Optimization of global production scheduling with deep reinforcement learning

Bernd Waschneck, André Reichstaller, Lenz Belzner, Thomas Altenmüller, Thomas Bauernhansl, Alexander Knapp, Andreas Kyek

Angaben zur Veröffentlichung / Publication details:

Waschneck, Bernd, André Reichstaller, Lenz Belzner, Thomas Altenmüller, Thomas Bauernhansl, Alexander Knapp, and Andreas Kyek. 2018. "Optimization of global production scheduling with deep reinforcement learning." *Procedia CIRP* 72: 1264–69. https://doi.org/10.1016/j.procir.2018.03.212.

Available online at www.sciencedirect.com

ScienceDirect

Procedia CIRP 72 (2018) 1264–1269

www.elsevier.com/locate/procedia

51st CIRP Conference on Manufacturing Systems

Optimization of global production scheduling with deep reinforcement learning

Bernd Waschneck[a,b,*], André Reichstaller[c], Lenz Belzner, Thomas Altenmüller[b], Thomas Bauernhansl[d], Alexander Knapp[c], Andreas Kyek[b]

[a] *Graduate School advanced Manufacturing Engineering (GSaME) - Universität Stuttgart, Nobelstr. 12, 70569 Stuttgart, Germany*
[b] *Infineon Technologies AG, Am Campeon 1-12, 85579 Neubiberg, Germany*
[c] *Institute for Software & Systems Engineering, University of Augsburg, Germany*
[d] *Fraunhofer Institute for Manufacturing Engineering and Automation IPA, Nobelstr. 12, 70569 Stuttgart*

* Corresponding author. Tel.: +49-160-96791228. *E-mail address:* bernd.waschneck@gsame.uni-stuttgart.com

Abstract

Industrie 4.0 introduces decentralized, self-organizing and self-learning systems for production control. At the same time, new machine learning algorithms are getting increasingly powerful and solve real world problems. We apply Google DeepMind's *Deep Q Network (DQN)* agent algorithm for *Reinforcement Learning (RL)* to production scheduling to achieve the Industrie 4.0 vision for production control. In an RL environment cooperative DQN agents, which utilize deep neural networks, are trained with user-defined objectives to optimize scheduling. We validate our system with a small factory simulation, which is modeling an abstracted frontend-of-line semiconductor production facility.

Keywords: Production Scheduling, Reinforcement Learning, Machine Learning in Manufacturing

1. Introduction

Deep Learning has made tremendous progress in the last years and produced success stories by identifying cat videos [1], dreaming "deep" [2] and solving computer as well as board games [3,4]. Still, there are hardly any serious applications in the manufacturing industry. In this paper we apply deep *Reinforcement Learning (RL)* to production scheduling in complex job shops such as semiconductor manufacturing.

Semiconductor manufacturers traditionally had a small product portfolio which was dominated mostly by logic and memory chips. The Internet of Things requires a broader range of customized chips like sensors in smaller production quantities. Most sensors and actuators do not benefit from Moore's law. Furthermore, the three traditional efficiency improvement methods in manufacturing, miniaturization, yield improvement and larger wafer sizes, are close to be fully exploited. This, as well as the new portfolio requirements, lead to a strong focus on operational excellence in the semiconductor industry.

For small problem sizes production scheduling in flexible job shops, such as segments of semiconductor frontend facilities, 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 a complex job shop, this local optimization of production scheduling can lead to non-optimal global solutions for the production.

In this paper cooperative *Deep Q Network (DQN)* agents [3] are used for production scheduling. The DQN agents, which utilize deep neural networks, are trained in an RL environment with flexible user-defined objectives to optimize production scheduling. Each DQN agent optimizes the rules at one workcenter while monitoring the actions of other agents and optimizing a global reward. The rules are directly tested and improved in the simulation. The system can be trained with data from legacy systems such as heuristics to capture their strategies in neural networks and import them into the simulation for further improvement. It is also possible to train completely new solutions 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 and self-optimizing system. The approach has several advantages:

- **Flexibility:** Agents can be retrained within hours e.g. for different portfolios or changes in the optimization objectives (e.g. time-to-market vs. utilization).
- **Global transparency:** The composition of different hierarchical dispatching heuristics at different workcenters is based on human experience. Heuristics (and production goals) are arranged in a hierarchy. The neural networks are not bound by these constraints and have more ways to model the right balance of objectives.

