Available online at www.sciencedirect.com

ScienceDirect

Procedia CIRP 81 (2019) 234–239

www.elsevier.com/locate/procedia

52nd CIRP Conference on Manufacturing Systems

Design, Implementation and Evaluation of Reinforcement Learning for an Adaptive Order Dispatching in Job Shop Manufacturing Systems

Andreas Kuhnle[a,*], Louis Schäfer[a], Nicole Stricker[a], Gisela Lanza[a]

[a]*wbk Institute of Production Science, Karlsruhe Institute of Technology (KIT), Karlsruhe, Germany*

* Corresponding author. Tel.: +49-721-608-44153; fax: +49-721-608-45005. *E-mail address:* andreas.kuhnle@kit.edu

Abstract

Modern production systems tend to have smaller batch sizes, a larger product variety and more complex material flow systems. Since a human oftentimes can no longer act in a sufficient manner as a decision maker under these circumstances, the demand for efficient and adaptive control systems is rising. This paper introduces a methodical approach as well as guideline for the design, implementation and evaluation of Reinforcement Learning (RL) algorithms for an adaptive order dispatching. Thereby, it addresses production engineers willing to apply RL. Moreover, a real-world use case shows the successful application of the method and remarkable results supporting real-time decision-making. These findings comprehensively illustrate and extend the knowledge on RL.

Keywords: Reinforcement Learning; Production Scheduling; Order Dispatching; Methodical Approach

1. Introduction

Manufacturing companies are nowadays, in the age of connected *Cyber-Physical Systems* (CPS), more than ever confronted with new challenges but also opportunities leading to the fourth industrial revolution [1]. The digitization already covers companies that generate two-thirds of the world's total gross domestic product [2]. These companies focus on the integration of technological innovations in their role as solution providers to their customers. At the same time, they concentrate on innovations to maintain their operational competitiveness.

Especially in the domain of *Artificial Intelligence* (AI) and *Machine Learning* (ML) promising progress has been made in the last years. Firstly, the available hardware, e.g. graphics processing units, leads to acceptable computation time for data-intensive calculations. Secondly, advanced algorithm libraries are publicly accessible and open source, e.g. *tensorflow*, *keras-rl* and *tensorforce*. Lastly, successful applications, in particular in the area of *Reinforcement Learning* (RL), such as *AlphaGo* have attracted a large public audience [3].

However, considering the manufacturing industry there is an almost negligible number of attempts applying state of the art ML algorithms. A literature review in the subsequent section reveals just a few applications to common production planning and control (PPC) problems, e.g. scheduling. Additionally, a methodical approach for the usage of state-of-the-art RL-algorithms such as *TRPO*, *PPO* or *Deep Q-Learning* is missing.

Furthermore, traditional solution approaches such as mathematical optimization and heuristics fail to meet the demand for adaptability in the presence of uncertainties, smaller batch sizes, higher product diversity and at the same time the call for maximum productivity and efficiency [4]. Hence, new control mechanisms matter. Decentral and data-driven approaches such as RL are promising approaches, as they show a good performance for dynamic applications [5,6].

In this paper we introduce a methodical approach for the **design**, **implementation** and **evaluation** of RL applications. Section 2 describes the fundamentals and literature review of ML in production control, in general. Section 3 forms the main part of this paper wherein the methodical approach is outlined. The findings are summarized in Section 4 and 5.

10.1016/j.procir.2019.03.041

The developed method is exemplarily applied to a complex job shop manufacturing system taken from a semiconductor fabrication company. We closely build this work on the previous research of Stricker et al. [5]. The semiconductor industry appears as perfect example because the external market demand is highly uncertain and internal processes, e.g. the material flow in the job shop manufacturing system are complex systems [4]. In addition, the industry acts since many years as a pioneer in the field of digitized and connected production systems, e.g. the throughout implementation of RFID and indoor tracking systems [4].

2. ML applications in production control

2.1. Problem description and requirements

In the manufacturing industry, production control is the central element that allows a stable and cost-efficient production process, i.e. leading to a high operational efficiency [4]. Considering a complex job shop environment, the underlying main concern can be described as a resource allocation problem where the crucial aim is to optimize the usage of production resources with respect to previously defined *Key Performance Indicators* (KPI) such as the utilization of the resources or the lead time of the production orders. Thus, the task of production control can be described as a **decision-making problem** optimized with respect to the KPI.

