Available online at www.sciencedirect.com

ScienceDirect

Procedia CIRP 103 (2021) 170–175

www.elsevier.com/locate/procedia

9th CIRP Global Web Conference – Sustainable, resilient, and agile manufacturing and service operations : Lessons from COVID-19

Reinforcement Learning Based Production Control of Semi-automated Manufacturing Systems

Leonard Overbeck*[a], Adrien Hugues[a], Marvin Carl May[a], Andreas Kuhnle[a], Gisela Lanza[a]

[a]*wbk Institute of Production Science, Karlsruhe Institute of Technology, Kaiserstr. 12, 76131 Karlsruhe, Germany*

* Corresponding author. Tel.: +49 152 3950 2641; *E-mail address:* leonard.overbeck@kit.edu

Abstract

In an environment which is marked by an increasing speed of changes, industrial companies have to be able to quickly adapt to new market demands and innovative technologies. This leads to a need for continuous adaption of existing production systems and the optimization of their production control. To tackle this problem digitalization of production systems has become essential for new and existing systems. Digital twins based on simulations of real production systems allow the simplification of analysis processes and, thus, a better understanding of the systems, which leads to broad optimization possibilities. In parallel, machine learning methods can be integrated to process the numerical data and discover new production control strategies. In this work, these two methods are combined to derive a production control logic in a semi-automated production system based on the chaku-chaku principle. A reinforcement learning method is integrated into the digital twin to autonomously learn a superior production control logic for the distribution of tasks between the different workers on a production line.
By analyzing the influence of different reward shaping and hyper-parameter optimization on the quality and stability of the results obtained, the use of a well-configured policy-based algorithm enables an efficient management of the workers and the deduction of an optimal production control logic for the production system. The algorithm manages to define a control logic that leads to an increase in productivity while having a stable task assignment so that a transfer to daily business is possible. The approach is validated in the digital twin of a real assembly line of an automotive supplier.
The results obtained suggest a new approach to optimizing production control in production lines. Production control shall be centered directly on the workers' routines and controlled by artificial intelligence infused with a global overview of the entire production system.

Keywords: Machine Learning, Reinforcement Learning, Digital Twin, Production Control, Task Allocation, Productivity

1. Introduction

A dynamic environment and a shortening of product life cycles force manufacturing to increase the flexibility of their production systems. Semi-automated assembly cells are a production system type that offers high flexibility with regard to output volume, material flow and product types. Yet, the optimal control of such a flexible system poses a challenge given the frequently changing optimization conditions. Classical analytical optimization methods are in many cases too slow and costly to be repeated often. One possible solution to automatically find and quickly adapt (after an initial learning period) a good control strategy for each system configuration could be the use of artificial intelligence (AI). The following work presents the application of a specific AI-tool,

10.1016/j.procir.2021.10.027

reinforcement learning (RL), on the problem of production control in a semi-automated production system.

2. Literature review

[1] present a successful application of RL for order dispatching in semiconductor manufacturing. The selection of an optimal production control agent depending on the current situation using a digital twin is presented by [2]. [3] design a control of dispositional orders by RL in the case of series production, whereas [4] develop a strategy for controlling shop floor inventories that are customer-oriented and versatile with reinforcement learning. [5] adopt a neural reinforcement learning approach in production planning to learn local allocation policies. [6] use a neural network trained with RL for a job-shop scheduling task, while [7] propose a deep reinforcement learning method to optimize global production scheduling in complex shops. RL is used by [8] to optimize the throughput of a transfer line. Multi-agent reinforcement learning methods for production control are also developed by [9], [10] and [11]. [12] adopt the IRT approach to compare an RL algorithm against human performance on two simple job-shop scheduling examples. RL for AGV routing is developed and analyzed by [13].

Although RL methods are already used for industrial applications and in particular for production control, this field of study is still in its infancy and no work has so far focused on the direct optimization of worker’s tasks distribution in a semi-automated production line. As shown, the exisiting literatur is more generally focused on the control of resources or the quality of order dispatching in a production system. Our work, therefore, has the ambition to study a new control task, based this time directly on the optimization of workers' activities in a linear, semi-automated production line.

3. Own Approach

3.1. Problem description

In a semi-automated production system, human operators interact with machines that require manual operations to function. When a semi-automated production line is clocked, the actions of the operators must be particularly coordinated with the work rhythm of the machines to avoid losing time and creating efficiency losses. Indeed, one of the main problems with this type of production strategy is that the smallest interruption can have a significant impact and generate so-called deadlock situations that are harmful to the productivity of the production line. The distribution of tasks between operators must be synchronized with the machine processes in order to maintain a fluid, smooth production and reduce productivity losses. In order to distribute the tasks among operators in an optimal way, the production line is modeled with a simulation software consisting of machines, operation principles and operators.

