Proceedings of the 2020 Winter Simulation Conference
K.-H. Bae, B. Feng, S. Kim, S. Lazarova-Molnar, Z. Zheng, T. Roeder, and R. Thiesing, eds.

INTEGRATION OF DEEP REINFORCEMENT LEARNING AND DISCRETE-EVENT SIMULATION FOR REAL-TIME SCHEDULING OF A FLEXIBLE JOB SHOP PRODUCTION

Sebastian Lang
Fabian Behrendt

Fraunhofer Institute for
Factory Operation and Automation (IFF)
Sandtorstraße 22
Magdeburg, 39106, GERMANY

Nico Lanzerath
Tobias Reggelin
Marcel Müller

Institute of Logistics and
Material Handling Systems (ILM)
Otto von Guericke University Magdeburg
Universitätsplatz 2
Magdeburg, 39106, GERMANY

ABSTRACT

The following paper presents the application of Deep Q-Networks (DQN) for solving a flexible job shop problem with integrated process planning. DQN is a deep reinforcement learning algorithm, which aims to train an agent to perform a specific task. In particular, we train two DQN agents in connection with a discrete-event simulation model of the problem, where one agent is responsible for the selection of operation sequences, while the other allocates jobs to machines. We compare the performance of DQN with the GRASP metaheuristic. After less than one hour of training, DQN generates schedules providing a lower makespan and total tardiness as the GRASP algorithm. Our first investigations reveal that DQN seems to generalize the training data to other problem cases. Once trained, the prediction and evaluation of new production schedules requires less than 0.2 seconds.

1 INTRODUCTION

In view of the recent developments in manufacturing, which are summarized under the term “Industry 4.0”, researchers and practitioners are discussing the opportunities of a decentralized, flexible and modular production. Concepts such as the matrix production (Bauernhansl et al. 2014; Greschke et al. 2014) propose a flexible routing of jobs by using autonomous mobile robots instead of fixed conveyor systems to transport products and material. Against this background, job shop scheduling problems and the flexible job shop problem in particular are becoming increasingly relevant.

The job shop scheduling problem is a combinatorial optimization problem, in which a certain number of jobs must complete a set of operations on a given number of machines. Typically, a specific operation can be only completed by a certain machine. A particularity of the job shop problem is that the sequence of operations to be completed is specific for each job. The job shop problem is usually NP-hard (Baker and Trietsch 2009; Brucker 2007; Pinedo 2016).

The flexible job shop generalizes the job shop problem by allowing the execution of specific operations on several machines. In the last decades, many solution techniques have been investigated to solve the flexible job shop problem, such as (mixed-) integer programming, priority dispatching rules and metaheuristics (Xie et al. 2019). The aforementioned methods provide either a high solution quality at the expense of computational time or vice versa. However, current challenges, such as shorter production lead times, shorter product life cycles, a greater product diversity or uncertainties in the supply chains (Gershwin 2018), require the investigation of new methods for production planning, which react to sudden changes in the production environment and which are capable to immediately compute solutions of sufficient quality (Lang et al. 2019).

Based on this motivation, we investigate the application of Deep Q-Networks together with Discrete-Event Simulation to solve a flexible job shop problem with integrated process planning. The analyzed problem was first described by Rajkumar et al. (2010). The integrated process planning is a particularity of the problem and requires the selection of an operation sequence, before releasing the job.

The further paper is organized in five sections. The next section provides a short theoretical foundation of reinforcement learning and Deep Q-Networks in particular. In the third section, we review related research and discuss the contribution of this paper. Section 4 describes the concept and implementation of the Deep Q-Network algorithm and the Discrete-Event Simulation model. In the fifth section, we describe our experiments and results. Section 6 is dedicated to the conclusions and outlook.

2 DEEP REINFORCEMENT LEARNING AND DEEP Q-NETWORKS

Deep Q-Network (DQN) is a machine learning algorithm, which is classified as deep reinforcement learning. The general idea of reinforcement learning is to train an agent, which interacts with its surrounding environment. The agent is a prediction model, which maps states of its environment (input) to actions (output). The agent and its environment are interacting in a loop. Every time the agent performs an action, the state of the environment changes and the agent is forced to select the next action. The process repeats, until the environment terminates. In contrast to supervised learning, reinforcement learning methods do not require labeled training data (i.e. an expected output to a given input). Training signals are instead provided by a reward function that evaluates actions of an agent with positive or negative rewards. The objective of the agent is to maximize the amount of rewards over time. By this means, the agent learns to predict the best action to a given state. In general, an agent can be represented by any kind of model that is capable to learn, such as a table, a rule-based logic or a regression model. However, the main difference between deep reinforcement learning and traditional methods is that the agent is always represented by one or more deep neural networks (Arulkumaran et al. 2017; François-Lavet et al. 2018; Sutton and Barto 2018).

DQN was introduced by Mnih et al. (2013). Better known, however, is the follow-up article of Mnih et al. (2015), which provides a more detailed and comprehensible description of the algorithm. DQN is an extension of the Q-Learning algorithm (Watkins 1989). Q-Learning relies on the assumption that the action space of an agent is finite and discrete. For a given state, the algorithm evaluates the advantageousness of every action with so-called Q-values. The agent may select the action with the highest Q-value or a random action. The latter is necessary to explore the solution space. However, as time progresses, the probability for selecting a random action decreases. The Q-values are updated with a modification of the Bellman equation:

$$Q'(s_t, a_t) = Q(s_t, a_t) + \alpha * (r_{t+1} + \gamma * \max_a(Q(s_{t+1}, a)) - Q(s_t, a_t)) \quad (1)$$

$Q(s_t, a_t)$ and $Q'(s_t, a_t)$ are the recent and the updated Q-values for all possible actions a in state s at time step t, α is the learning rate, which is usually a value close to zero, r_{t+1} is the resulting reward for the action at the current time step. The discounting factor γ is a value close to, but less than one. The term $\max_a Q(s_{t+1}, a)$ denotes the prediction of the best action for the future state s_{t+1}.

