Deep Reinforcement Learning for Semiconductor Production Scheduling

Bernd Waschneck
GSaME, Universität Stuttgart
Nobelstr. 12, 70569 Stuttgart, Germany
bernd.waschneck@gsame.uni-stuttgart.de

André Reichstaller
Institute for Software & Systems Engineering
University of Augsburg
Germany

Lenz Belzner
Lenz Belzner AI Consulting
Munich, Germany

Thomas Altenmüller
Infineon Technologies AG
Am Campeon 1-12,
85579 Neubiberg, Germany

Thomas Bauernhansl
Fraunhofer Institute for Manufacturing Engineering and Automation IPA
Nobelstr. 12, 70569 Stuttgart, Germany

Alexander Knapp
Institute for Software & Systems Engineering
University of Augsburg
Germany

Andreas Kyek
Infineon Technologies AG
Am Campeon 1-12,
85579 Neubiberg, Germany

***Abstract*—Despite producing tremendous success stories by identifying cat videos [1] or solving computer as well as board games [2], [3], the adoption of deep learning in the semiconductor industry is moderatre. In this paper, we apply Google DeepMind's *Deep Q Network (DQN)* agent algorithm for *Reinforcement Learning (RL)* to semiconductor production scheduling. In an RL environment several cooperative DQN agents, which utilize deep neural networks, are trained with flexible user-defined objectives. We show benchmarks comparing standard dispatching heuristics with the DQN agents in an abstract frontend-of-line semiconductor production facility. Results are promising and show that DQN agents optimize production autonomously for different targets.**

***Index Terms*—Production Scheduling, Reinforcement Learning, Machine Learning, Semiconductor Manufacturing**

I. Introduction

The three traditional efficiency improvement methods in semiconductor manufacturing (miniaturization, yield improvement and larger wafer sizes) are close to be fully exploited. At the same time, the Internet of Things requires a broad range of customized chips in smaller production quantities, e.g. sensors, which do not benefit from Moore's law. The key to meet cost reduction targets is operational excellence. In this context, we present a new method for production control which satisfies new requirements due to large portfolios with small quantities of each product. This mode of operation differs tremendously compared to traditional portfolios dominated by large volume logic and memory chips.

For small volumes at workcenters production scheduling in flexible job shops can be solved optimally with mathematical optimization. For larger, dynamic environments the model complexity and run-time limit the application of mathematical optimization to the *Job-Shop Scheduling Problem (JSP)*, which is *Non-deterministic Polynomial-time (NP)* hard. As a result, optimization is used locally and separated at workcenters. In complex job shops, local optimization of production scheduling can result in non-optimal global solutions for production.

A new method to increase performance in production control is the use of new technologies like machine learning. Deep Learning has made tremendous progress in the last years and produced various success stories [1], [2], [3], [4]. Still, there are very few serious applications in the semiconductor manufacturing industry.

In this paper we apply deep *Reinforcement Learning (RL)* to production scheduling in complex job shops utilizing cooperative *Deep Q Network (DQN)* agents [2]. The DQN agents, which use deep neural networks for decision making, are trained in a RL environment with user-defined flexible objectives to optimize production scheduling. The rules at one workcenter are optimized by one agent while optimizing a global reward and monitoring the actions of other agents. In the simulation the rules are directly tested and improved. During a pre-training phase with e.g. data from legacy systems, the DQN system captures existing dispatching strategies such as heuristics in neural networks. Of course, completely new solutions can also be trained in the simulation environment. With this application of deep RL, we achieve the Industrie 4.0 vision for production control of a decentralized, self-learning, self-organizing and self-optimizing system. The approach has several advantages:

- **Global transparency:** The composition of different hierarchical dispatching heuristics at workcenters is based on human experience. It works in a strictly discrete space/hierarchy and balancing of goals is only possible in a hierarchical way. The DQN agents work in a continuous space where they ensure that the right balance of objectives is met.

This work was supported by Infineon Technologies AG. A part of the work has been performed in the project *Power Semiconductor and Electronics Manufacturing 4.0 (SemI40)*, under grant agreement No 692466. The project is co-funded by grants from Austria, Germany, Italy, France, Portugal and the *Electronic Component Systems for European Leadership Joint Undertaking (ECSEL JU)*. This work was supported as part of the joint undertaking SemI40 by the German Federal Ministry of Education and Research under the grant 16ESE0074.

