Available online at www.sciencedirect.com

ScienceDirect

Procedia CIRP 79 (2019) 391–396

www.elsevier.com/locate/procedia

12th CIRP Conference on Intelligent Computation in Manufacturing Engineering, 18-20 July 2018, Gulf of Naples, Italy

Autonomous order dispatching in the semiconductor industry using reinforcement learning

Andreas Kuhnle[a,]*, Nicole Röhrig[a], Gisela Lanza[a]

[a] *wbk Institute of Production Science, Karlsruhe Institute of Technology (KIT), Kaiserstr. 12, 76137 Karlsruhe, Germany*

* Corresponding author. Tel.: +49-721-608-46166; fax: +49-721-608-45005. *E-mail address:* andreas.kuhnle@kit.edu

Abstract

Cyber Physical Production Systems (CPPS) provide a huge amount of data. Simultaneously, operational decisions are getting ever more complex due to smaller batch sizes, a larger product variety and complex processes in production systems. Production engineers struggle to utilize the recorded data to optimize production processes effectively because of a rising level of complexity. This paper shows the successful implementation of an autonomous order dispatching system that is based on a Reinforcement Learning (RL) algorithm. The real-world use case in the semiconductor industry is a highly suitable example of a cyber physical and digitized production system.

Keywords: Production planning; Reinforcement learning; Semiconductor industry

1. Introduction

The productivity of manufacturing systems and thus their economic efficiency depends on the performance of production control mechanisms. Because of an increasing global competition and high customer demands, the optimal use of existing resources is ever more important. Optimizing production control is therefore a central issue in the manufacturing industry.

Depending on the industry, companies are additionally facing complex manufacturing processes due to high product diversity, lot size reduction and high quality requirements. In the example of semiconductor industry complexity arises through a high number of manufacturing processes and their precision on a nanometer level [1]. Planning and coordinating processes is hence a challenging task and requires a suitable control method.

Moreover production control deals with a dynamic and non-deterministic surrounding and thus has to handle uncertainty and unexpected incidents [2]. Established approaches for production control, such as mathematical programming, heuristics and dispatching rules are not able to meet these needs [3].

Through the integration of manufacturing components, enhanced process monitoring and data collection, Cyber Physical Production Systems (CPPS) provide real time data such as order tracking data, machine breaks and inventory levels. This makes it possible to apply data-driven techniques, such as Machine Learning (ML), that are able to adjust to the current system state by analyzing the available data in real time. The potential of ML in production control to create decentralized and autonomous systems has been evaluated by several researchers such as [3,4,5]. This paper describes and evaluates the implementation of an autonomous agent-based control system for order dispatching in a real-world use case in the semiconductor industry.

1.1. The order dispatching problem

Order dispatching is a task of operational production control. It describes the allocation problem of processing n single-stage orders $O = \{O_1, O_2 \dots, O_n\}$ with time duration $T = \{t_1, t_2, \dots, t_n\}$ on a set of m machines $M = \{M_1, M_2, \dots, M_m\}$. Each machine can handle one order at a time. The objective is to allocate the orders in a way as to optimize a set of k performance measures $F = \{F_1, F_2, \dots, F_k\}$ such as workload, throughput time or tardiness [2,6,7]. Depending on the production environment a valid solution has to consider additional constraints such as machine-allocation constraints ($O \rightarrow M$) or buffer capacities ($M_i \rightarrow \mathbb{R}^+$).

The static order dispatching problem described above is NP-hard [6]. In real-world use cases non-deterministic events, such as machine break downs and dynamic processes, such as new order arrivals, present an additional challenge for production control [3]. Furthermore, most problems are multi-criterial. Whilst predefined decision rules or mathematical optimization methods can hardly perform effectively under such circumstances [5] agent-based, learning control mechanisms can react flexible to changes in their environment. An agent-based and decentralized architecture also has benefits with regard to the complexity, as it allows splitting the problem into sub-problems that can be solved by independent agents [7].

The structure of the paper is as follows: Section 2 describes the theoretical concepts of agent-based production control and Reinforcement Learning (RL). Those are used in Section 3 to implement the components of an agent-based production control. The control system is tested and evaluated in Section 4. As a benchmark, the simulation results of the agent-based solution is compared to a heuristic rule-based algorithm in a real-world scenario.