The continuous adjustment, i.e. adaptability, according to dynamics of turbulent markets, the increasing customer requirements and unstable production processes are inevitable for any production control system. It is ever more important to quickly adapt production control decisions to changing environmental conditions [1]. For instance, the semiconductor industry is in a phase in which economic success is to a large extend determined by the operational excellence [4].

To overcome these obstacles, manufacturers consider decentralized, autonomous decision-making units to achieve a high flexibility and adaptability [5,6]. Mathematical optimization and heuristics cause a high manual adjustment effort in case of system changes due to their static nature and model-based implementation. Herein, RL-algorithms can, firstly, realize a **real-time-capable** and, secondly, **adaptive production control system**. RL-algorithms are not hardcoded, meaning in most cases model-free, and continuously adjust their procedure according to the data input [13].

2.2. Control applications of machine learning

A comprehensive overview of ML algorithm applications reveals that control application is one kind of recurring pattern. Independent of the system under control the algorithms that are applied are in the class of RL. Table 1 shows an extract of the review. It does not only reveal the suitability of RL for dynamic and complex production systems but also the transferability motivated in this paper. Due to its approach presented in the next section, RL perfectly integrates into the domain of adaptive production control.

Table 1. Overview and classification of ML applications.

Application	Context	Author	ML algorithm category
AlphaGo Zero	Go-Engine	[3]	RL
Locomotion Behaviors	Humanoid limbs	[7]	RL
Soccer Robot	Robotics	[8]	RL
3D Walking Biped	Robotics	[9]	RL
Improving Elevator Performance	Elevator control	[10]	RL
Playing Atari	Joystick	[11]	RL
Dispositive Order Control	Production control	[12]	RL
Complex Job Shop Scheduling	Production control	[6]	RL

2.3. Reinforcement learning

RL is one of the three categories of ML algorithms [13]. For a detailed definition and differentiation of RL from Supervised Learning and Unsupervised Learning we refer to [13]. RL follows the nature of learning just like animals or humans do through a continuous interaction of the decision maker (agent) with its environment. Fig. 1 illustrates on a high level, using the terminology of control theory, the application of RL as a controller unit for the system "production".

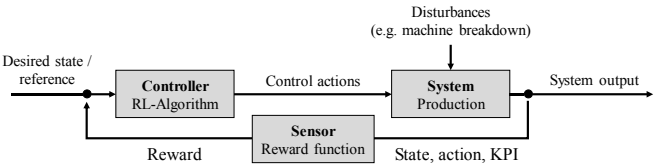

Fig. 1. RL for an adaptive production control.

The agent chooses an action $a_t \in A$ that influences its environment state $s_t \in S$ and is rewarded by a reward signal $r(S,A) \in \mathbb{R}$. It learns via the direct interaction with the environment, meaning from the context of its previous actions and the associated reward. The relationship, i.e. state-action-pairs, learned over time is called strategy $\pi: S \rightarrow A$. It is the agent's overall goal to maximize the long-term, cumulative reward by optimizing its strategy. The value function $V^{\pi}(s_t)$ is used as an estimator of how good it is for the agent to be in a particular state s_t of the environment and $Q^{\pi}(s_t, a_t)$ to choose a corresponding action a_t in a given state s_t. [14]

When modelling a RL-agent, the design of the **action space**, **state representation** and **reward function** as well as further RL-algorithm specific **hyperparameters** have to be considered in detail for a successful RL-implementation.

2.4. Related work

There are some successful RL-applications (see Table 1) such as AlphaGo Zero [3], the outperforming achievements of RL playing Atari games [11] as well as early ones like the optimization of elevator performance in 1995 [10].