- **Continuity:** The DQN agents can be pre-trained from different, existing dispatching systems. Errors in existing dispatching systems are revealed. Legacy systems in production can be modernized and unified easily.
- **Flexibility:** Agents can be retrained and deployed within hours for different portfolios and changes in the optimization objectives (e.g. time-to-market vs. utilization) or in the machine park.
- **Global optimization:** Breaking down and balancing global goals to local *Key Performance Indicators (KPIs)* is challenging in complex job shop environments. The DQN agent system automatically optimizes globally instead of locally. It is not necessary to break down production objectives manually.
- **Automation:** Dispatching rules do not have to be implemented by human experts.

Despite all advantages, there are currently two disadvantages: First, training is computationally expensive. Second, as neural networks are black box models, it is hard to predict how the DQN agents act in unknown situations.

One of the first successful applications of RL and a neural network to a static job shop scheduling was published by Zhang and Dietterich [5], [6]. Mahadevan and Theocharous optimize the maintenance schedule of one machine with RL [7] and subsequently extended their model to a transfer line [8]. Bradtke and Duff solved the routing to two heterogeneous servers in order to minimize queue length with RL [9]. One agent is trained to adopt the dispatching policy of one machine in a three-resource scenario in [10]. Paternina-Arboleda and Das implement dynamic scheduling of multiple products at a single server [11]. A multi-agent learning approach for multi-machine scheduling is used by Brauer and Weiss, but without RL and neural networks [12]. Another approach utilizes neural networks and RL to optimize one resource center without constraints [13]. Recently, multi-agent RL was implemented with multiple machine types and Q-learning [14].

Since these publications, deep learning has made immense progress [15]. Promising results have been achieved with deep RL for resource management such as computing or memory resources [16], [17]. We apply a recent algorithm (DQN agent) in a multi-agent setting to a dynamic, complex job shop consisting of workcenters with different constraints, multiple machines of different types and multiple products.

In Section II the basics of job shop scheduling are defined. Section III presents the deep RL system for production scheduling. In Section IV, the factory environment used for validation is described and characterized. The method and test bed will be published in [18] in detail. Results on delivery reliability are presented for the first time. Discussion and conclusion follow in Sections V and VI.

II. Problem statement: Complex Job Shop Production and Scheduling

We apply machine learning to a production environment which is considered complex and dynamic. A job shop is an elementary type of manufacturing, where similar production devices form closed units. Different products p in the production portfolio take different routes r_p, which consist of a number of N ordered *Single Process Steps* ($SPS_{p,1}, \ldots, SPS_{p,N}$). The loading l_p, which gives the number of jobs of product p started in production per week, determines the production quantity. Each *SPS* is handled on a specific resource in a resource pool of M resources for a certain duration. The dedication matrix d determines the possible allocation of jobs to machines:

$$d_{(p,n),m} = \begin{cases} 1 & \text{if machine } m \text{ can process } SPS_{p,n} \\ 0 & \text{if machine } m \text{ can not process } SPS_{p,n}. \end{cases} \quad (1)$$

In a flexible job shop processes can be handled by several equipment, which is achieved by similar tools working in parallel. Several conditions and constraints have to apply to characterize a job shop as complex:

- **Technological constraints:** Sequence-dependent setup times, varying process times, time coupling, different types of processes (e.g. single jobs vs. batch processing).
- **Logistic constraints:** Re-entrant flows of the jobs, different lot sizes, prescribed due dates of the jobs, varying availability of tools (e.g. machine breakdowns).
- **Production quantity:** In a mass production emergent phenomena become visible as a result of interacting jobs (e.g. *Work In Progress (WIP)* waves).

Dispatching and scheduling control the performance of a complex job shop as a manufacturing system concerning logistic and economic KPIs [19]. Scheduling refers to the static planning process of allocating waiting lots to available resources [20]. Most research was done on the static problem, while real-world environments have continuous ongoing processes with constantly updated real-time information [19]. Dispatching (or dynamic scheduling) refers to the real-time decision upon the next job at a specific machine in a complex, dynamic environment [19], [20]. Schedules are determined mostly by linear optimization or genetic programming; heuristics are the most common method for dispatching (details for dispatching heuristics see [20]).

