelmaier, W; Laroque, C; Klaas, A., editors, *Simulation in Produktion und Logistik 2013: Proceedings of the 15th ASIM Conference on Simulation in Production and Logistics*, pages 641–649, Paderborn, Germany, 2013.
[11] A. Krizhevsky, I. Sutskever, G.E. Hinton. ImageNet Classification with Deep Convolutional Neural Networks. In F. Pereira, C. Burges, L. Bottou, K. Weinberger, editors, *Advances in Neural Information Processing Systems 25 (NIPS 2012)*, Lake Tahoe, Nevada, USA, 2012.
[12] S. Lange, M. Riedmiller. Deep Auto-encoder Neural Networks in Reinforcement Learning. In *The 2010 International Joint Conference on Neural Networks (IJCNN), IEEE proceeding*, pages 1–8, Barcelona, Spain, 2010.
[13] Y. Lecun, L. Bottou, Y. Bengio, P. Haffner. Gradient-based Learning Applied to Document Recognition. *Proc. IEEE*, vol. 86, 11:2278–2324, 1998.
[14] S. Legg, M. Hutter. Universal Intelligence: A Definition of Machine Intelligence. *Minds & Machines*, vol. 17, 4:391–444, 2007.
[15] J.L. McClelland, D.E. Rumelhart. Parallel Distributed Processing: Explorations in the Microstructure of Cognition, 1. ed. MIT Press, Cambridge, Massachusetts, USA, 1986.
[16] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra et al. Playing Atari with Deep Reinforcement Learning, NIPS Deep Learning Workshop, 2013.
[17] V. Mnih, K. Kavukcuoglu, D. Silver, A.A. Rusu, J. Veness, M.G. Bellemare et al. Human-level Control through Deep Reinforcement Learning. *Nature*, vol. 518, 7540:529–533, 2015.
[18] N. Montfort, I. Bogost. Racing the beam: The Atari Video Computer System. MIT Press, Cambridge, Massachusetts, USA, 2009.
[19] M. Rabe, F. Dross. A Reinforcement Learning Approach for a Decision Support System for Logistics Networks. In Yilmaz, L; Chan, W. K. V; Moon, I; Roeder, T. M. K; Macal, C; Rossetti, M. D., editors, *Proceedings of the 2015 Winter Simulation Conference (WSC)*, pages 2020–2032, Huntington Beach, CA, USA, 2015.
[20] M. Rabe, F. Dross, A. Vennemann. A Procedure Model for the Credible Measurability of Data Warehouse Metrics on Discrete-event Simulation Models of Logistics Systems. In Rabe, M; Clausen, U., editors, *Simulation in Produktion und Logistik 2015: Proceedings of the 16th ASIM Conference on Simulation in Production and Logistics*, pages 167–176, Dortmund, Germany, 2015.
[21] R. Ray. Supply Chain Management for Retailing. Tata McGraw-Hill Education, New Delhi, 2010.
[22] M. Riedmiller. Neural Fitted Q Iteration – First Experiences with a Data Efficient Neural Reinforcement Learning Method. In J. Gama, editor, *Proceedings of the 16th European Conference on Machine Learning (ECML 2005)*, pages 317–328, Springer, Berlin, 2005.
[23] B. Scholkopf. Artificial Intelligence: Learning to See and Act. *Nature*, vol. 518, 7540:486–487, 2015.
[24] Siemens PLM Software. Tecnomatix Plant Simulation. http://www.plm.automation.siemens.com/de_de/products/tecnomatix/plant_design/plant_simulation.shtml.
[25] SimPlan AG. SimChain. http://www.simchain.net/.
[26] Stella: A Multi-Platform Atari 2600 VCS Emulator. https://stella-emu.github.io/.
[27] R.S. Sutton, A.G. Barto. Reinforcement Learning: An Introduction. MIT Press, Cambridge, Massachusetts, USA, 1998.
[28] TensorFlow. https://www.tensorflow.org/.
[29] The Arcade Learning Environment. http://www.arcadelearningenvironment.org/.
[30] J.N. Tsitsiklis, B. van Roy. An Analysis of Temporal-difference Learning with Function Approximation. *IEEE Trans. Automat. Contr.*, vol. 42, 5:674–690, 1997.
[31] C. Watkins. Learning from Delayed Rewards. Ph.D. thesis, Kings College, Cambridge, England, 1989.