2212-8271 © 2018 The Authors. Published by Elsevier B.V.
Peer-review under responsibility of the scientific committee of the 51st CIRP Conference on Manufacturing Systems.
10.1016/j.procir.2018.03.212

- **Global optimization:** Breaking down and balancing global goals to local *(Key) Performance Indicators ((K)PIs)* 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.
- **Continuity:** The DQN agents can be pre-trained from existing disptaching systems. Errors in existing dispatching systems are revealed. Legacy systems and different systems in production can be modernized and unified easily.

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

In this paper we present the method of DQN agent based dispatching. In the validation, we focus on the automation aspect.

In section 1.1 the basics of job shop scheduling are defined and in section 1.2 related work is presented. Section 2 presents the deep RL system for production scheduling. In section 3, the factory environment used for validation is described and characterized. Results, discussion and conclusion follow in sections 4 and 5.

1.1. Problem statement: Complex Job Shop Production and Scheduling

For the application of machine learning we choose a production environment which is considered complex and dynamic. A job shop is an elementary type of manufacturing, where similar production devices are grouped in closed units. In a flexible job shop each processes can be handled by several tools, which is mostly achieved by identical tools working in parallel. Under certain constraints and conditions the flexible job shop is characterized as complex job shop:

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

Different products p in the production portfolio take different routes r_p, which consist of a number of N ordered *Single Process Steps* $r_p = (SPS_{p,1}, \dots, SPS_{p,N})$. Each SPS has to be 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:

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

Dispatching and scheduling are crucial to control the performance of a complex job shop as a manufacturing system concerning logistic and economic KPIs [5]. Scheduling refers to the static planning process of allocating waiting lots to available resources [6]. Research has focused on the the static problem, while real-world environments have continuous ongoing processes with constantly updated real-time information [5]. Dispatching (or dynamic scheduling) refers to the real-time decision upon the next job at a specific machine in a complex, dynamic environment [5,6]. Sometimes the dispatching decision follows a pre-defined schedule. Schedules are determined mostly by linear optimization or genetic programming; heuristics are the most common method for dispatching.

1.2. Related Work

Cooperative multi-agent learning has been applied successfully to several areas such as network management and routing, electricity distribution management and meeting scheduling in order to exploit the adaptive dynamics of the approach [7]. One of the first successful applications of RL with a neural network to a static job shop scheduling was presented by Zhang and Dietterich [8,9]. Mahadevan et al. use RL to optimize the **maintenance schedule of one machine** [10] and later extended their model to a transfer line [11]. Bradtke and Duff solved the routing to **two heterogeneous servers** minimizing queue length with RL [12]. One agent is trained to adopt the dispatching of one machine in a **three-resource scenario** in [13]. Paternina-Arboleda et al. implement a dynamic scheduling at a single server on **multiple products** [14]. Brauer and Weiss use a **multi-agent learning approach for multi-machine scheduling**, but without RL[15]. One approach uses neural networks and RL to optimize a resource center without constraints [16]. In recent work, a multi-agent RL approach was implemented with **multiple machine types** and Q-learning [17].

Since these publications, deep learning has seen tremendous developments increasing the power of the methods immensely [18]. New RL agents have solved problems where a few years ago humans seemed distinctly superior, such as the ancient game of Go [19]. Deep RL for resource management such as abstract computing or memory resources has shown promising results [20,21]. In this paper several instances of Google DeepMind's DQN agent, which offers a much more efficient learning algorithm able to develope complex strategies, are used to optimize production scheduling in a multi-agent setting. It is applied to a dynamic, complex job shop environment consisting of workcenters with different constraints, multiple machines of different types and multiple products.

2. Methods: Application of RL to Production Scheduling

2.1. Production Scheduling as Markov Decision Process

RL requires an environment in which an agent can take actions and observe the results. The factory simulation environment runs as *Discrete-Event Simulation (DES)*, where events occur in an ordered sequence and mark changes in the system. Two types of events can be distinguished: such events that require scheduling and such that do not. In the following only events are considered which require scheduling. These events introduce a new discretization of scheduling time steps t, which is coarser than the event sequence in the DES. The state of the system $s_t \in S$, where S is the space of all possible states, at time t is handed over to a dispatching system. This system provides its decision encoded in an action $a_t \in A$, where A is the space of all possible actions available to the system. The two event types which may require scheduling are `ARRIVAL` of a new lot and `MOVEOUT` of a lot from a machine.