III. Methods: Application of RL to Production Scheduling

A. Production Scheduling as Markov Decision Process

The foundation for RL is an environment in which an agent can take actions and observe the results. The factory simulation, which serves as environment, runs as *Discrete-Event Simulation (DES)*, where events occur in an ordered sequence and mark changes in the system. Two types of events are distinguished: Events which do not require scheduling are handled internally, while the remaining events require the dispatching decision as external input. In the following, only dispatching events are considered. Thereby, a new discretization of scheduling time steps t is introduced. The state of the system $s_t \in S$ at time t, where S is the space of all possible states, is given to a dispatching system as input. This system provides an action $a_t \in A$, where A is the space of all possible actions available to the system. The event types which require

scheduling are ARRIVAL of a new lot and MOVEOUT of a lot from a machine.

For RL, states and actions need to fulfill the requirements of a *Markov Decision Process (MDP)*. The Markov property in RL is equivalent to the requirement that all relevant information for the decision is contained in the state vector s_t (for complete definition of MDP see [21, p. 57]).

The state space $S = S_{\text{machines}} \times S_{\text{jobs}}$ is a combination of machine states $\mathbf{s}_{\text{machine}} = \langle s_1, \ldots, s_{M_w} \rangle \in S_{\text{machines}}$ for M_w machines at a workcenter w and the state of surrounding jobs $\mathbf{s}_{\text{jobs}} = \langle s_1, \ldots, s_j \rangle \in S_{\text{jobs}}$ for j jobs. The machine space is defined by machine availability (breakdowns), capabilities and the setup. Machine capabilities are encoded in the dedication matrix d (see Section II). Availability av is a binary property, $av \in \{0,1\}^{M_w}$. In this example, the equipment at one workcenter is identical and breakdowns are not explicitly considered. This leaves the setup, which is encoded for the workcenter specific agent: The machine state s_x reduces to a one-hot vector $s_x \in \{0,1\}^{ST}$ for *ST* setup types.

The second part of the state space consists of the properties of the jobs s_j. First, it comprises the product type, which is a one-hot vector $\{0,1\}^p$. Then, the normalized deviation of a set due date for the current operation is given. Last, the location is encoded as one-hot vector $\{0,1\}^{\text{locations}}$.

The action space consists of $(pos{+}1)$ actions: the lots at pos possible positions and one option to start no lot. Before each call of a DQN agent lot positions are shuffled to randomize the samples in the training set.

B. Supervised and RL in a factory simulation

The default scheduling and dispatching logic is described in Section IV and implemented in an event handler called *Job Shop Management (JSM)*. The JSM is based on expert knowledge and is the benchmark for factory performance. It provides the state-action pairs (s_t, a_t) for supervised learning (see Fig. 1). The neural network predicts the action a_t based on the state s_t. Thereby, existing dispatching strategies are captured by observation of existing solutions.

Still, it is not possible to improve on the existing systems with supervised learning. During RL, the DQN agents interact directly with the factory simulation. The DQN agents receive rewards, in this case a factory KPI, and correlates actions a_t with rewards. The agent determines its actions by using a neural network or by choosing an explorative action to discover new strategies. Essentially, the agent trains the neural network in such a way that it predicts the cumulative, weighted rewards for all actions. The relationship between agent, neural network, JSM and simulation is shown in Fig. 1.

The DQN agents are based on Q-learning, which is used to adapt the action-selection policy $\pi_t(a|s)$ so that it maximizes the reward. The policy $\pi_t(a|s)$ is the probability distribution that $a_t = a$ if $s_t = s$ [21]. The Q-function $Q : S \times A \to \mathbb{R}$ expresses the reward over successive steps weighted by the