2. Reinforcement Learning agents

One main idea of agent-based control is to split up the control task into several subtasks that can be solved on a local level. The resulting decentralized structure is flexible [7] and furthermore allows a reduction of computation time through the parallelization of decisions. Operational production control for example requires an order release decision, order dispatching decision and a single-machine dispatching decision. The responsibility for the subtask can be assigned to independent agents, then acting on a local level in the production environment. The role of an agent as a decision-making entity is defined in the following.

2.1. Agent definition

Agents are an essential concept of intelligent computing and distributed system design [5]. On a functional level, an agent is a computational system that [5,8]:

- Interacts with a dynamic environment
- Is able to perform autonomous actions
- Acts with regard to a specific objective

To achieve this behavior [9] proposes an agent architecture that has three key components: For the interaction with its environment the agent needs sensors to perceive aspects of its surrounding and actuators to execute its actions. To generate autonomous and objective-driven actions, a third component, the so-called "agent function" is required [8].

In this model, the agent function is the key component for defining the agent's behavior. It determines how the perceived information is processed to decide on actions that lead to a "good" performance with regard to the objective. At the same time, it compromises the agent's experiences captured in its surrounding. This is crucial to learn the consequences of the agent's decisions. Eventually, the "agent function" represents a learned model of the environment.

The (production control) system can consist of several agents with overlapping environments. In that case it is called a multi-agent system [3].

This agent definition describes a general approach to design systems that solve a particular problem. However, to specify the interaction process and the components additional techniques are required. This paper proposes the application of an RL algorithm for that purpose.

2.2. Reinforcement Learning

RL applies the ideas of a learning agent-based approach to optimization problems. Because the learning capability is based on repeated interaction with the environment it is often referred to as "trial and error" learning [10].

Despite the existence of many different RL algorithms that vary in the concrete realization of the learning functionality, they follow the same steps in the agent-environment interaction, see Fig. 1.

The agent perceives the actual state of the environment as a vector S_t. In order to decide on an action A_t the information is processed in the agent function that stores the current policy

$$\pi_t(a|s) = \mathbb{P}(A_t = a \,|S_t = s) \tag{1}$$

After the action is performed in the environment the agent perceives the new state S_{t+1} and a reward signal R_{t+1}. Note that the environmental transformation is associated to a Markov Decision Process (MDP). According to the received feedback, the agent adapts its policy. [10]

These steps are repeated in an iterative procedure. As a result, the agent optimizes its behavior in a way to find a policy π maximizing the long-term reward – therefore a policy that corresponds best to the agent's objectives. [10]

Finding an optimal policy is a dynamic process. In each iteration, the current policy π_t is adapted depending on the latest experiences. There are two main techniques to determine the new policy: (i) value-based approaches and (ii) policy-based approaches. A summary of the update functions and the instructions for both cases can be found in Table 1, where α is the learning rate and η an evaluation rate for the current policy representation.

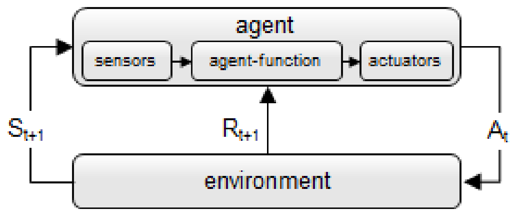


Fig. 1. Agent-environment interaction, derived from [9,10].

Table 1. Policy and value approximation, derived from [10].

	Policy approximation	Value approximation
Update function	policy function $\pi_t = \mathbb{P}(A_t = a \vert S_t = s)$	action-value function $Q_\pi = \mathbb{E}[\,\mathrm{R_t} \vert S_t = s, A_t = a]$
Update rule	$\overrightarrow{\theta_{t+1}} \leftarrow \overrightarrow{\theta_t} + \alpha\, \nabla\widehat{\eta(\theta_t)}$	$\overrightarrow{\theta_{t+1}} \leftarrow \overrightarrow{\theta_t} + \alpha\ [Q_\pi - Q_t]\, \nabla_{\overrightarrow{\theta_t}}\, Q_t$

The main difference between both approaches is that value approximation learns the action-value function during the interaction instead of directly learning a policy π. The value function $Q_\pi = Q_\pi(s, a)$ defines the expected long-term return when choosing an action a in state s following policy π. The policy is then derived from the estimated value of all possible actions in each state. Policy approximation, on the other hand, directly updates the policy function $\pi_t = \pi_t(a|s)$.