A major drawback of the original Q-Learning algorithm is that the state-action space is represented by a table. This representation requires a discretization of the state space. Each value of each state corresponds to a row in the Q-table. Therefore, the algorithm is only applicable to problems, where the underlying environment has a fairly low complexity. The main idea of DQN is to replace the Q-table by two deep neural networks. Figure 1 on the following page illustrates the architecture and the training procedure of the DQN algorithm.

At every time step, the current state, the action carried out, the resulting reward and the future state are stored as 4-tuple transition in the replay memory. The intention of storing transitions is to collect training data over time. Once the replay memory contains a minimum number of transitions, the agent starts to learn.

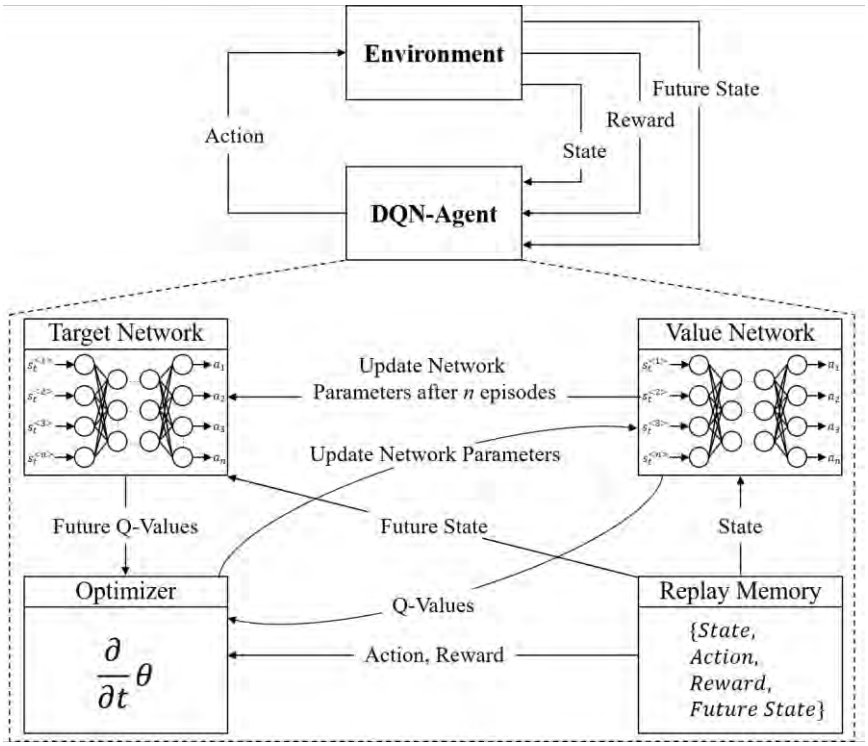

Figure 1: Concept of DQN

Every time the replay memory has been updated, DQN generates a mini batch of transitions for training. Using the action, the reward, the current and the future Q-values, the algorithm updates the current Q-values according to equation (1). Finally, the optimizer updates the parameters θ of the value network, based on gradient descent by using the difference between the current and the future Q-values. The parameters of the target network remain constant over several episodes to avoid interferences when predicting the current and future Q-values. The target network is updated with the current parameters of the value network after a user-defined number of episodes.

3 RELATED RESEARCH

In this section, we briefly summarize the related research and discuss the contribution of this paper. As there are only few articles that describe deep reinforcement learning for job shop scheduling in particular, we extend the scope of our review to reinforcement learning (RL) for production scheduling in general. The following review is essentially based on our previous publication (Lang et al. 2020), but has been extended by recently published research.

To the best of our knowledge, Zhang and Dietterich (1995) present the first application of RL for production scheduling. The authors use RL to evaluate the feasibility and quality of schedules in a job shop environment. Aydin and Öztemel (2000) apply RL to select priority dispatching rules depending on the current state of a production system. Comparable approaches are investigated by Wang and Usher (2005), Shiue et al. (2018), Lin et al. (2019) and Luo (2020). Paternina-Arboleda and Das (2005) use RL to learn a dynamic control policy for a stochastic lot scheduling problem with a single machine. The agent changes the setup type of the machine according to the current system state. Qu et al. (2016) describe a similar approach for a multi-stage flow shop problem. Martínez et al. (2011) apply tabular Q-Learning for selecting and sequencing operations of jobs in a flexible job shop environment. The objective is the minimization of the makespan. The authors conduct experiments on ten problem instances. They compare the solution quality of Q-Learning with Genetic Algorithm, Ant Colony Optimization, the GENACE algorithm and

Tabu Search. Apart from genetic algorithms, the authors' Q-Learning approach performs better than the compared metaheuristic in almost all cases.

Stricker et al. (2018) present an RL-based multi-agent system for a semiconductor production. The agents are responsible for assigning jobs to machines and for sequencing jobs in front of each machine. The multi-agent system improves the system utilization by 10% compared to the FIFO dispatching strategy. Waschneck et al. (2018) combine DQN with supervised learning for job-shop scheduling in a semiconductor production. The DQN agents are responsible for the release and sequencing of jobs. The agents are pre-trained with supervised learning. The training samples are taken from an event-based scheduler, which operates based on expert knowledge.