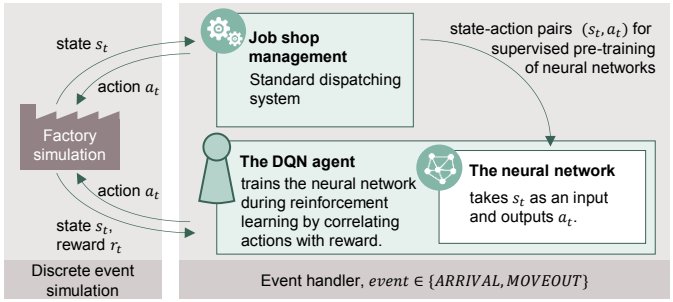

Fig. 1. Relationship and data exchange between factory simulation as discrete event simulation, job shop management with standard dispatching heuristics and the DQN agent with neural networks.

discount factor γ. The optimal action-value function $Q^*(s,a)$ is approximated during the Q-learning algorithm: [2]

$$Q^*(s,a) = \max_{\pi} \mathbb{E}\left[\sum_t r_t \cdot \gamma^t \,|s_t = s, a_t = a, \pi\right]. \quad (2)$$

The function $Q^*(s,a)$ is the maximum of the sum of rewards discounted by γ per time-step t that can be achieved with the policy π. The DQN algorithm uses the neural network as a representation of π and also to predict the Q values for actions.

In our experiments, we experienced difficulties in capturing different dispatching strategies at different workcenters with different resources and constraints in one neural network. Additionally, one agent with a separate neural network for each workcenter improves scalability and stability. The agents are trained separately, but use the neural networks of the other agents for the remaining workcenters. This stabilizes the first learning phase. While all neural networks are controlling the simulation, only one agent is actively training its neural network. The learning agent considers the actions of the other agents by observing their activity. As all agents optimize a global reward they act cooperatively. The cooperative learning of different agents is presented in the upper part of Fig. 2.

The training of the DQN agents is separated into two phases:

- **Phase A:** While one DQN agent is trained, the other workcenters are controlled by heuristics. Each DQN agent is trained once.
- **Phase B:** All workcenters are controlled by DQN agents, which are learning separately. The DQN agents are trained in cycles, each time for a relatively short period.

The separation speeds up training for two reasons: First, the factory performance is stabilized if only one workcenter is controlled by an agent. Second, the heuristics at the remaining workcenters have a faster execution time than neural networks. Still, training can be started directly in phase B.

As DQN agents are model-free, they start without any knowledge about the system (if no prior supervised learning was done). A linear annealed ϵ-greedy policy controls the share of explorative actions the agent takes in different phases.

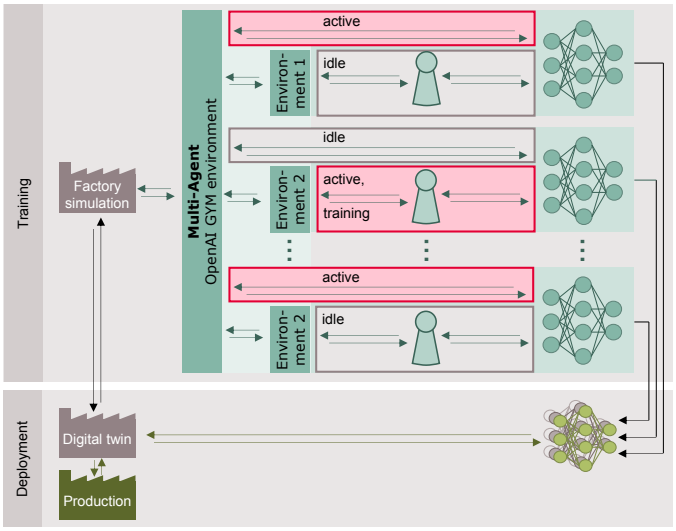

Fig. 2. Complete setup of the DQN agent based production control. The training phase in the upper part shows the sequential training algorithm for multi-agent systems. The deployment layer at the bottom demonstrates the fast transferability and applicability in the factory and the synchronization with the digital twin.

In a separate deployment phase, the performance is determined without changes due to the learning process and random actions due to the ϵ-greedy policy. The whole training and learning process is presented in Fig. 3.

C. Implementation and Application

The MATLAB factory simulation is made accessible via the MATLAB API for Python and an Python OpenAI Gym Interface [22], with which standardized agents can observe and control the environment. For the RL framework keras-rl [23] is used, which builds on keras [24]. The TensorFlow backend is used in Keras. In keras-rl, the implementation of Google DeepMind's DQN agent is used [2].