The application areas of RL are potentially diverse but there

are very few serious applications in the manufacturing context. Most well-known use cases concentrate on an illustrative demonstration of RL using simplified problems with for example just a small action-space. Early applications of RL in production control date back to 1996 when Zhang and Diettrich used a neural network trained with RL to schedule a static job shop [15]. Scholz-Reiter et al. [16] apply RL for the inventory control in production systems. Recent research by Waschneck et al. [6] addresses the optimization of a global production scheduling with Deep-Q-Learning, Gabel et al. [17] focus on a multi-agent approach for the distributed job shop scheduling and Shiue et al. [18] concerning the optimal scheduling rule selection based on Q-Learning.

Although RL has been applied successfully, it often lacks sufficient information regarding the modeling of the solution algorithm. Therefore, this paper is split up into the three phases of design, implementation and evaluation of RL and exemplarily illustrates the application by the example of adaptive order dispatching.

3. Methodical approach for RL application

This section presents the methodical approach for the design, implementation and evaluation of RL-algorithms for an adaptive order control. Generally, one has to note the iterative nature of the phases, meaning it is for instance mostly necessary to come back to the design phase when looking at the evaluation results. The method addresses production engineers not familiar with RL and compromises key questions that need to be answered by practitioners when applying it.

3.1. Design

The design phase aims to translate the real-world problem into a RL problem formulation. Not all types of optimization tasks are suitable for RL. Dynamic optimization problems in the domain of production control with the goal to achieve on average a high performance are in line with RL. However, optimization problems that are, for instance, looking for the (static) best weekly production schedule or a strategic planning decision miss the interactive nature of RL.

Furthermore, a common design characteristic of RL is to approximate the optimal strategy by the application of an *Artificial Neural Network* (ANN). Alternatively, one could use a lookup table. However, this is not computationally efficient in case of a large number of and / or continuous variables.

Fig. 2 shows an ANN. It summarizes the key elements of RL, their connection to elements of ANNs as well as their relationship and interplay within the learning procedure. The following subsections elaborate on the design of these four key elements.

Action space representation

Purpose: The action selection in RL is determined by the strategy which is technically based on probabilities, i.e. weights in the ANN. So, it is representing a classification problem (see Fig. 2). Moreover, the reward is always related to an action.

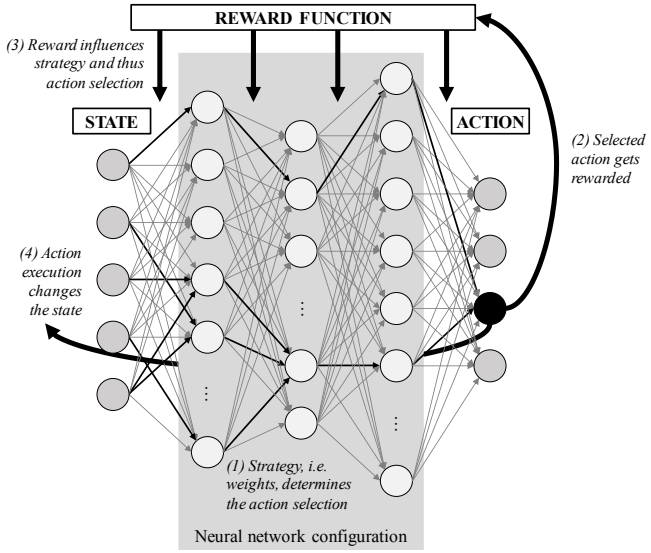

Fig. 2. ANN application, RL set-up, key elements and learning procedure.

Main considerations: Designing the agent's action space is rather straightforward, because it directly emerges from the problem description. The decision agent has to fulfill a specific task, e.g. order dispatching, and hence, a fixed set of actions is required that influences the environment resulting in the dispatching of orders. To give another example, just two actions "move right" and "move left" are required for most simple Atari Games [11]. However, in most real-world problems the action space is larger. For large action spaces one has to consider, that the actions are biunique and not conditional to each other [14]. Moreover, the action space has to be designed in such a way that in any environment state there is always at least one action that yields a higher reward than any other. Otherwise the agent is not able to learn the relation between a given state and the "optimal" action.

Methodical key steps:

- Is the agent able to change the environment (state)?
- Is the number of actions as high as necessary and as small as possible, i.e. no conditional actions?
- Are the actions modeled in a way that the agent can always choose an action that is better than any other?
- Are there any infeasible actions that are not executable in a specific environment states and therefore have to be taken into account when designing the reward function?