Zhang et al. (2018) apply neural fitted Q-Learning for batching and sequencing orders in a job shop with three machines. The authors compare their approach with the FIFO dispatching rule combined with three different batching strategies. Neural fitted Q-Learning achieves a shorter cycle time and a lower average WIP compared to the three combinations of FIFO and batching strategies. In terms of system throughput, however, some combinations of FIFO and batching still perform better than Q-Learning. Shufang et al. (2019) investigate different state-action spaces and reward functions for the neural fitted Q-Learning algorithm with the objective to minimize the total completion time and the maximal lateness of jobs in a single-machine environment. The authors observe that unnecessary inputs can impair the learning of the agent. Shi et al. (2020) deploy DQN to schedule jobs of an automated production line. The authors train the agent to control a free-moving transport robot with random transport times. The robot is responsible for moving jobs from one machine to another. The authors compare the performance of DQN with the FIFO, LPT and SPT dispatching strategy. DQN is able to achieve a comparatively high utilization of machines. In some scenarios, DQN often achieves the highest machine utilization. In terms of job waiting times, however, the dispatching strategies outperform DQN most of the time.

In a previous paper, we apply NeuroEvolution of Augmenting Topologies (NEAT) to solve a two-stage hybrid flow shop scheduling problem with family setup times (Lang et al. 2020). NEAT is a genetic algorithm, which evolves the structure and hyper-parameters of neural networks. NEAT is comparable to RL, as the exploration of the solution space is guided by a fitness function (similar to the reward function in RL). It turns out that the solution quality of NEAT is comparable to metaheuristics, such as Tabu Search and Simulated Annealing, while the runtime is significant lower.

In conclusion of the related research, we want to summarize the contribution of this paper: (1) We apply DQN for the flexible job shop scheduling problem with integrated process planning of Rajkumar et al. (2010) and compare the performance of DQN with the authors' GRASP algorithm. (2) We present a concept for the integration of DQN and DES, thus to teach an agent the allocation of jobs to machines, while interacting with the simulation model. (3) We train a second DQN agent to select operation sequences for jobs. A particularity is the representation of the agent. Due to the lack of meaningful inputs to describe the state space of the operation sequence selection problem, we use LSTM networks instead of dense neural networks. The underlying assumption is that previous selection decisions influence the selection of operation sequences of future jobs. By using LSTM, we are able to invoke previous state-action pairs for the computation of current and future Q-values.

4 CONCEPT AND IMPLEMENTATION

This section comprises the concept and implementation of the DQN algorithm and the DES model of the flexible job shop. The section is organized as follows: First, we will shortly summarize the analyzed flexible job shop problem. Second, we will describe the DES model. Third, we will explain the particularities of our DQN implementation, respectively the state-action spaces and the reward functions of the agents for allocating jobs to machines and for selecting operation sequences for jobs.

4.1 Problem Description

The following description is essentially a summary of Rajkumar et al. (2010). To obtain a full problem statement including the problem instances, we recommend to read the article of the authors. We want to emphasize that some of the following definitions are not required to understand the problem. However, they will become relevant, when we explain the simulation model.

The considered job shop problem consists of J jobs and M machines, whereas the number of jobs and machines depends on the problem instance. Furthermore, the last problem instance is characterized by higher due dates, because of the significantly higher number of jobs. Table 1 shows the differences between the problem instances in a highly aggregated manner. All due dates and process times are deterministic.

Table 1: Properties of the problem instances.

Problem Instance	Number of Machines	Number of Jobs	Due Dates	Process Times (min, max, μ, σ)
1	5	5	{80, 100, 110, 120}	(5, 98, 54.8, 31.6)
2	8	8		
3	5	10		
4	5	20	{400, 500, 600, 700}	

Every job $\in \{1, \dots, J\}$ is available at time zero and must complete a certain number of operations before leaving the system. The number of operations to complete depends on the selected operation sequence. A job has four possible operation sequences $O_{j,o}$ $\forall\, o \in \{1, .., 4\}$, where each sequence consists of two, three or four operations from the set $\{P_1, P_2, \dots, P_p, \dots, P_7\}$. We denote the length of a operation sequence (i.e. the number of operations to be completed) as $|O_{j,o}|$ and the p^{th} operation of a sequence as $O_{j,o}(p)$. Both, the operation sequences and the decision for an operation sequence are immutable.

Each operation can be executed on each machine. However, the process times of operations vary depending on the selected machine $P_p(M_m)$ $\forall\, p \in \{1, .., 7\}, m \in \{1, \dots, M\}$.

Let us define Q_m as the buffer in front of a machine m. We further denote the length of a buffer – the number of jobs within a buffer respectively – as $|Q_m|$ and the q^{th} job of the buffer as $Q_m(q)$, where $q \in \{0, \dots, (|Q_m| - 1)\}$. Since we have no further information, we assume that the capacities of buffers are infinite. Each machine can only process one job at the same time.

The objective is to identify a production schedule, defined by J operation sequence decisions and $\sum_{j=1}^{J} |O_{j,o}|$ machine allocation decisions. We assess the quality of a production schedule based on the resulting makespan C_{Max} and total tardiness T. The makespan C_{Max} is the maximum completion time over all jobs, $C_{Max} = \max(C_j\ \forall\, j \in \{1, \dots, J\})$, while the total tardiness T is the sum of tardiness over all jobs $T = \sum_{j=1}^{n} T_j$. The tardiness of a single job T_j is the difference between its completion time and due date, if the job is finished after its due date

$$T_j = \begin{cases} c_j - d_j & if\ (c_j - d_j) > 0 \\ 0 & otherwise \end{cases}$$

4.2 Discrete-Event Simulation Model

We implemented the model of the flexible job shop with the open-source Python library salabim (van der Ham 2018). The decision for salabim against an off-the-shelf tool was mainly driven by the fact that our agent is also implemented with Python. Thus, using salabim reduces the effort for both, the integration of the DES model with DQN and the communication between the agents and the model. Modeling in salabim is fundamentally based on the SIMULA language (Dahl and Nygaard 1966), as it involves concepts like

activating, passivating and holding objects. In contrast to many commercial DES tools, such as ARENA or Plant Simulation, the model behavior is not only defined by stationary material flow blocks, but also by routines within the flow objects. The process logic of each job and each machine are simplified in figure 2.