For an application in a factory, the performance of the system depends on the quality of the simulation model. A digital twin of the production offers the best way to let the RL algorithm interact with the production. If simulation and reality have significant deviations, the DQN-agents can keep learning after deployment in production to account for and adapt to differences.

During training of the algorithm, the solutions are not optimal. Furthermore, the training is computationally expensive and not running in real-time. Therefore the training of the algorithms runs offline in a simulation. When optimal solutions are found, the essence is captured in the neural networks. Next, the neural networks are transferred to the online environment. The neural networks can be updated when objectives, production resources, portfolios or logistics in the digital twin change.

IV. Characterization of the Factory Simulation

Semiconductor wafer processing is characterized as complex job shop production. The testbed factory is modeled after an abstracted frontend-of-line production with four workcenters, where transistors are formed on the wafer. The first workcenter is equipped with two lithography clusters, but only one reticle set for each product. In workcenter 2 the implanter requires different setups for different processes with stetup times shown in Table I. The next processing steps are merged in the testbed and modeled as a buffer with an infinite capacity but transport batching. In the last workcenter, 3 furnaces process batches of two identical lots.

TABLE I
Setup Times from one technology class to another.

[arbitrary time units]	TC 1	TC 2	TC 3
TC 1	0.00	0.11	0.19
TC 2	0.41	0.00	0.32
TC 3	0.13	0.30	0.00

Three different *Technology Classes (TCs)* are running in the simulation, on which different products can be realized. The *Raw Process Times (RPTs)* are given in Fig. 4. All RPTs follow a normal distribution with a coefficient of variance of 50%, which also models delays at machines. The implanter requires a different setup for each TC. Each lot re-enters the line for a fixed number of cycles creating a re-entrant flow. Transport-times and machine breakdowns are not considered explicitly.

Each workcenter is controlled by different dispatching heuristics which are typical for semiconductor manufacturing. At workcenter 1 *Operations Due Date (ODD)* with a plan *Flow Factor (FF)* is applied (also called X-factor; FF=*Cycle Time (CT)*/($\sum$RPT)). If ODD is ambiguous, *First-In-First-Out (FIFO)* is applied for lots with identical due dates. At workcenter 2, the setup optimization increases the throughput. For lots with identical setup, FIFO is used. Workcenter 3 acts as a buffer. At workcenter 4 the batch with the lot with the largest due date deviation is started. Single lots are not started.

The WIP level is kept constant at 48 lots by controlling the loading of the simulation in order to achieve a realistic, measured FF of 2.8. Small variations in the WIP level are created by a random period of 0–18 hrs between closing of a lot and loading of the next. The uptime utilization is 95%.

V. Experiment, Results and Discussion

The standard dispatching heuristics mix the optimization objectives of uptime utilization and delivery performance. The uptime utilization leads to a high throughput, but also higher deviations in the delivery performance and a high *Cycle Time Spread (CTS)*. CTS is the distribution of CTs for one TC. In this experiment, the DQN agent dispatching system is rewarded for delivery performance. While CT might increase, we expect the CTS to decrease (and therefore the delivery reliability to increase).

The plan FF is set to 2.8. Critical are lots which surpass their planned CT by 10% (lots with FF= 3.1), as they are seriously delayed.