Most real-world problems deal with continuous action and state spaces. Storing and updating the policy or value function in a table is therefore computationally inefficient and requires lots of memory space. Therefore, the original policy or value function has to be stored approximatively. Artificial neural networks are widely used for that purpose, as they are capable of approximating complex functional relationships via the weights within the network and allow to adapt those weights dynamically during the learning process [10]. As a result, neural networks reduce the computational effort by updating a set of weight parameters $\overrightarrow{\theta_t}$ instead of the values for each state-action pair in each iteration, see Table 1.

Depending on the dimension and the characteristics of the problem, different learning approaches might lead to good results. In recent years new kinds of RL algorithms such as PPO [11], TRPO [12] and DQN [13] were developed to deal with complex problems in different domains. They can be regarded as advanced policy or value approximation algorithms that are optimized with regard to an efficient and stable learning process together with neural networks. The solution to the order dispatching problem, presented in this paper, builds on these RL algorithms

2.3. Applications of Reinforcement Learning agents in the manufacturing industry

Because of the iterative learning process (see section 2.2), RL algorithms are suitable methods in dynamic environments in which complete information cannot be provided entirely from the beginning. The generality of the concepts allows the application of RL for decision problems in various industrial applications. Next to the purpose of production control, other successful implementations of RL agents exist for example in board games (e.g. chess), robotics or process engineering.

Games, such as strategy games or Atari games, are used by many researchers to design and test new algorithms. Results presented in [13] and [12] show that modern RL algorithms already exceed human performance in specific domains. The design of autonomous robotic systems is another field of application to learn coordinating movements to fulfill a certain task [14,15].

Other industrial applications are found in process engineering. In [16] an actor-critic RL algorithm is implemented to control the process parameter laser power in a welding process. The experimental results for a particular setup show that RL generates stable solutions and is suitable for a real-time and dynamic control mechanism.

In the context of production control, other authors investigated the usage of RL methods for order scheduling. The scheduling approaches differ in their overall architecture. The system proposed in [2] and [3], for example, focuses on a highly distributed form of RL, where each resource and each order are considered as agents. In this kind of architecture resources bid for the allocation of an order depending on the estimated processing cost when being selected. To reduce computational complexity a RL solution is presented to estimate the benefit of allocating a job to a specific resource. The implemented RL algorithm uses a table-representation in a single-objective problem.

The work of [7] and [17] apply Q-learning to a single-machine scheduling problem and a layout with several process steps. The order scheduling within each machine and the order release in [17] are performed by RL agents. In both cases, a table is used to store the value function.

The solution developed in this paper is an autonomous, single-agent dispatching system. In contrast to previous RL applications for order scheduling, the work presents an implementation that is based on function approximation with artificial neural networks. This approach is appropriate for continuous action and state spaces and hence a suitable approach for real-world situations.

3. Agent-based approach for the autonomous dispatching problem

The following section specifies the architecture of the agent as well as the overall RL algorithm setup that was developed to solve the dispatching problem. The main features of an agent are a dynamic environment and autonomous, objective-driven actions (see section 2.1). In the following, the components of the agent design are therefore described with regard to those features. The whole system is evaluated in exemplary production layout in the end of this section.

3.1. Dynamic production environment

The environment of the dispatching problem is the production system. The layout is defined by the position of the resources (e.g. machines) of the system. The dynamic process of order processing is modelled as follows:

Released orders wait in entry buffers until they are affected by a decision of the dispatching agent. The agent allocates orders to a specific machine whereupon a transportation medium moves the orders to the machine's buffer. When a machine has processed an order the order waiting longest in the buffer is processed next. Note that instead of applying a simple decision rule it is possible to implement agents for the single-machine dispatching problem. However, for reasons of simplicity the proposed implementation uses a rule-based approach in this step. After being processed the order is stored in the buffer again until the dispatching agent decides to move it to one of the final exit stock for finished goods.

Machines might differ in their tools and their capabilities. Similar machines are assigned to the same machine group. Orders arrive dynamically and contain information about their work content and their processing times. Work content and processing times are stochastically distributed. According to the work content, all machines of one specific machine group can process the order. Therefore, the dispatching agent has to decide on the allocation within that group. Stochastic events in the environment occur due to machine failure. They lead to a limited availability of the machines.