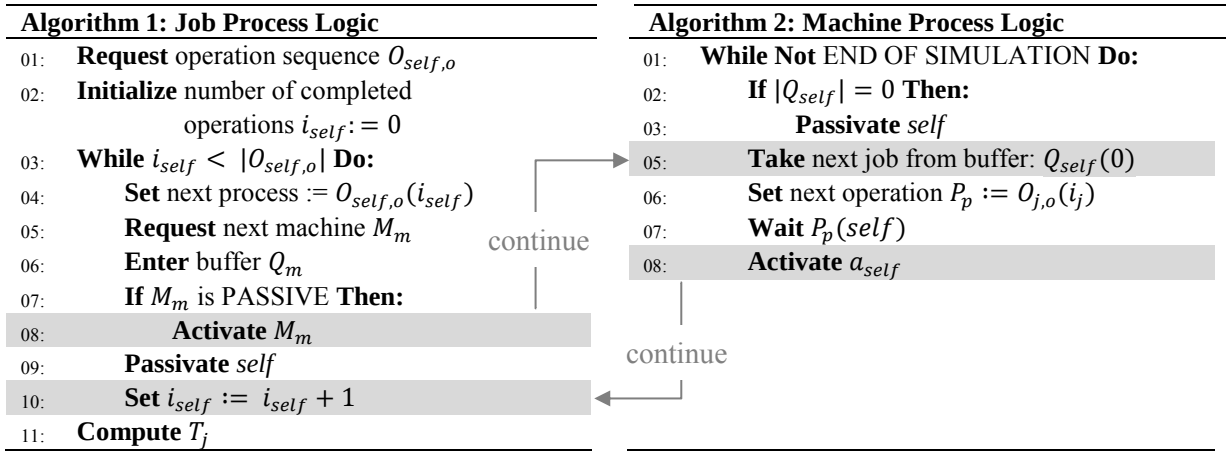

Figure 2: Process logic of the DES model (*self* denotes the object that executes the method).

Furthermore, the model contains a source object with own process logic. When starting the simulation, the source generates the number of jobs according to the loaded problem instance, while all machines go in passive state, as there are no jobs in the corresponding upstream buffers at the beginning of the simulation. After a job has been initialized, its process logic will be executed automatically. At the beginning, the job requests a operation sequence to be started. The operation sequence is determined either randomly or by a DQN agent. After that, the job reiterates the following steps until it is completed. First, the job requests the next machine to enter. Similar to the selection of operation sequences, the allocation of jobs to machines is either randomized or controlled by a DQN agent. Second, the job enters the buffer of the machine. If the corresponding machine is currently passive, the job reactivates the machine, which means that the machine continues the process from the step where it has been recently passivated. Meanwhile, the job goes in passive state until it has been processed by the current machine. A machine is active, while notifying jobs in the upstream buffer. The machine receives jobs according to the FIFO dispatching strategy. After completing the operation, the machine reactivates the job. If the number of completed operations does not correspond to the length of the operation sequence, the job requests the next machine to enter. Otherwise, the job computes its tardiness before leaving the system.

In order to verify and validate our model, we developed a simple visualization based on the native animation capabilities of salabim. The visualization allows to track the position of jobs in the system, while the simulation is executed. Furthermore, we are able to observe several system KPIs, for instance whether a machine is currently active or passive, the queue lengths in front of machines, the current workload of machines, the makespan and the current total tardiness. Figure 3 shows a snapshot of the job shop problem with 8 jobs and 8 machines.

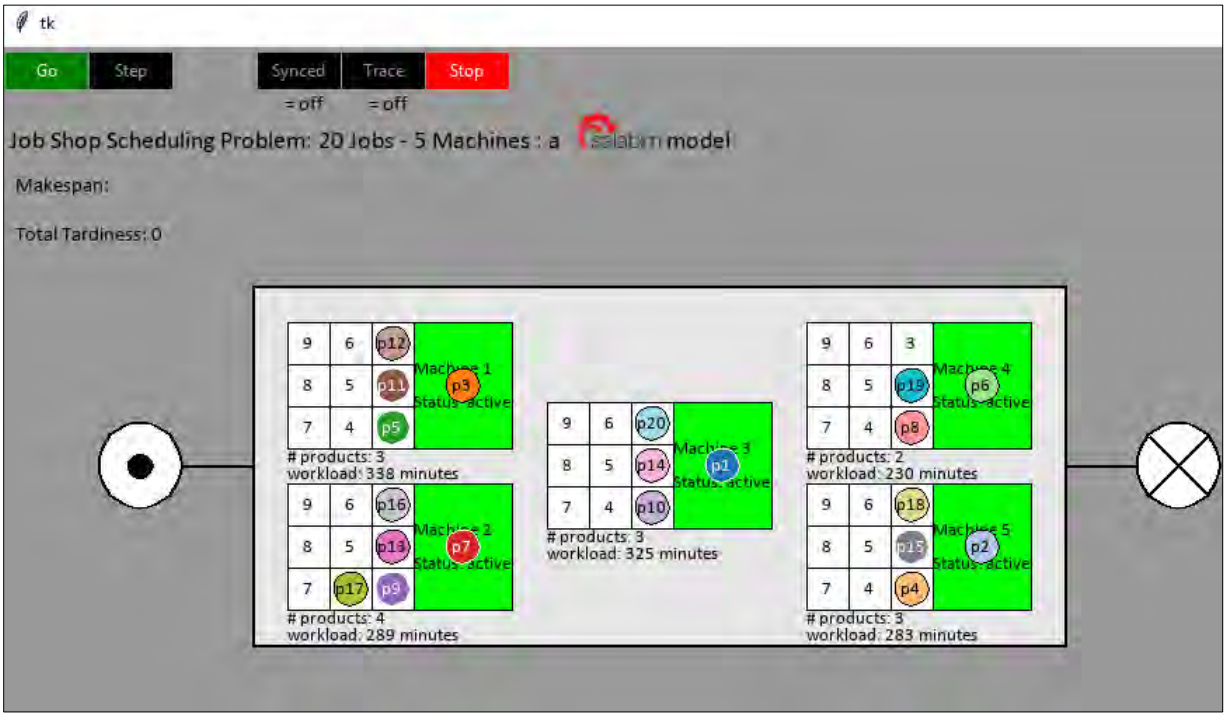

Figure 3: Animated DES model of the flexible job shop problem with 8 jobs and 8 machines.