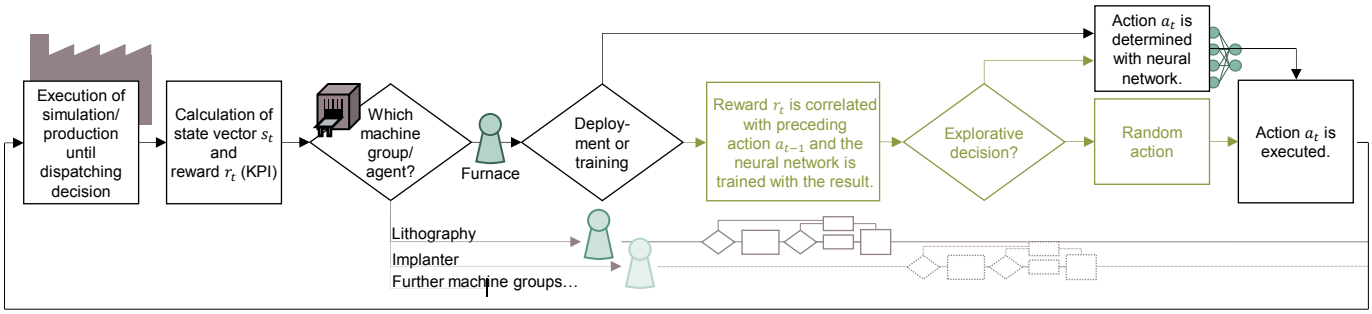

Fig. 3. Flow sequence of the training and deployment process for the DQN agent dispatching. Exploration and training steps are represented by the green boxes.

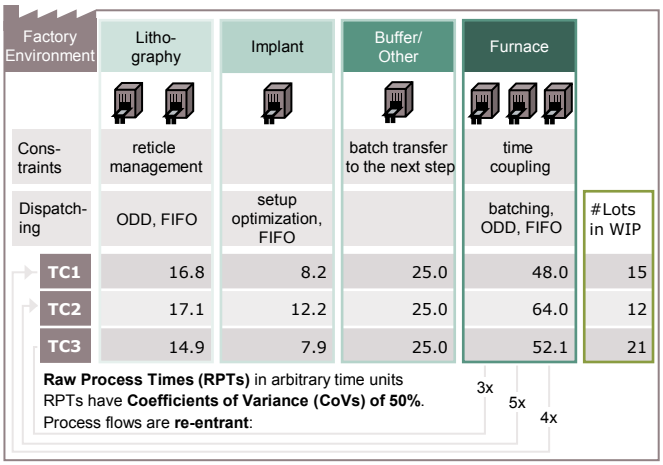

Fig. 4. Summary of the simulation model with three TCs and four workcenters with different constraints and dispatching heuristics.

The reward in phase A is considered to be a pre-training: It is the negative, normalized due date deviation of the chosen lot. Therefore the DQN agents are incentivized to choose the lot whose projected delivery date differs the most from the planned. In the second phase B, the global optimization starts: The reward is the total sum of the due date deviations of all lots in the line. For actions which are not possible to execute, e.g., when a reticle is already in use, a penalty of -1 is given. In deployment mode the penalties are set to zero.

The neural networks of each DQN agent have the same topology with three densely connected layers of 512, 128 and 21 neurons. The last layer corresponds to the actions. *Rectified Linear Units (ReLU)* are used as activation functions. Adam [25] is used as optimizer for training with a learning rate of $lr = 10^{-4}$ in phase A and $lr = 10^{-6}$ in phase B. The batch size bs is set to $bs = 32$ in phase A and $bs = 8$ in phase B. A decaying ϵ-greedy policy is used in phase A (shown in Fig. 5a); a constant ϵ-greedy policy of 0.2 is used in phase B. During deployment the best action is always selected. The target model update tmu of the DQN agent is set to $tmu = 10^{-2}$. The discount factor in Q-learning is $\gamma = 0.1$ in phase A and $\gamma = 0.9$ in phase B. Reference for the parameters is the publication of the DQN agent algorithm [2].

TABLE II
COMPARISON OF KEY PARAMETERS OF A FACTORY CONTROLLED BY DISPATCHING HEURISTICS OR DQN AGENT OPTIMIZATION IN DEPLOYMENT MODE AVERAGING OVER 10^5 LOTS PRODUCED WITH THE RESPECTIVE CONTROL METHOD.

		TC 1	TC 2	TC 3
Dispatching Heuristics	mean CT	1133.5	1600.1	838.0
	σ(CT)	118.2	138.9	102.5
	ratio lots w/ FF$>$ 3.1	17.0%	1.7%	13.9%
DQN agents	mean CT	1118.2	1729.5	840.5
	σ(CT)	56.3	60.9	54.5
	ratio lots w/ FF$>$ 3.1	1.3%	1.2%	1.9%

The key parameters of the learning process are shown in Fig. 5. In phase A a good pre-training is achieved: loss (Fig. 5a) and mean Q (Fig. 5b) functions are quickly converging. The mean Q ($= \text{mean}_t(\mathbb{E}(\max_a(Q(a,t))))$ value is converging towards Q^* (see Eq. 2). In phase A / Fig. 5b the reward is quickly rising.