State representation

Purpose: The agent's state representation builds the central basis for the action selection and represents the agent's observation of the environment state (see Fig. 2). The state design has a big influence on the achievable learning performance [14]. Therefore, it is crucial to understand the way the agent observes the environment. Whereas, games such as Atari are easy to model with all pixels on the screen [11], most real-world control problems are not so easy to comprehend and multiple dimensions and perspectives such as machine state, inventory level, order due dates and resource utilization are relevant.

Main considerations: The state representation is used in the RL implementation as input vector for the ANN and it is just allowed to contain numerical values. This might necessitate

data preprocessing, such as transformation, normalization or feature extraction, which is well-known for data mining and supervised learning applications. One possibility to numerically encode categorical data is a (binary) one-hot vector, where the input simply either contains a one or a zero for the specific category [6]. Moreover, it is in some cases beneficial to code state information gradually, meaning multiple binary inputs. In comparison to the action space it is not that relevant to keep the state representation rather small. Advanced RL-algorithms are able to handle large input vectors such as demonstrated by Silver et al. [3]. They even propose a duplication of important state information in order to increase the "attention" and relevance for the RL-algorithm [3].

Additionally, the state vector has to display the optimal target state, if such state exists. For example, in chess the victory at the end of a game can be considered as optimal (terminal) state and rewarded as such. However, production control applications focus on maintaining a state of high performance, representing a kind of endless game. This issue is discussed later.

Finally, the state representation needs to make the change resulting from the action execution "sufficiently" transparent. This means that for instance the machine utilization representation via absolute percentage value is less sensible to changes than a representation by the relative increase or decrease. Furthermore, the latter representation is centered on zero and independent of the application.

Methodical key steps:

- Is the environment numerically encodeable?
- Does the consideration of one-hot vectors for categorical or gradual data applies?
- Which important input might be duplicated to point it out?
- Does an optimal terminal state exist and is it representable with the state representation?
- Does the state representation make changes in the environment due to the action execution transparent?

Reward function

Purpose: The reward function brings both, the action space and the state representation, together and directs the strategy optimization.

Main considerations: The reward function can be modeled linearly, exponentially or by another mathematical function. In most applications a piece-wise defined function is given. If there are multiple KPI, it is very important to consider the dependencies. Multiple objectives are usually integrated by a linear combination.

Additionally, the reward function should consider an increasing gradient towards the target state. This is beneficial for most gradient-based RL optimization algorithms. So, exponential functions are preferable to linear functions or step functions that just reward the target state.

In case there is no natural terminal state, e.g. the end of a game, terminal states can be simulated e.g. with a fixed time limit. Another modelling guideline is that positive reward is accumulated and tries to avoid terminal states and a negative reward is followed or close to a terminal state.

Furthermore, it is not always advisable to reward interim target states because the agent might develop a strategy that follows these interim states by maximizing the reward and hence ignoring the actual target state [14]. Therefore, it is crucial when designing the reward function that it achieves the overall goal, e.g. minimal lead times as a result of the maximization of the cumulative reward. This is also recommended if multiple target states exist. It might also require splitting KPI up into its components, because an aggregated figure contains components that cannot be influenced by the agent. The order lead time, for example, contains the process, waiting and transport time, whereas the process time is given and cannot be changed by the dispatching decision.

Methodical key steps:

- Is there a single optimal state that is targeted or does the optimization focus on a continuous control operation?
- How do the KPI relate to the level of reward (increasing gradients are preferable)?
- Are there any dependencies between multiple KPI?
- Does a positive reward avoid terminal states and a negative reward lead to a terminal state?
- Is the maximal reward in line with the target state?
- Is just the RL-agent affecting the KPI by its actions?

RL-Algorithm hyperparameter

Purpose: Depending on the RL-algorithm, there are usually multiple hyperparameters that have to be modeled.

Main considerations: Most RL-algorithms are based on a learning rate α, discount factor γ, exploration rate ϵ, batching capacity and the ANN configuration [14]. The learning rate α determines the step size of the ANN-weight update. The discount factor γ determines the degree of farsightedness of the agent, i.e. to how many time steps is the reward assigned to. The exploration rate ϵ determines how the agent behaves regarding exploring new actions versus following (exploiting) the so far best-known action. The batching capacity is a parameter that affects the speed and stability of the learning process. For more detailed information we refer to [14].