In order to be used in RL states and actions need to fulfill the criteria of a *Markov Decision Process (MDP)*. For RL, the

This is a resupply of March 2023 as the template used in the publication of the original article contained errors. The content of the article has remained unaffected.

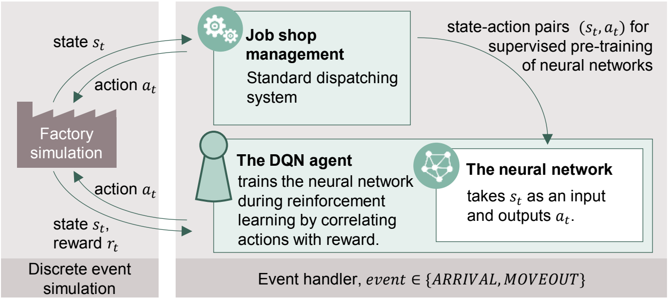

state s_t
action a_t
Factory simulation
action a_t
state s_t, reward r_t
Discrete event simulation
Job shop management
Standard dispatching system
state-action pairs (s_t, a_t) for supervised pre-training of neural networks
The DQN agent
trains the neural network during reinforcement learning by correlating actions with reward.
The neural network
takes s_t as an input and outputs a_t.
Event handler, $event \in \{ARRIVAL, MOVEOUT\}$

Fig. 1. Data exchange between factory simulation as a discrete event simulation, Job Shop Management System with standard dispatching heuristics and the DQN agent with neural networks.

Markov property is equivalent to the requirement that all relevant information for the decision is encoded in the state vector s_t (for complete definition of MDP see [22, 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 capabilities, availability (breakdowns) and the setup. Machine capabilities are described by the dedication matrix d (see Section 1.1). Availability av is in most cases binary, $av \in \{0,1\}^{M_w}$. In this example factory, all machines at one workcenter are identical and breakdowns are not explicitly considered. Therefore only the setup needs to be 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 for each machine.

The second part of the state space are the properties of the jobs s_j. Firstly, it comprises the product type, which is encoded in a one-hot vector $\{0,1\}^{\text{p}}$. The locations, which correspond to the workcenters, are also encoded as one-hot vector $\{0,1\}^{\text{locations}}$. Then the processing percentage of the product is included, which is a relative measure for the number of processing steps already completed. Last, the deviation of a set due date for an operation is given.

The action space consists of $pos + 1$ actions: the lots at pos possible positions in the and one option to not start a lot. Lot positions are shuffled before each call of a DQN agent to randomize the samples in the training set.

2.2. *Supervised and Reinforcement Learning in a factory simulation*

The default scheduling and dispatching logic is described in the next section and implemented in an event handler called *Job Shop Management (JSM)*. The JSM, which is based on expert knowledge, is the reference and benchmark for factory performance and provides the state-action pairs (s_t, a_t) for supervised learning (see Fig. 1). The neural network is trained to predict the action a_t based on the state s_t. With this setup, it is possible to capture existing dispatching strategies in a factory in neural networks by observation of existing solutions.

Still, with supervised learning it is not possible to improve on the existing systems. During RL, the DQN agents interact directly with the factory simulation, where they develop new strategies. The agent receives a reward, in this case a factory KPI, and correlates actions a_t with rewards. The agent determines its actions by using a neural network and mixing the output of the neural network with random actions to sample its training set. In essence, the agent trains the neural network in such a way that it predicts the cumulative, weighted rewards for all actions. The relationship between simulation, JSM, agent and neural network is shown in Fig. 1.

The DQN agents are based on Q-learning, which is used to optimize the action-selection policy $\pi_t(a|s)$ in such a way that it maximizes the reward. The policy $\pi_t(a|s)$ is the probability distribution that $a_t = a$ if $s_t = s$ [22]. The Q-function $Q : S \times A \to \mathbb{R}$ gives the reward over successive steps weighted by discount factor γ. The optimal action-value function $Q^*(s,a)$ is approximated in the course of the Q-learning algorithm: [3]

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

The function $Q^*(s,a)$ is the maximization of the sum of rewards discounted by γ per time-step t that can be achieved with the policy π. In the DQN algorithm the neural network is not only used as a representation of π but also to predict the Q values for actions.

Our experiments have shown difficulties in capturing different dispatching strategies at different workcenters with different resources and constraints in one neural network. In addition, one agent with a separate neural network is for each workcenter improves scalability and stability. The agents are trained separately, but can use the neural networks of the other agents for controlling the remaining workcenters. This stabilizes the first learning phase tremendously. All neural networks are controlling the simulation, but only one agent is actively training one neural network. The learning agent takes the actions of the other agents into account by observing their activity. As all agents optimize a global reward they act cooperatively. The cooperative learning of three different agents is shown in the upper part of Fig. 2.