The performance of each dispatching system is evaluated in deployment mode and results are shown in Table II and Fig. 6. While the mean CT for TC 1 and TC 3 is nearly identical for both systems, the DQN agent system leads to an increase of the mean CT for TC 2 of 8%. Still, the optimization meets its objective: The standard deviation of CT / CTS is decreased by 50% for all TCs. The ratio of lots with serious delay (FF$>$ 3.1) is decreased tremendously. The share of delayed lots in TC 1 is reduced from 17.0% to 1.3% and in TC 3 from 13.9% to 1.9%. With the standard dispatching heuristics the ratio of delayed lots for TC 2 is rather low. Interestingly, this share is only slightly increased, but the CT for TC 2 is increased. The gained flexibility is used to significantly increase the performance of TC 1 and 3. This shift is pronounced in the histogram Fig. 5.

With regards to the rapid developments in machine learning we expect the model to be able to scale to larger simulations. The factory size for which optimization is still possible is limited by the available processing power.

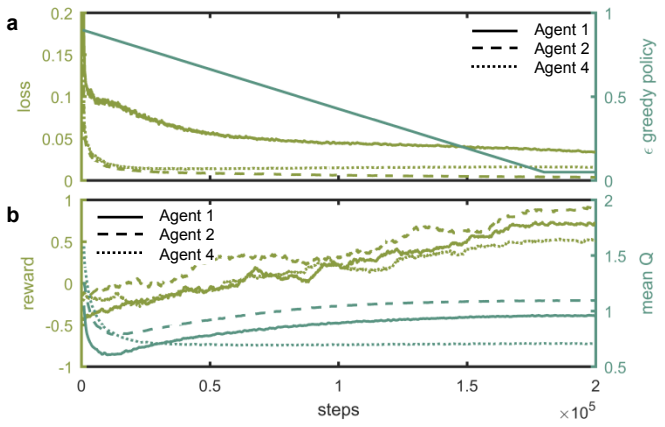

Fig. 5. Key parameters of the learning process. Agent 1, 2 and 4 correspond to the respective workcenters; workcenter 3 is not controlled by an agent as it acts mostly as a buffer. Steps are the time steps in the MDP. In (a), the loss functions of each agent as well as the ϵ parameter of the ϵ-greedy policy is shown (share of random actions to blend the training set, identical for all agents). The loss functions are converging quickly and set a good starting point for optimization of the whole line in phase B. In (b), the mean Q value is converging towards it's theoretical maximum of 1.11 for agent 2. Agent 1 and 4 at the respective workcenters cannot reach this value, as they are no bottlenecks and have more contraints on choosing the lots. The reward for all agents is increasing during the training process. For a definition of the machine learning parameters see [2].

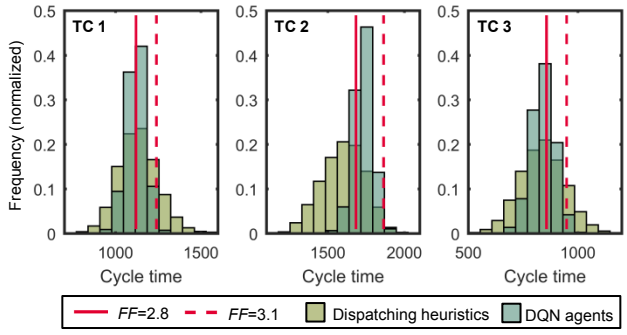

Fig. 6. *Cycle Time Spread (CTS)* of lots produced with the respective scheduling mechanisms. FF= 2.8 is the plan FF and FF= 3.1 is considered the critical FF in terms of delays and delivery performance.

VI. Summary and Conclusion

In this paper, RL with the DQN agent was successfully applied to production scheduling. The system automatically develops global optimal scheduling solutions without human intervention or any prior expert knowledge. It can be trained and exchanged within hours. The system adopts to the set objectives. In this case, the share of delayed lots could be reduced significantly, in TC 1 from 17% to 1.3%.

In future work, optimization under several balanced objectives will be presented. The impact of parameters in the RL process, e.g., size of the neural network, can be investigated. The methodology will be applied to different factory environments, where the performance in different settings and the scaling of the method can be investigated.