4.3 Deep Q-Networks Implementation

As discussed in section 4.1, the solution of the present problem requires both, the allocation of jobs to machines and the selection of operation sequences for jobs. Therefore, our solution strategy deploys two agents, where one agent is responsible for machine allocation decisions, while the other for selecting operation sequences. Table 2 on the following page summarize essential decisions for designing our agents, i.e. the state and action spaces, the network architectures and the reward functions.

We determined the network architecture of the agents through an explorative search. We limited our experiments by considering only two hidden layers for each agent and by assuming that the number of hidden neurons is the same for both hidden layers. For each agent, we started to experiment with 8 neurons per hidden layer. We doubled the amount of neurons every time the agent was not able to identify a meaningful policy. In general, the decision space for designing deep neural networks is very large, requires extensive investigations and is worth to be discussed in a separate article. In this paper, however, we will not focus on parameter studies, but on the general implementation and of the DQN agents.

The design of rewards is another major factor for the successful training of an agent. When dealing with an optimization problem, the intuition arises to use the objective function for rewarding decisions of an agent, which is in our case the makespan and total tardiness. In contrast, the training signal is most evident, when different decisions of an agent are evaluated individually. Both, the makespan and the total tardiness, are aggregated performance indicators that summarize the quality of all planning decisions into a single value. It is therefore reasonable to consider different reward functions for machine allocations and the selection of operation sequences as well as to differentiate the giving of rewards during and after the simulation. For rewarding the allocation of jobs to machines during the simulation, the workload of the machine, to which the job has been assigned, has proven to be an unambiguous measure. However, mapping the system workload of a specific time step to the selection of an operation sequence is not possible. Instead, we use the earliness of a job as positive reward and the lateness of a job as negative reward to evaluate the selection of a single operation sequences while simulating. Both agents receive the negative sum of the makespan and total tardiness as episode reward, when the simulation ends.

Table 2: Essential design decisions for our DQN implementation.

	Agent for Allocating Jobs to Machines	**Agent for Selecting Operation Sequences**
States (Input)	Job States (2 neurons): • Due date (1 neuron) • Completeness of job (ratio between the number of completed operations and the number of operations to be completed) (1 neuron) System States (10 or 16 neurons): For each machine: • Process time of the next operation of the job to allocate (5 or 8 neurons) • Workload of the buffer summed with the remaining time to complete the current operation (5 or 8 neurons)	Job States (8 × 4 neurons): • The four operation sequences of the considered job, defined by the indexes of the four operations (operation sequences with less than four operations are supplemented with the corresponding number of entries containing zero) (4 × 4 neurons) • The average process time over all machines of each operation of each operation sequence of the considered job (4 × 4 neurons)
Agent (Network)	Two hidden dense layer, each with 16 neurons each and ReLu activation function	Two hidden LSTM layer, each with 1024 cells, Tanh activation function and 20% dropout chance
Actions (Output)	Allocation to Machine $1, 2, m, \dots, M$ (5 or 8 neurons)	Selection of Operation Sequence $1, 2, 3, 4$ (4 neurons)
Reward	Interim (job-specific): For the machine, where the job has been allocated: Workload of the upstream buffer (including the operation time of the allocated job) summed with the remaining time to complete the current operation Episode (end of simulation): $(-1) * (C_{Max} + T)$	Interim (job-specific): $c_j - d_j$ Episode (end of simulation): $(-1) * (C_{Max} + T)$

5 EXPERIMENTS AND RESULTS

Our design of experiments focuses the following questions: (1) How does DQN perform on the flexible job shop problem with 5 and 8 machines in terms of makespan and total tardiness optimization? For the problem with 5 machines: (2) Is DQN able to interpolate the learned knowledge on problem instances with 5 or 10 jobs when trained with 20 jobs? (3) Is DQN able to extrapolate the learned knowledge on problem instances with 10 or 20 jobs, when trained on 5 jobs?

In all scenarios, we trained our agents sequentially, with the agent for machine allocation decisions always being trained first. Agents responsible for machine allocation decisions have been trained by selecting operation sequences randomly. When training agents for selecting operation sequences, we always deployed an agent for allocating jobs to machines. Table 3 shows the results of our experiments.

Table 3: Computational results of DQN trained on different problem instances compared to the GRASP algorithm of (Rajkumar et al. 2010).

Problem / Method	5M × 5J		5M × 10J		5M × 20J		8M × 8J	
	C_{Max}	T	C_{Max}	T	C_{Max}	T	C_{Max}	T
GRASP algorithm	242	365	421	2025	924	2919	253	908
DQN (5M × 5J)	186	295	405	2337	768	3022	-	-
DQN (5M × 10J)	193	345	342	1874	977	2475	-	-
DQN (5M × 20J)	218	473	435	1902	676	700	-	-
DQN (8M × 8J)	-	-	-	-	-	-	201	465

The results show that DQN always find a better solution than the GRASP algorithm for every problem DQN has been trained. Furthermore, DQN also performs quite well on problem instances other than the training data. As expected, the agents are better at interpolating training data than extrapolating from it. The results are achieved after 5000 episodes (i.e. simulation experiments) of training for each agent, where the training of the agent for machine allocations requires approximately 25 minutes, while the training of the other agent has been completed after 35 minutes. However, as the learning curves in figure 4 show, the best rewards has been already achieved after 3000 episodes and 100 episodes. Thus, the real training effort is much less as the computational time indicates. The training was carried out on a workstation with 6 × 3.7 GHz CPU and a GTX 1080 GPU.