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

- **Phase A:** While the one DQN agent is trained, the other workcenters are controlled by heuristics. 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 is used to diversify the samples of the agent. Each DQN agent is trained once.
- **Phase B:** All workcenters are controlled by DQN agents which are learning separately. The ϵ-greedy policy is set to a fixed value and the learning rate can be reduced. The DQN agents are trained in cycles, each time for a relatively short number of steps.

The separation speeds up training due to two reasons: First, the factory performance is stabilized if only one workcenter is agent-controlled. Second, the heuristics at the remaining workcenters can be executed faster than neural networks. Still, training can be started directly in Phase B and reach the same performance, but taking about four times as long as with Phase A.

In a separate deployment phase, the performance is determined without dynamic changes due to the learning process and random actions due to the ϵ-greedy policy.

2.3. *Implementation and Application*

The factory simulation is implemented in MathWorks MATLAB. In order to work with recent machine learning algorithms, the MATLAB API for Python is used to implement an OpenAI Gym Interface in Python towards the simulation [24]. The sim-

This is a resupply of March 2023 as the template used in the publication of the original article contained errors. The content of the article has remained unaffected.

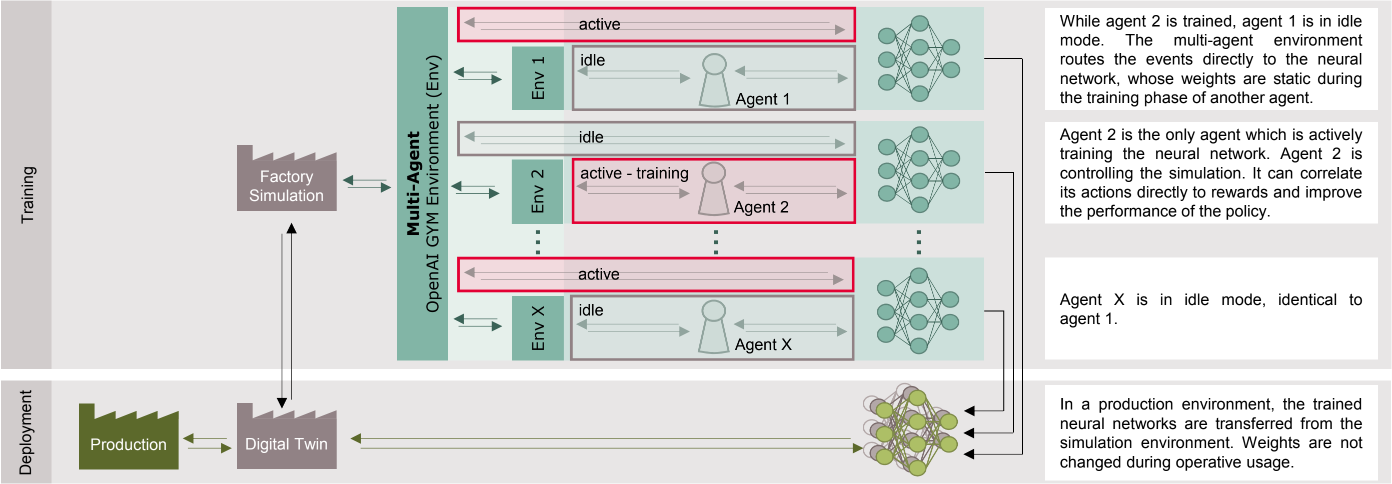

Training
Factory Simulation
Multi-Agent OpenAI GYM Environment (Env)
active
idle
Env 1
Agent 1
idle
active - training
Env 2
Agent 2
active
idle
Env X
Agent X
While agent 2 is trained, agent 1 is in idle mode. The multi-agent environment routes the events directly to the neural network, whose weights are static during the training phase of another agent.
Agent 2 is the only agent which is actively training the neural network. Agent 2 is controlling the simulation. It can correlate its actions directly to rewards and improve the performance of the policy.
Agent X is in idle mode, identical to agent 1.
Deployment
Production
Digital Twin
In a production environment, the trained neural networks are transferred from the simulation environment. Weights are not changed during operative usage.