The environment is implemented in Python using the discrete-event simulation provided by the library simpy. Hence, simulation data instead of real-world data is used for the training in order to have a proper "trial and error" interaction with the environment of the agent. When a dispatching decision is required, the simulation transfers the environments state to the learning agent and performs the action selected and returned by the agent. A decision results in a movement of the transportation resource with or without an order. Thus, the dispatching agent can be associated with the transportation resource.

3.2. Objective-driven action determination

Objective-driven actions require a feedback from the environment to the agent. This feedback has to be a numeric signal that is transferred to the agent after each decision. Therefore a reward function is composed with the values of k performance measures $F = \{F_1, F_2, ..., F_k\}$ for its multiple objectives. Each performance measure F_i is represented in a reward $r_i = f(F_i) \in [0,1]$. The total reward r is then calculated as the weighted sum of the single rewards. The single reward functions have to be defined depending on the values of the performance measures in the particular problem.

A reward of zero is given when the agent decides on an action that cannot be executed by the transportation resource, for example due to machine failure or a buffer overflow. The low value indicates that the agent should avoid such kind of actions, whereas a high value makes the agent behave similarly in the future.

3.3. Autonomous decision-making

The capability of performing autonomous actions requires learning from the experience from preceding interactions with the environment. For a RL agent this means defining the way the agent adapts its strategy. For the implementation of this characteristic, the paper utilizes the TRPO algorithm [12] provided by the RL Python library tensorforce. These algorithms have predefined interfaces for the information transmission between learning agents and their environment. In order to match the specific problem they have to be customized with a set of learning parameters.

However, independent from a specific algorithm, the input for the algorithm is a state representation and a reward signal, whereas the output is an action. Therefore, an action and a state representation have to be defined.

Each action A_i results in a movement of the transportation resource between the sources, exit stocks and machines. They are represented as a positive integer $A_i \in \mathbb{N}^+$. For the dispatching problem there are three types of actions:

- A_{Aloc}: Allocating an unprocessed order to a machine, which results in the transportation to that machine
- $A_{Transport}$: Transporting a processed order from a machine to one of the exit stocks
- A_{Empty}: Changing the location of the transportation resource without affecting any order

The state representation S_t needs to contain all decision-relevant information from the production environment. It includes information about:

- Location of the transportation resource.
- For each machine M_i: One binary value composed of the machine's current availability and the buffer filling state to indicate whether an action A_{Aloc} ending at machine M_i is possible or not. A second binary value based on the existence of a processed order in the buffer indicating whether an action $A_{Transport}$ starting machine M_i is possible or not. Two floats value for the sum of processing times of unprocessed orders and waiting times of processed orders.
- For each entered order: One float value for the waiting time of the longest waiting order. A second float indicates on which machines the longest waiting order can be processed.

3.4. Proof of concept

In order to prove that the RL algorithm architecture is suitable to solve the described dispatching problems, the functionality is tested in a simple exemplary production layout. The chosen scenario is a system with three similar machines with an availability of 100% and equal order processing times. All production parameters are determined in such a way that it allows a constant maximum workload of 100% when performing the correct actions in the correct order. A human can determine the trivial correct order. In a simulation run the concept can be verified where the action-sequence found by the dispatching agent matches the known optimal solution. With a proper agent configuration, i.e. the algorithm is not stuck in one of the local optima, the (global) optimal solution is computed after 5000 to 10000 simulation iterations.

4. Real-world use case from the semiconductor industry

The architecture presented in the previous section is applied to a real-world use case from semiconductor industry. The Python implementation of the environment is therefore instantiated with the layout data from the use case. Additionally, the action representation, the state representation and the reward function are customized to match the specific problem. This section gives a detailed description of the use case as well as the experimental results. The agent's performance is evaluated in comparison with a heuristic dispatching rule serving as a benchmark. Furthermore, the impact of varying specific model parameters is investigated.

4.1. Use case description

In order to simulate the production system of the use case scenario the implementation of the dynamic environment as well as the reward function and the action and state representation (see section 3.3) are adapted.

In the use case there are eight machines $\{M_1, M_2, ..., M_8\}$ and three sources for order entry $\{L_1, L_2, L_3\}$. The sources also represent the exit stocks of the production system. The buffer capacities vary between two and four orders per machine. The stochastic machine failures are defined by the parameters Mean Time Between Failure (MTBF) and Mean Time To Repair (MTTR).