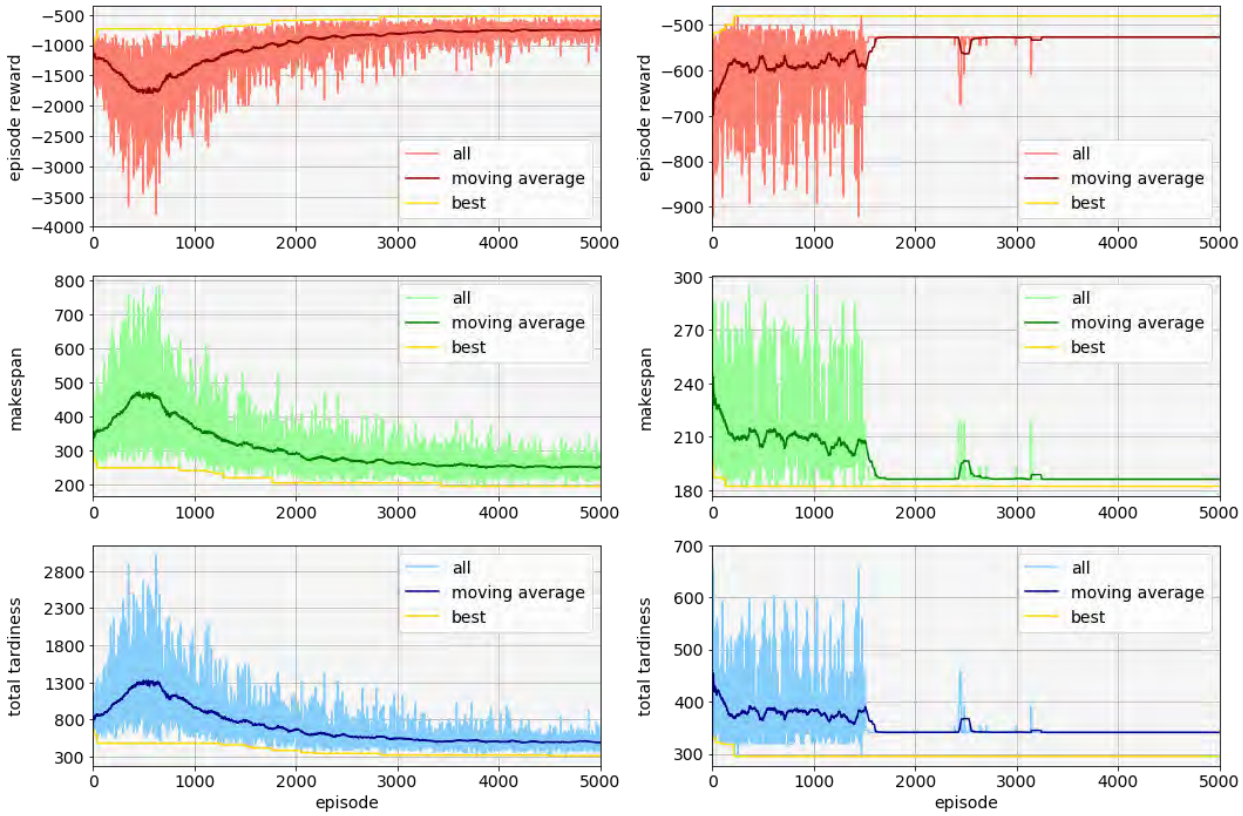

Figure 4: Training metrics for learning (a) the allocation of jobs to machines and (b) the selection of operation sequences.

Once the training is completed, the prediction and evaluation of new production schedules requires less than 0.1 seconds. Unfortunately, we do not have any information about the computational time of the compared GRASP algorithm. However, Rajkumar et al. (2010) state that the GRASP algorithm is a metaheuristic, which are usually not real-time capable.

6 CONCLUSION AND OUTLOOK

In this work, we examined the application of DQN to schedule jobs in a flexible job shop with integrated process planning. Our experiments showed that DQN is able to compute high-quality solutions in real-time with comparatively little training effort. In particular, the trained agents are further able to compute sufficient solutions for new problem instances, which indicates that the agents were able to generalize the learned data to a certain degree.

During our research, however, we faced many challenges that require further investigations. As a major problem, we consider the sensitivity of DQN to changes of any parameter. As discussed in section 4.3, the creation of a suitable network architecture involves many design decisions, which are worth to analyze in a separate paper. In the future, we want to investigate tools for automatic hyper-parameter tuning, as for instance AutoKeras (Jin et al. 2019), thus to improve the quality, convergence and the stability of the trainings process. Furthermore, the design of rewards has a significant influence on the training. During our research, we tried many reward functions that are unreported, before we identified something that worked. Many of the investigated reward functions were much more complex, but gave poorer results. At this moment, it is too early to judge whether or not this observation can be generalized. Furthermore, there are other aspects having a significant influence on the learning of an agent, which are anything but obvious. We were, for instance, surprised about the influence of the scaling method for preprocessing input states of an agent. In previous projects, the Min-Max scaling method, which normalize data in a range between a minimum and maximum (typical 0 and 1 in machine learning), has proven to perform well for most of the problems. However, it turned out that the agent were not able to learn, when applying Min-Max scaling for our input states. In contrast, scaling by dividing all features by 1000 has greatly improved the training. In summary, it is necessary to conduct experiments with different reward functions and to perform extensive parameter studies, in order to develop a methodological approach for using deep reinforcement learning to solve job shop scheduling problems.