Fig. 2. Complete setup of the DQN agent based production control. The training phase on top 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 [23].

ulation is imported as OpenAI Gym environment, which standardized agents can observe and control. For the RL framework keras-rl [25] is used, which is built on keras [26] and TensorFlow. In keras-rl, the implementation of Google DeepMind's DQN agent is used [3].

For an application in a factory, the performance of the framework depends tremendously on the quality of the simulation model. A digital twin of the production is optimal to let the RL algorithm interact with the production. Thereby training is separated from execution. The training of the algorithm, which is computationally expensive, not running real-time and possibly not always producing optimal solutions, runs offline in a simulation environment. When optimal solutions are found, the essence is captured in the neural networks, which are then transferred to the online environment. If simulation and reality have significant deviations, the DQN-agents can keep learning after deployment in production to account for and adapt to differences.

The execution of neural networks is fast, and in terms of most production processes it can be considered real-time. The system is running stable and predictable, as no learning is done in the running production. The neural networks can be updated regularly when portfolios, objectives, production resources or logistics in the digital twin change. The whole process of training and transferring neural networks to production is shown in Fig. 2.

3. Characterization of the Factory Simulation

Semiconductor wafer processing is characterized as complex job shop production. In the frontend-of-line the transistors are formed on the wafer. In the backend-of-line the metalization layers are processed which connect the transistors and create the logic interconnections. The factory simulation used for optimization is modeled after a frontend-of-line production.

The frontend-of-line workflow is modeled with four workcenters. The first workcenter is equipped with two lithography clusters. Each reticle for the lithography exposure is only available once, meaning that the machines can not process the same product at the same time. In workcenter 2 the implanter requires different setups shown in Table 1. The next steps are merged and modeled as a buffer with an infinite capacity but necessary transport batching. In the last workcenter 3 furnaces are located which take batches of two identical lots.

Table 1. Setup times for the setup change from one Technology Class (TC) to another.

[arbitrary time units]	TC 1	TC 2	TC 3
TC 1	0.0	1.1	1.9
TC 2	4.1	0.0	3.2
TC 3	1.3	3.0	0.0

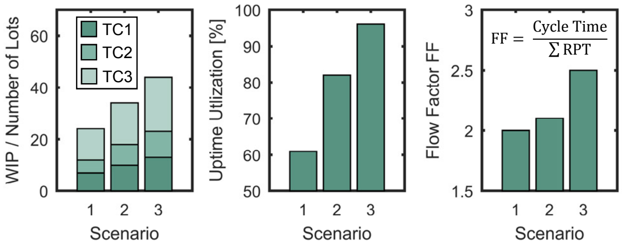
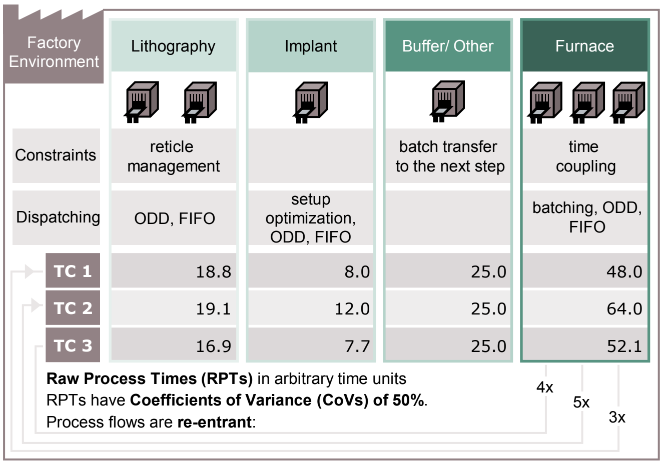


Fig. 3. Characterization of the scenarios in the factory simulation controlled by standard dispatching: loading/WIP, Uptime Utilization and Flow Factor.

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

Each workcenter is controlled by different, semiconductor typical dispatching heuristics. At workcenter 1 *Operations Due Date (ODD)* with a plan *Flow Factor (FF)* is applied (sometimes called X-factor; definition see Fig. 3). ODD ensures a continuous flow of production due to the underlying principle of a continuous speed in production (corresponding to a queueing time proportional to the RPT). If ODD is not decisive, *First-In-First-Out (FIFO)* is applied for lots with identical due dates. At workcenter 2, a hierarchy of three rules is applied: First setup optimization, then ODD among the lots eligible to run under setup constraints and last FIFO. Workcenter 3 acts as buffer with infinity capacity. At workcenter 4 the batch with the largest due date deviation is started. Single lots are not started.