References

[1] Q. V. Le, "Building high-level features using large scale unsupervised learning," in *IEEE Int. Conf. on Acoustics, Speech and Signal Processing (ICASSP), 2013*, pp. 8595–8598, 2013.

[2] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, *et al.*, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, 2015.

[3] D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton, *et al.*, "Mastering the game of Go without human knowledge," *Nature*, vol. 550, no. 7676, pp. 354–359, 2017.

[4] A. Mordvintsev, C. Olah, and M. Tyka, "Inceptionism: Going deeper into neural networks," *Google Research Blog. Retrieved June*, vol. 20, p. 14, 2015.

[5] W. Zhang and T. G. Dietterich, "A reinforcement learning approach to job-shop scheduling," in *IJCAI*, vol. 95, pp. 1114–1120, 1995.

[6] W. Zhang and T. G. Dietterich, "High-performance job-shop scheduling with a time-delay td (λ) network," in *Advances in neural information processing systems*, pp. 1024–1030, 1996.

[7] S. Mahadevan, N. Marchalleck, T. K. Das, and A. Gosavi, "Self-improving factory simulation using continuous-time average-reward reinforcement learning," in *Machine learning international workshop*, pp. 202–210, Morgan Kaufmann Publishers, 1997.

[8] S. Mahadevan and G. Theocharous, "Optimizing production manufacturing using reinforcement learning.," in *FLAIRS Conference*, pp. 372–377, 1998.

[9] S. J. Bradtke and M. O. Duff, "Reinforcement learning methods for continuous-time Markov decision problems," in *Advances in neural information processing systems*, pp. 393–400, 1995.

[10] S. Riedmiller and M. Riedmiller, "A neural reinforcement learning approach to learn local dispatching policies in production scheduling," in *IJCAI*, vol. 2, pp. 764–771, 1999.

[11] C. D. Paternina-Arboleda and T. K. Das, "A multi-agent reinforcement learning approach to obtaining dynamic control policies for stochastic lot scheduling problem," *Simulation Modelling Practice and Theory*, vol. 13, no. 5, pp. 389–406, 2005.

[12] W. Brauer and G. Weiß, "Multi-machine scheduling – a multi-agent learning approach," in *Multi Agent Systems, 1998. Proceedings. International Conference on*, pp. 42–48, IEEE, 1998.

[13] T. Gabel and M. Riedmiller, "Scaling adaptive agent-based reactive job-shop scheduling to large-scale problems," in *Computational Intelligence in Scheduling, 2007. SCIS'07.*, pp. 259–266, IEEE, 2007.

[14] S. Qu, J. Wang, S. Govil, and J. O. Leckie, "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*, vol. 57, pp. 55–60, 2016.

[15] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, no. 7553, pp. 436–444, 2015.

[16] H. Mao, M. Alizadeh, I. Menache, and S. Kandula, "Resource management with deep reinforcement learning.," in *HotNets*, pp. 50–56, 2016.

[17] A. I. Orhean, F. Pop, and I. Raicu, "New scheduling approach using reinforcement learning for heterogeneous distributed systems," *Journal of Parallel and Distributed Computing*, 2017.

[18] B. Waschneck, A. Reichstaller, L. Belzner, T. Altenmüller, T. Bauernhansl, A. Knapp, and A. Kyek, "Optimization of global production scheduling with deep reinforcement learning," *Procedia CIRP*, 2018. in press.

[19] D. Ouelhadj and S. Petrovic, "A survey of dynamic scheduling in manufacturing systems," *Journal of scheduling*, vol. 12, no. 4, pp. 417–431, 2009.

[20] B. Waschneck, T. Altenmüller, T. Bauernhansl, and A. Kyek, "Production scheduling in complex job shops from an industry 4.0 perspective," in *SAMI@ iKNOW*, 2016.

[21] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*, vol. 1. MIT Press Cambridge, 1998.

[22] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, "Openai gym." arXiv:1606.01540, 2016.

[23] M. Plappert, "keras-rl." github.com/matthiasplappert/keras-rl, 2016.

[24] F. Chollet *et al.*, "Keras." https://keras.io, 2015.

[25] D. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.