As illustrated in Fig. 2, the eight machines are arranged in three areas and one source is assigned to each area. Furthermore, similar machines are grouped as followed: $G_1 = \{M_1\}$, $G_2 = \{M_2, M_3, M_4, M_5\}$ and $G_3 = \{M_6, M_7, M_8\}$. Depending on the work content, incoming orders are assigned to the source of an area containing at least one machine that is able to process that order. L_1 for example provides orders for G_1 and G_2 but not for G_3.

Besides the production environment, the reward function is defined with the two performance measures workload $WL = F_1$ of the machines, i.e. utilization, and the throughput time $TPT = F_2$ of the processed orders. The rewards r_1 and r_2 are modeled as linear functions of the performance measure values.

The action and state representation are adapted as follows. In total, there are 37 possible actions (12 actions A_{Aloc}, 24 actions A_{Empty} and 11 actions A_{Empty}). The system state is described in a vector with 47 elements.

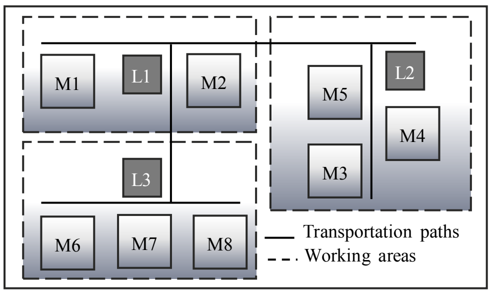


Fig. 2. Schema of the production layout.

4.2. Experimental setup

In order to apply the dispatching agent on the use case the use case production system is simulated in a series of experiments. Each simulation run has a duration of three million iterations. During one simulation run the reward, the throughput time and the workload are recorded and analyzed. The results of mainly three simulation experiments are presented hereinafter:

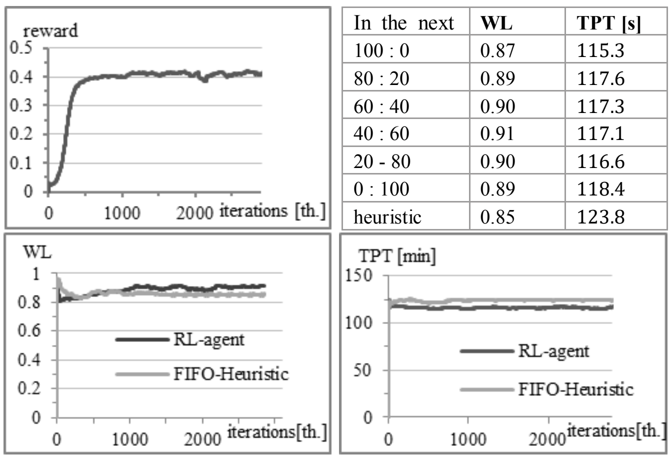
- Comparing the autonomous agent-based dispatching with a rule-based benchmark
- Modifying the weights of the two different performance measures (while keeping the sum of the weights constant)
- Performance evaluation of the agent in a slightly modified scenario with different buffer capacities

As a benchmark, a rule-based heuristic is implemented: The two layered decision rule prioritizes orders with regard to their waiting time (FIFO) and then selects the machine with the least workload.

4.3. Experimental results

The reward values and the two performance measures are shown in Fig. 3. For the first simulation experiment (see section 4.2), the course of the reward signal shows an overall successful learning process. It results from a strategy improvement with regard to the defined objectives. A further evaluation of the agent's self-learned strategy requires a closer look at the performance measures *WL* and *TPT*. In comparison with the benchmark heuristic, the agent achieves better results to the end of the learning process for both objectives and thus reaches a higher performance.

In many dispatching problems, the task includes conflictive objectives. To a certain extent, *WL* and *TPT* are conflictive objectives, too. A maximum workload usually results in a maximum exploitation of existing buffer capacities to ensure a constant supply of the machines. However, this leads to increasing order waiting times and therefore longer throughput times.



In the next	**WL**	**TPT [s]**
100 : 0	0.87	115.3
80 : 20	0.89	117.6
60 : 40	0.90	117.3
40 : 60	0.91	117.1
20 - 80	0.90	116.6
0 : 100	0.89	118.4
heuristic	0.85	123.8

Fig. 3: Reward signal (top-left), workload (bottom-left), throughput time (bottom-right) and variations of the reward weights (top-right).

Table 2. Performance results of different buffer capacities.