This is a resupply of March 2023 as the template used in the publication of the original article contained errors. The content of the article has remained unaffected.



Factory Environment	Lithography	Implant	Buffer/ Other	Furnace
Constraints	reticle management		batch transfer to the next step	time coupling
Dispatching	ODD, FIFO	setup optimization, ODD, FIFO		batching, ODD, FIFO
TC 1	18.8	8.0	25.0	48.0
TC 2	19.1	12.0	25.0	64.0
TC 3	16.9	7.7	25.0	52.1

Raw Process Times (RPTs) in arbitrary time units
RPTs have **Coefficients of Variance (CoVs) of 50%**.
Process flows are **re-entrant**:

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

The dispatching heuristics are the benchmark for the RL dispatching. A detailed description of dispatching techniques can be found in [6].

For benchmarking a second dispatching system with a small random element is constructed. In this second reference system, a random action out of the action space will be executed with a probability of 30% at the workcenter 2.

Three loading scenarios corresponding to different FFs are evaluated. The WIP level is kept constant by controlling the loading of the simulation. Small variations in WIP are created by a random period of time of 0–18 hrs between closing of a lot and loading the next. WIP levels, *Uptime Utilization (UU)* and FF of the scenarios are presented in Fig. 3. UUs and FFs serve as reference to the RL dispatching model.

4. Experiment, Results and Discussion

In this experiment the UU (and therefore indirectly the throughput) is optimized. The rewards in phases A and B are given accordingly:

- **Reward phase A:** The number of lots in process at a workcenter divided by the total capacity (UU at workcenter) plus negative penalties. For actions which are not possible to execute, e.g., when a reticle is already in use, a penalty of -1 is given. If 80% of the total WIP are in queue at one workcenter, the dispatching heuristics are activated to avoid a crash. A penalty of -2 is given in this case.
- **Reward phase B:** The UU of the whole line (all machines in the factory) plus the negative penalties.

In deployment mode the penalties are set to zero.

The neural networks of each DQN agent have the same topology. The networks consist of three densely connected layers with 512, 128 and 18 neurons, where the last layer corresponds to the actions. The activation functions are *Rectified Linear Units (ReLU)*. The optimizer used for training is Adam [27] with a learning rate of $lr = 10^{-4}$ in phase A and $lr = 10^{-5}$ in phase B. A decaying ϵ-greedy policy is used in phase A (shown in Fig. 5); a constant ϵ-greedy policy of 0.3 is used in phase B. During deployment the optimal action is always selected. The target model update of the DQN agent is set to 10^{-2}. The batch size is set to 32. The discount factor in Q-learning is $\gamma = 0.9$. Reference for all parameters is the publication of the DQN agent algorithm [3] and the open source implementation of the DQN agent in keras-rl [25].

Table 2. Comparison of dispatching heuristics and DQN agent optimization in deployment mode. The presented reward values are the average reward over 25000 steps.

[Reward in test mode]	Scenario 1	Scenario 2	Scenario 3
Benchmark dispatching	0.61	0.83	0.96
Dispatching with 10% random at workcenter 2	0.58	0.79	0.91
Dispatching with 30% random at workcenter 2	0.55	0.73	0.85
DQN agent optimization	0.62	0.83	0.94

Results for the three scenarios are shown in Fig. 5. In phase A a good pre-training is achieved: loss (Fig. 5(a)) and mean Q (Fig. 5(b)) 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. 5(b) the reward is quickly rising. The local optimization (local reward) is easier to achieve than the line optimization (global reward, phase B). In phase B Fig. 5(c) the local potential is already exploited and the global reward/whole production line is further optimized. Due to this fine-tuning, The reward in Fig. 5(c) increases slowly and the Q-values of the DQN agents are converging (Fig. 5(c)). Although all agents get the same reward, the influence of random actions introduced by the ϵ-greedy policy at the workcenters is different. This explains the offset in the rewards in phase B (Fig. 5(c)). More information is given in the figure caption of Fig. 5.

The performance of each dispatching system is evaluated in deployment mode. The comparison of results is shown in Table 2. In all scenarios the DQN agent algorithm shows the same performance as the state of the art benchmark. The DQN dispatching system performs considerably better than the dispatching system with a small random element introduced at one machine. Introducing 30% random actions at only one workcenter decreases the UU over 10%.

The DQN agents are optimizing the factory simulation. For the deployment it is therefore crucial that the simulation models the properties of production correctly.