It is further worth to investigate, whether a third agent that is responsible for the sequencing of jobs in front of machines can further improve the solution quality. However, we do not believe that DQN is a suitable algorithm for sequencing, since the action space of DQN is fixed. In contrast, the possible positions of a job in a sequence depends on the current number of jobs in the buffer and thus require a flexible action space. As proposed in our previous paper (Lang et al. 2020), a possible approach is the training of an agent with a single output, where the activation of the output can be interpreted as a sequence priority. In the past, we applied NeuroEvolution of Augmenting Topologies to train neural networks with one output neuron. In future work, however, we also want to investigate policy-based deep reinforcement learning strategies, such as deep deterministic policy gradients, to teach an agent the sequencing of jobs.

Considering the motivation of our research, it is clear that the investigated problem is far from the complexity of a matrix production. In general, DRL methods are able to learn policies for stochastic and highly dynamic environments, such as Atari Games. Against this background, the recent paper represents just a first proof of concept that DQN is able to train agents that compute solutions of reasonable quality for job-shop scheduling problems. The next step will be to consider stochastics in our model, such as random inter-arrival times, noisy process times and machine breakdowns, thus to investigate whether DQN is able to learn meaningful scheduling policies for highly dynamic production systems.

REFERENCES

Arulkumaran, K., M. P. Deisenroth, M. Brundage, and A. A. Bharath. 2017. "Deep Reinforcement Learning: A Brief Survey". *IEEE Signal Processing Magazine* 34(6):26–38.

Aydin, M.E., and E. Öztemel. 2000. "Dynamic Job-Shop Scheduling using Reinforcement Learning Agents". *Robotics and Autonomous Systems* 33(2-3):169–178.

Baker, K. R., and D. Trietsch. 2009. *Principles of Sequencing and Scheduling*. Hoboken, N.J: John Wiley. https://onlinelibrary.wiley.com/doi/book/10.1002/9781119262602.

Bauernhansl, T., M. ten Hompel, and B. Vogel-Heuser. 2014. *Industrie 4.0 in Produktion, Automatisierung und Logistik*: *Anwendung, Technologien, Migration*. Wiesbaden: Springer Vieweg. http://dx.doi.org/10.1007/978-3-658-04682-8.

Brucker, P. 2007. *Scheduling Algorithms.* 5th ed. Berlin, New York: Springer.

Dahl, O.-J., and K. Nygaard. 1966. "SIMULA: An ALGOL-based Simulation Language". *Communications of the ACM* 9(9):671–678.

François-Lavet, V., P. Henderson, R. Islam, M. G. Bellemare, and J. Pineau. 2018. "An Introduction to Deep Reinforcement Learning". *Foundations and Trends in Machine Learning* 11(3-4):219–354.

Gershwin, S. B. 2018. "The Future of Manufacturing Systems Engineering". *International Journal of Production Research* 56(1-2):224–237.

Greschke, P., M. Schönemann, S. Thiede, and C. Herrmann. 2014. "Matrix Structures for High Volumes and Flexibility in Production Systems". *Procedia CIRP* 17:160–165.

Jin, H., Q. Song, and X. Hu. 2019. "Auto-Keras: An Efficient Neural Architecture Search System". In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining,* August 4th -7th , Anchorage, AK, USA, 1946–1956.

Lang, S., T. Reggelin, F. Behrendt, and A. Nahhas. 2020. "Evolving Neural Networks to Solve a Two-Stage Hybrid Flow Shop Scheduling Problem with Family Setup Times". In *Proceedings of the 53rd Hawaii International Conference on System Sciences (HICSS 2020),* January 7th -10th , Wailea, HI, 1298–1307.

Lang, S., M. Schenk, and T. Reggelin. 2019. "Towards Learning- and Knowledge-Based Methods of Artificial Intelligence for Short-Term Operative Planning Tasks in Production and Logistics: Research Idea and Framework". *IFAC-PapersOnLine* 52(13):2716–2721.

Lin, C.-C., D.-J. Deng, Y.-L. Chih, and H.-T. Chiu. 2019. "Smart Manufacturing Scheduling With Edge Computing Using Multiclass Deep Q Network". *IEEE Transactions on Industrial Informatics* 15(7):4276–4284.

Luo, S. 2020. "Dynamic Scheduling for Flexible Job Shop with New Job Insertions by Deep Reinforcement Learning". *Applied Soft Computing* 91:106208.

Martínez, Y., A. Nowé, J. Suárez, and R. Bello. 2011. "A Reinforcement Learning Approach for the Flexible Job Shop Scheduling Problem". In *Learning and Intelligent Optimization*: *5th International Conference, LION 5, Rome, Italy, January 17 - 21, 2011 ; Selected Papers,* edited by C. A. Coello Coello, 253–262. Berlin: Springer.

Mnih, V., K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis. 2015. "Human-Level Control Through Deep Reinforcement Learning". *Nature* 518(7540):529–533.

Mnih, V., K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller. 2013. "Playing Atari with Deep Reinforcement Learning". Technical Report, NIPS Deep Learning Workshop 2013, Deepmind Technologies. http://arxiv.org/pdf/1312.5602v1.

Paternina-Arboleda, C. D., and T. K. Das. 2005. "A Multi-Agent Reinforcement Learning Approach to Obtaining Dynamic Control Policies for Stochastic Lot Scheduling Problem". *Simulation Modelling Practice and Theory* 13(5):389–406.

Pinedo, M. L. 2016. *Scheduling*. Cham: Springer International Publishing.