Defining these parameters often appears to be similar for different applications and in the end also "trial and error" when fine-tuning the performance. Universal guidelines on the influence of all mentioned hyperparameters are not available. Generally speaking, the convergence during the learning phase only occurs for sufficiently low learning rate values. Considering the ANN configuration, a small network often results in loosing information and therefore non-convergence and a large one increases the computation effort and the chance of overfitting.

Methodical key steps:

- Which hyperparameters are relevant for the RL-algorithm?
- Are default values for any hyperparameters available?
- How many input and output neurons are given, which can be considered for the hidden layer design?

3.2. Implementation

Purpose: After designing the RL-algorithm the implementation phase focuses on the realization as an operational system.

Main considerations: The application and evaluation of new

control methods in real-world systems is very time- and cost-intensive. Just as the broad range of Atari games [11] the implementation of RL-algorithms is therefore recommended in a simulation environment. For CPS the "digital twin" can be used as a perfect training environment. When the agent's performance is good enough, it can be applied to the real-world.

Simulation can be either event- or time-discrete [19]. With regard to the use case of production control we suggest an **event-discrete model** with discrete decision points. For the simulation one first has to ensure that the simulation is valid and verified [19]. The simulated environment must also only contain information that is also available and accessible in the real-world. Otherwise the transfer is not possible.

Finally, for the implementation of the agent-environment interaction, two alternatives prevail: First, the (simulation) environment is leading and the agent is asked whenever an action is requested. Second, the agent is in control and "plays" with the environment. We propose the first alternative as it is more suitable for the most applications in production control.

Methodical key steps:

- Can the environment be adequately simulated in order to facilitate the training and testing?
- Which software tools provide a computational efficient implementation for the data-intensive computations?

3.3. Evaluation

Purpose: One important step before applying a RL model to a real-world problem is the evaluation of the agent system. This section describes a procedure of how to analyze and test the adaptive control system before finally transferring it to the real-world. Multiple experiments and sensitivity analysis can be easily performed within the "digital twin" simulation model.

Main considerations: First, one has to look at the **realized reward** during the learning phase. This gives information on the general learning ability of the RL-agent, i.e. whether the agent is able to improve its strategy during the learning phase. It is important to observe whether the reward increases and convergences. Moreover, the time / number of learning iterations it takes until a high reward level is reached is also noteworthy. The maximum reward level usually is an arbitrary number that does not directly translate into meaningful real-world number. Lastly, next to a high reward a stable reward signal is desirable. In some applications the reward signal collapses after a high level has been reached before. This is due to the dynamic nature of the RL-algorithm and thus needs to be evaluated with respect to the application.

Although a high and converging reward is essential, it does not necessarily guarantee a "good" control strategy in the first place. This is mostly caused by a wrongly modeled reward function that does not match the overall objectives (KPI) that are relevant to the control task. Another reason could be external disturbances, e.g. breakdown of machines that hinder the agent to perform any better than that. Therefore, it is crucial to **review the agent's actual behavior**. For example, if it is possible to select an action which is not executable in some environment states it can be revealing to look at the development of the number of infeasible actions the agent chooses. Moreover, analyzing the overall KPI, e.g. utilization or lead time, is crucial and gives a more accurate view on the quality of the agent's strategy. The KPI also offer a good possibility to benchmark the adaptive RL-algorithm with alternative solution approaches such as heuristics. Moreover, existing, historic KPI can be applied as benchmark, too.

In addition to that, it is advisable to validate the agent's strategy applicability within a slightly modified problem.

Methodical key steps:

- Does the reward increase and converge on a high level?
- Do the KPI show the same performance?
- Is the agent behavior valid – maybe consider a simplified, trivial problem for validation first?
- How does the RL-algorithm perform comparing to a benchmark solution approach?

4. Application and experiments

After proposing the methodical approach this section exemplarily applies it to an adaptive order dispatching, which advances the work of Stricker et al. [5].