With regards to the rapid developments of machine learning in recent years we expect the model to be able to scale to larger simulations. The factory sizes for which optimization still is possible only depend on the available processing power.

5. Summary and Conclusion

In this paper a successful application of RL with the DQN agent to production scheduling was presented. The system automatically develops a scheduling solutions, which is on a par with the expert benchmark, without human intervention or any prior expert knowledge. While we do not beat the heuristics, we are able to reach the level of expert knowledge within 2 days of training. Non-optimal rules or errors in the implementation, such as the introduction of 30% random actions at workcenter 2, are detected. The system offers a high transparency due to the direct connection between solution and global optimization targets. It can be trained and exchanged within hours.

In future work, optimization under several balanced objectives will be shown. The methodology will be applied to different factory environments, where the performance in different settings and the scaling can be investigated. For strategic board games RL agents have been able to find new, formerly unknown strategies and outperform human grandmasters [4]. We hope

This is a resupply of March 2023 as the template used in the publication of the original article contained errors. The content of the article has remained unaffected.

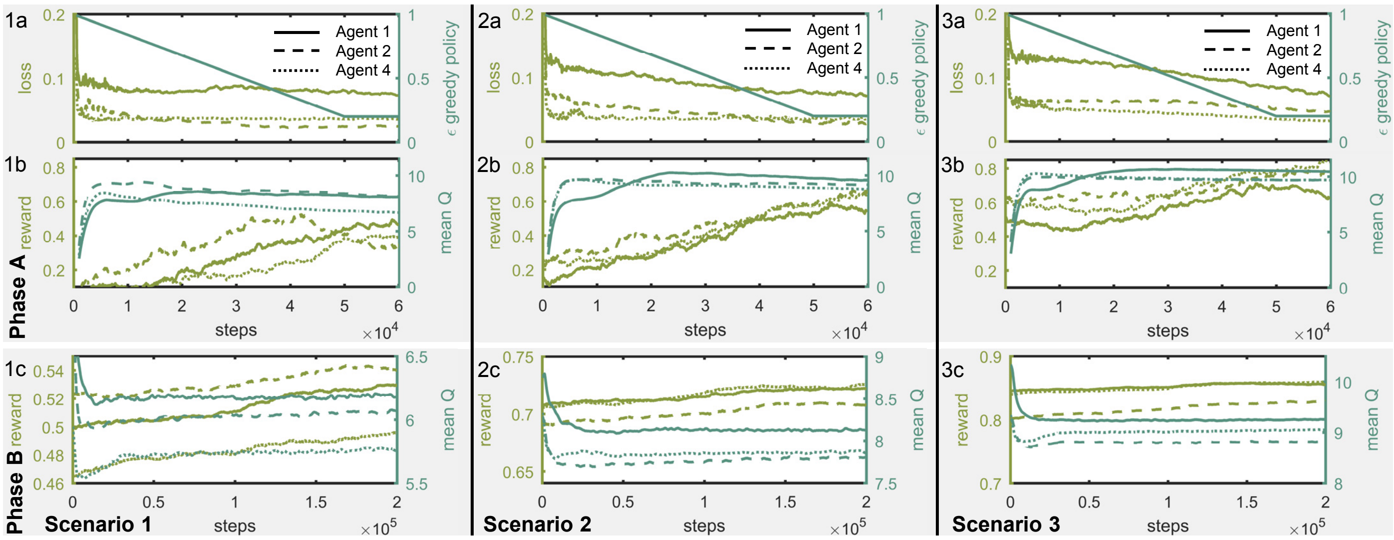


Fig. 5. Key parameters of the learning process for three factory scenarios in learning phases A and B. Agent 1, 2 and 4 correspond to the respective workcenters; workcenter 3 is not controlled by an agent as it acts mostly as a buffer. Steps are the time steps in the MDP. In (a), the loss functions of each agent as well as the ϵ parameter of the ϵ-greedy policy is shown for each of the scenarios 1,2 and 3 (share of random actions to blend the training set). The loss functions are converging quickly and set a good starting point for optimization of the whole line in phase B. The ϵ-greedy policy in phase A is identical for all agents. In phase B ϵ is set constant to 0.2. In (b) and (c) reward and mean Q value are presented for phase A and B respectively. In both the mean Q has converged to a fixed value. For a thorough definition of the machine learning parameters we refer to the first DQN agent algorithm publication [3] and the open source implementation in keras-rl [25].

to demonstrate the capability to develop superior dispatching strategies in more complex simulation models.