Qu, S., J. Wang, S. Govil, and J. O. Leckie. 2016. "Optimized Adaptive Scheduling of a Manufacturing Process System with Multi-skill Workforce and Multiple Machine Types: An Ontology-based, Multi-agent Reinforcement Learning Approach". *Procedia CIRP* 57:55–60.

Rajkumar, M., P. Asokan, T. Page, and S. Arunachalam. 2010. "A GRASP Algorithm for the Integration of Process Planning and Scheduling in a Flexible Job-Shop". *International Journal of Manufacturing Research* 5(2):230.

Shi, D., W. Fan, Y. Xiao, T. Lin, and C. Xing. 2020. "Intelligent Scheduling of Discrete Automated Production Line via Deep Reinforcement Learning". *International Journal of Production Research* :1–19.

Shiue, Y.-R., K.-C. Lee, and C.-T. Su. 2018. "Real-Time Scheduling for a Smart Factory using a Reinforcement Learning Approach". *Computers & Industrial Engineering* 125:604–614.

Shufang, X., T. Zhang, and O. Rose. 2019. "Online Single Machine Scheduling Based on Simulation and Reinforcement Learning". In *Simulation in Produktion und Logistik 2019,* edited by M. Putz and A. Schlegel, 59–68. Auerbach, Germany: Verlag Wissenschaftliche Scripten.

Stricker, N., A. Kuhnle, R. Sturm, and S. Friess. 2018. "Reinforcement Learning for Adaptive Order Dispatching in the Semiconductor Industry". *CIRP Annals* 67(1):511–514.

Sutton, R. S., and A. Barto. 2018. *Reinforcement Learning*: *An Introduction*. Cambridge, MA, London: The MIT Press.

van der Ham, R. 2018. "Salabim: Discrete Event Simulation and Animation in Python". *Journal of Open Source Software* 3(27):767–768.

Wang, Y.-C., and J. M. Usher. 2005. "Application of Reinforcement Learning for Agent-based Production Scheduling". *Engineering Applications of Artificial Intelligence* 18(1):73–82.

Waschneck, B., A. Reichstaller, L. Belzner, T. Altenmüller, T. Bauernhansl, A. Knapp, and A. Kyek. 2018. "Optimization of Global Production Scheduling with Deep Reinforcement Learning". *Procedia CIRP* 72:1264–1269.

Watkins, C. J. C. 1989. *Learning from Delayed Rewards.* Ph.D. thesis, *King's College,* University of Cambridge, Cambridge, UK. http://www.cs.rhul.ac.uk/~chrisw/new_thesis.pdf, accessed 1st April 2020.

Xie, J., L. Gao, K. Peng, X. Li, and H. Li. 2019. "Review on Flexible Job Shop Scheduling". *IET Collaborative Intelligent Manufacturing* 1(3):67–77.

Zhang, T., S. Xie, and O. Rose. 2018. "Real-Time Batching in Job Shops based on Simulation and Reinforcement Learning". In *Proceedings of the 2018 Winter Simulation Conference (WSC'18),* edited by M. Rabe, A. A. Juan, N. Mustafee, A. Skoogh, S. Jain, and B. Johansson, 3331–3339. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers, Inc.

Zhang, W., and T. G. Dietterich. 1995. "A Reinforcement Learning Approach to Job-Shop Scheduling". In *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence,* edited by C. S. Mellish, 1114–1120. San Mateo, Calif.: Morgan Kaufmann.

AUTHOR BIOGRAPHIES

SEBASTIAN LANG is research fellow at Fraunhofer Institute for Factory Operation and Automation IFF. He holds a master's degree in mechanical engineering with focus on production technologies and a master's and bachelor's degree in industrial engineering and logistics. His research interests include studying and applying methods of machine learning and artificial intelligence, simulation modeling and mathematical optimization for problems in production and logistics. His e-mail address is sebastian.lang@iff.fraunhofer.de. His ResearchGate profile is https://www.researchgate.net/profile/Sebastian_Lang5.

NICO LANZERATH is a master student at the Otto von Guericke University Magdeburg. He holds a bachelors's degree in industrial engineering and logistics. His research interests include modeling and simulation of production and logistic systems and the investigation of machine learning methods for improving production and logistics processes. His e-mail address is nico.lanzerath@st.ovgu.de.

TOBIAS REGGELIN is a project manager, researcher and lecturer at Otto von Guericke University Magdeburg and Fraunhofer Institute for Factory Operation and Automation IFF Magdeburg. His main research and work interests include modeling and simulation of production and logistics systems and developing and applying logistics management games. Tobias Reggelin received a doctoral degree in engineering from Otto von Guericke University Magdeburg. Furthermore, he holds a master's degree in Engineering Management from Rose-Hulman Institute of Technology in Terre Haute, IN and a diploma degree in Industrial Engineering in Logistics from Otto von Guericke University Magdeburg. His email address is tobias.reggelin@ovgu.de.

MARCEL MÜLLER is a research fellow at the Otto von Guericke University Magdeburg. He earned his master's degree in Industrial Engineering for Logistics at the Otto von Guericke University Magdeburg. His research interests include modeling and simulation of logistics systems and handling of deadlocks. His email address is marcel1.mueller@ovgu.de. His website is https://www.ilm.ovgu.de/mueller.

FABIAN BEHRENDT is a full professor for industrial engineering at SRH Fernhochschule - The Mobile University with focus on distance learning in subject area production and logistics. Moreover, he is head of research management at Fraunhofer Institute for Factory Operation and Automation IFF and responsible for research strategies. He holds a doctoral degree in engineering and a diploma in industrial engineering. His research is dedicated to the study of intelligent logistics zones and new technologies of Industry 4.0. His email address is fabian.behrendt@iff.fraunhofer.de.