Design: Action space representation

The action space follows the order-oriented dispatching problem. All orders need to be dispatched to a machine or to a buffer. So, all machines and buffers are possible discrete actions. Additionally, an idle action is added. Hence, there is always at least one action that is better than any other.

Design: State representation

The environment state offers more alternative design approaches because there are, for example, always infeasible actions that depending on the current state cannot be executed, e.g. transporting order from A to B when there is no order at A. So, it is reasonable to include the binary information. This data is encoded as one-hot vector. Additionally, the information about the buffer level in front of a machine is represented by the sum of the processing time of all waiting orders as well as the current maximum waiting time of an order at every machine in order to emphasize it. Evaluation reveals better results for this representation in contrast to e.g. adding the current buffer level into the state representation. Finally, the location of the dispatching unit is included.

Design: Reward function

A reward function r based on the KPI utilization $util$ and lead time LT_i is developed and defined as follows:

$$r(s,a) = \begin{cases} 0 & \text{infeasible or empty move} \\ 0.2 & \text{order to machine} \\ 0.5 \cdot r_{LT} + 0.5 \cdot r_{Util} & \text{finished order to exit} \end{cases} \quad (1)$$

$$r_{LT}(LT_i) = \begin{cases} 1 & LT_i < 20 \\ -0.008 \cdot LT_i + 1.16 & LT_i \in [20,120] \\ 0 & \text{others} \end{cases} \quad (2)$$

$$r_{util}(t_0,t) = \begin{cases} -1 + 2 \cdot util(t_0,t) & util(t_0,t) > 0.5 \\ 0 & \text{others} \end{cases} \quad (3)$$

The reward is given in order to reach a continuous high performance level, without just one single optimal state. Note that for the lead time just the waiting time is considered, and the utilization leaves out disturbances. So, just what the agent affects defines the learning feedback.

Design: RL-Algorithm hyperparameter

The herein presented results are based on the following hyperparameter set: *TRPO* policy-based RL-algorithm, *adam* stochastic gradient optimizer, $\gamma = 0.2$, $\alpha = 0.001$, a dense two layered net with 64 neurons each and *tanh* activation function.

Implementation

The simulation-based adaptive production control and the RL-algorithm are implemented using the Python libraries SimPy (simulation) and TensorForce (RL). Both together ensure an integrated and computational efficient runtime environment. The validation is based on a detailed evaluation of the simulation events as well as tests for extreme values.

Evaluation

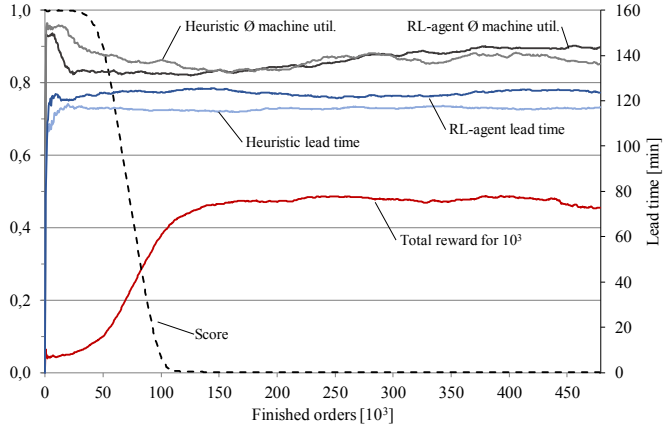

Fig. 3. Computational results.

Fig. 3 illustrates the computational results. Firstly, the reward (averaged values over 10^3 finished orders) shows, that the RL-algorithm achieves a good learning performance. Secondly, the KPI machine utilization and lead time validate and confirm the optimized performance not just with respect to the agent's reward. Moreover, the adaptive RL-algorithm is able to optimize both KPI, which are at least partially contradicting objectives, as the utilization is focusing on resources and the lead time on orders. Thirdly, for the ratio of infeasible to feasible actions we observe a *score* representing the number of different actions the agent proposes when asked multiple times per interaction, i.e. an indicator for the action selection uniformity. Comparing this to the reward also reveals that both improvements happen in parallel. Lastly, the comparison to a heuristic dispatching rule demonstrates the overall quality of the adaptive control. So, the computational results show that following the method described in Section 3 can successfully applied to the order dispatching.