	WL agent	WL heuristic	TPT Agent	TPT Heuristic
Small Buffers	0.79	0.79	93.6 s	97.9 s
Large Buffers	0.94	0.90	145.5 s	155.2 s

In order to show that the presented RL architecture is a suitable approach to solve problems in a dynamic environment the agent is tested in a modified scenario. In this simulation experiment, a production system layout with changing buffer capacities is investigated. In this scenario, smaller buffer capacities result in smaller performance differences between the agent and the heuristic with regard to both performance measures (see Table 2). This is due to the fact, that larger buffers lead to more decision options in each iteration while smaller buffers limit the number of options within the problem. Therefore, the agent that always considers all options achieves overall better results than the heuristic when increasing the buffer capacities.

5. Conclusion

The implementation of an autonomous order dispatching agent in a real-world production environment shows that a RL agent is able to outperform an existing benchmark heuristic. As the architecture of RL agents is autonomous, self-developing and adaptive to changes it is a promising response to the current demands for flexible production systems and a real-time capable as well as adaptive production control.

Based on the implementation and experiments in this paper further research in modified scenarios is possible. The proposed architecture can be extended to a multi-agent production system by adding for example another dispatching agent that is referred to a second transportation resource. Another agent-based system could be designed for a learning-based and autonomous order release mechanism or even for predictive maintenance.

The architecture described in this paper still requires customization effort to the particular dispatching scenario, for example when defining the state and action representation or the reward function. A generic formulation of these components and variable would be an important step to reduce the implementation effort of autonomous control mechanisms.

Acknowledgments

We extend our sincere thanks to the German Federal Ministry of Education and Research (BMBF) for supporting this research project 02P14B161 “Empowerment and Implementation Strategies for Industry 4.0”.

References

[1] Mönch L, Fowler JW, Mason SJ. Production planning and control for semiconductor wafer fabrication facilities. 1st ed. New York: Springer; 2013.
[2] Monostori L, Csáji BC, Kádár B. Adaptation and Learning in Distributed Production Control. CIRP Annals 2004;53:349-352.
[3] Csáji BC, Monostori L, Kádár B. Reinforcement learning in a distributed market-based production control system. Advanced Engineering Informatics 2006;20:279-288.
[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] Monostori L, Váncza J, Kumara SRT. Agent-Based Systems for Manufacturing. CIRP Annals 2006;55:697-720.
[6] Lawler EL, Lenstra JK, Kan AHR, Shmoys DB. Sequencing and Scheduling: Algorithms and Complexity. Handbooks in operations research and management science 1993;4:445-522.
[7] Wang YC, Usher JM. Application of reinforcement learning for agent-based production scheduling. Engineering Applications of Artificial Intelligence 2005;18:73-82.
[8] Luck M, McBurney P. Agent Technology Roadmap. 1st ed. Southampton: AgentLink; 2005.
[9] Russel S, Norvig P. Artificial intelligence. 3rd ed. Malaysia: Pearson Education Limited; 2016.
[10] Sutton RS, Barto AG. Reinforcement Learning: An Introduction. 1st ed. Cambridge: MIT press; 1998.
[11] Schulman J, Wolski F, Dhariwal P, Radford A, Klimov O. Proximal Policy Optimization Algorithms. arXiv preprint:1707.06347, 2017.
[12] Schulman J, Levine S, Moritz P, Jordan MI, Abbeel P. Trust Region Policy Optimization. International Conference on Machine Learning 2015;1889-1897.
[13] Mnih V, Kavukcuoglu K, Silver D, Graves A, Antonoglou I, Wierstra D, Riedmiller M. Playing Atari with Deep Reinforcement Learning. arXiv preprint:1312.5602, 2013.
[14] Kormushev P, Calinon S, Caldwell DG. Robot motor skill coordination with EM-based Reinforcement Learning. Intelligent Robots and Systems 2010; 3232-3237.
[15] Philip T, Michael B, Antonie vdB, Kathleen J. Application of the Actor-Critic Architecture to Functional Electrical Stimulation Control of a Human Arm. Proc Innov Appl Artif Intell Conf. 2009;165-172.
[16] Günther J, Pilarski PM, Helfrich G, Shen H, Diepold K. Intelligent laser welding through representation, prediction, and control learning. Mechatronics 2016;34:1-11.
[17] Stegherr F. Reinforcement Learning zur dispositiven Auftragssteuerung in der Varianten-Reihenproduktion. Herbert Utz Verlag, 1st ed. 2000.