Acknowledgements

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.

References

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

[2] Mordvintsev, A., Olah, C., Tyka, M.. Inceptionism: Going deeper into neural networks. Google Research Blog Retrieved June 2015;20:14.

[3] Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A.A., Veness, J., Bellemare, M.G., et al. Human-level control through deep reinforcement learning. Nature 2015;518(7540):529–533.

[4] Silver, D., Schrittwieser, J., Simonyan, K., Antonoglou, I., Huang, A., Guez, A., et al. Mastering the game of Go without human knowledge. Nature 2017;550(7676):354–359.

[5] Ouelhadj, D., Petrovic, S.. A survey of dynamic scheduling in manufacturing systems. Journal of scheduling 2009;12(4):417–431.

[6] Waschneck, B., Altenmüller, T., Bauernhansl, T., Kyek, A.. Production scheduling in complex job shops from an industry 4.0 perspective. In: SAMI@ iKNOW. 2016,.

[7] Panait, L., Luke, S.. Cooperative multi-agent learning: The state of the art. Autonomous agents and multi-agent systems 2005;11(3):387–434.

[8] Zhang, W., Dietterich, T.G.. A reinforcement learning approach to job-shop scheduling. In: IJCAI; vol. 95. 1995, p. 1114–1120.

[9] Zhang, W., Dietterich, T.G.. High-performance job-shop scheduling with a time-delay td (λ) network. In: Advances in neural information processing systems. 1996, p. 1024–1030.

[10] Mahadevan, S., Marchalleck, N., Das, T.K., Gosavi, A.. Self-improving factory simulation using continuous-time average-reward reinforcement learning. In: Machine learning international workshop. Morgan Kaufmann Publishers; 1997, p. 202–210.

[11] Mahadevan, S., Theocharous, G.. Optimizing production manufacturing using reinforcement learning. In: FLAIRS Conference. 1998, p. 372–377.

[12] Bradtke, S.J., Duff, M.O.. Reinforcement learning methods for continuous-time Markov decision problems. In: Advances in neural information processing systems. 1995, p. 393–400.

[13] Riedmiller, S., Riedmiller, M.. A neural reinforcement learning approach to learn local dispatching policies in production scheduling. In: IJCAI; vol. 2. 1999, p. 764–771.

[14] Paternina-Arboleda, C.D., Das, T.K.. A multi-agent reinforcement learning approach to obtaining dynamic control policies for stochastic lot scheduling problem. Simulation Modelling Practice and Theory 2005;13(5):389–406.

[15] Brauer, W., Weiß, G.. Multi-machine scheduling – a multi-agent learning approach. In: Multi Agent Systems, 1998. Proceedings. International Conference on. IEEE; 1998, p. 42–48.

[16] Gabel, T., Riedmiller, M.. Scaling adaptive agent-based reactive job-shop scheduling to large-scale problems. In: Computational Intelligence in Scheduling, 2007. SCIS'07. IEEE; 2007, p. 259–266.

[17] Qu, S., Wang, J., Govil, S., Leckie, J.O.. 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 2016;57:55–60.

[18] LeCun, Y., Bengio, Y., Hinton, G.. Deep learning. Nature 2015;521(7553):436–444.

[19] Silver, D., Huang, A., Maddison, C.J., Guez, A., Sifre, L., Van Den Driessche, G., et al. Mastering the game of Go with deep neural networks and tree search. Nature 2016;529(7587):484–489.

[20] Mao, H., Alizadeh, M., Menache, I., Kandula, S.. Resource management with deep reinforcement learning. In: HotNets. 2016, p. 50–56.

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

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

[23] Uhlemann, T.H.J., Lehmann, C., Steinhilper, R.. The digital twin: Realizing the cyber-physical production system for industry 4.0. Procedia CIRP 2017;61:335 – 340. doi:https://doi.org/10.1016/j.procir.2016.11.152; the 24th CIRP Conference on Life Cycle Engineering.

[24] Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J., et al. Openai gym. arXiv:1606.01540; 2016.

[25] Plappert, M.. keras-rl. 2016. URL: `https://github.com/matthiasplappert/keras-rl`.

[26] Chollet, F., et al. Keras. 2015. URL: `https://github.com/fchollet/keras`.

[27] Kingma, D., Ba, J.. Adam: A method for stochastic optimization. arXiv preprint arXiv:14126980 2014;.