5. Summary

This research focuses on learning algorithms for adaptive control systems that are applied to complex manufacturing systems in order to increase the operational efficiency. Herein, a method is presented that addresses production engineers not familiar with RL and focusses on the design, implementation and evaluation of RL-algorithms in the domain of order dispatching. The promising results motivate this field of research also in future. Especially, data-intensive CPS and digital twins are prone for future research in PPC. This work facilitates the application of RL to any production control case.

Acknowledgements

We extend our sincere thanks to the German Federal Ministry of Education and Research (BMBF) for supporting this research project 02P14B161 "Intro4.0".

References

[1] Bauernhansl T, ten Hompel M, Vogel-Heuser B. Industrie 4.0 in Produktion, Automatisierung und Logistik. Wiesbaden: Springer; 2014.
[2] Schaeffer E. Industry X.0. Digitale Chancen in der Industrie nutzen. München: Redline Verlag; 2017.
[3] Silver D, Schrittwieser J, Simonyan K, Antonoglou I, Huang A, Guez A, Hubert T, Baker L, Lai M, Bolton A, Chen Y, Lillicrap T, Hui F, Sifre L, van den Driessche G, Graepel T, Hassabis D. Mastering the game of Go without human knowledge. Nature 2017;550(7676):354-359.
[4] Waschneck B, Altenmüller T, Bauernhansl T, Kyek A. Production Scheduling in Complex Job Shops from an Industry 4.0 Perspective. CEUR Workshop Proceedings 2016;1793:12-24.
[5] Stricker N, Kuhnle A, Sturm R, Friess S. Reinforcement learning for adaptive order dispatching in the semiconductor industry. CIRP Annals 2018;67:511-514.
[6] Waschneck B, Reichstaller A, Belzner L, Altenmüller T, Bauernhansl T, Knapp A, Kyek A. Optimization of global production scheduling with deep reinforcement learning. Procedia CIRP 2018;72:1264-1269.
[7] Heess N, TB D, Sriram S, Lemmon J, Merel J, Wayne G, Tassa Y, Erez T, Wang Z, Eslami SMA, Riedmiller M, Silver D. Emergence of Locomotion Behaviours in Rich Environments. arXiv 2017:1707.02286.
[8] Asada M, Noda S, Tawaratsumida S, Hosoda K. Purposive Behavior Acquisition for a Real Robot by Vision-Based Reinforcement Learning. Machine Learning 1996;23:279–303.
[9] Tedrake R, Zhang TW, Seung HS. Stochastic policy gradient reinforcement learning on a simple 3D biped. IEEE/RSJ International Conference on Intelligent Robots and Systems 2004;3:2849–2854.
[10] Crites RH, Barto AG. Improving elevator performance using reinforcement learning. Advances in neural information processing systems 1996:1017–1023.
[11] Mnih V, Kavukcuoglu K, Silver D, Graves A, Antonoglou I, Wierstra D, Riedmiller M. Playing Atari with Deep Reinforcement Learning. arXiv 2013:1312.5602.
[12] Stegherr F. Reinforcement-Learning zur dispositiven Auftragssteuerung in der Variantenreihenproduktion. München: Utz Wiss; 2000.
[13] Russell SJ, Norvig P. Artificial Intelligence. A modern approach. Boston: Pearson; 2016.
[14] Sutton RS, Barto AG. Reinforcement Learning. An introduction. Cambridge, Mass: MIT Press; 2010.
[15] Zhang W, Diettrich TG. High-performance job-shop scheduling with time-delay td(λ) network. Advances in neural information processing systems 1996:1024-1030.
[16] Scholz-Reiter B, Hamann T. The behaviour of learning production control. CIRP Annals 2008;57:459-462.
[17] Gabel T, Riedmiller M. Distributed policy search reinforcement learning for job-shop scheduling tasks. International Journal of production research 2012;50(1):41-61.
[18] Shiue, YR, Lee KC, Su CT. Real-time scheduling for a smart factory using a reinforcement learning approach. Computers & Industrial Engineering 2018;125:604-614.
[19] Law AM, Kelton WD. Simulation modeling and analysis. New York: McGraw-Hill; 1991.