Based on this model, an RL method can be used to control the distribution of workers' tasks in an optimal way, i.e. in such a way that deadlock problems are avoided and workers' productivity is maximized.

3.2. Reinforcement learning

Reinforcement learning is a machine learning method in which an agent takes actions $a_t \in A$ based on the state $s_t \in S$ of the environment at time t, where the action space A is the set of all valid actions in a given environment and S is the set of states, and in return receives a reward r_t depending on the impact of its action that leads to the next state $s_{t+1} \in S$. The strategy for selecting actions is referred to as the policy π with the agent's objective of learning an optimal policy, i.e. one that maximizes the cumulative reward in the long run. This principle is based on a Markov Decision Process (MDP) model which relies on the basic Markov property stating the absence of memory of a process, i.e. that the probability of the future state s_{t+1} depends only on the present state and action s_t and a_t and not on past states and actions [14]. Figure 1 describes the interaction of a reinforcement learning agent with the environment.

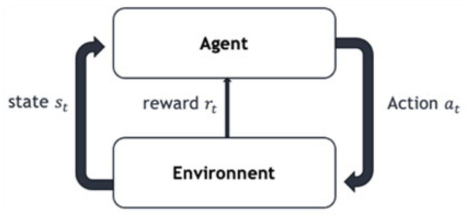


Figure 1: Interaction of reinforcement learning agent with environment [19]

Thus, for using a reinforcement learning agent, it is necessary to properly model the states of the environment, the actions that the agent can take in this environment and also the reward function that enables the agent to understand the quality of the actions taken in respect to reaching the optimal policy.

3.2.1. RL for worker control

The environment is in this case an event-discrete simulation of a semi-automated production line modeling the routines of the workers where the different actions per worker and their activities in the production line are controlled by the agent. Figure 2 shows how the agent is used to control the worker’s actions. When a new action has to be performed, it is put in a waiting list until a worker is available. When a worker is available, the RL agent assigns an action from the waiting list and the worker then performs that action it was delegated from the centralized agent. This triggers a new state in the environment that leads to new actions in the waiting list. The principle of using a reinforcement learning method is that the agent learns to allocate the optimal action from the waiting list, i.e. the one that globally helps in reaching the best productivity of the worker by trying to maximize the reward it gets after each allocation.

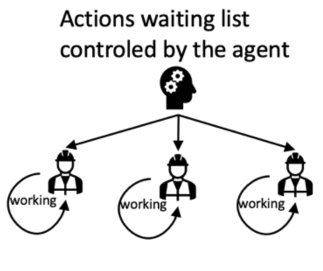


Figure 2: Agent-worker interaction

3.3. Selection of a reinforcement learning method

There are many algorithms implementing reinforcement learning and the selection of an RL algorithm is an important step when designing a reinforcement learning framework because although based on the same principle, ease of use and performance differ widely between different algorithms for each use case. Two main families are known: model-based algorithms, used for example to play chess with AlphaZero [15], and model-free algorithms. As there is no model known for this kind of problems, this work focuses on model-free algorithms. Two types of algorithms constitute the family of model-free algorithms, value-based algorithms and policy-based algorithms.

3.3.1. Value-based algorithms

Value-based methods involve determining the optimal policy for the reinforcement learning algorithm by optimizing the action-value function $Q_\pi(s_t, a_t)$ that gives the expected reward after starting from state s_t, having taken action a_t and acting according to policy π. Common value-based algorithms include Q-learning [16] and deep Q-networks (DQN) [17]. Both methods are off-policy methods, i.e. they use different policies for selecting actions and optimizing the optimal target policy, which poses reliability problems. Furthermore, the use of a DQN algorithm has shown that it is difficult to find the ideal parameters to obtain reliable and reproducible results. This and a bad sampling efficiency hampering the speed of the algorithm made it easier to work with a policy-based method.

3.3.2. Policy-based algorithms

Rather than using approximations of value functions, policy-based algorithms approximate the policy directly with a gradient-based method and are thus a more direct way to learn the optimal policy [14]. Algorithms using this type of method include the vanilla policy gradient algorithm (VPG), the trusted region policy gradient algorithm (TRPO) [18] and the proximal policy optimization algorithm (PPO) [19]. The three algorithms have the same operating principles, but PPO uses techniques that solve some of the shortcomings of the other methods, such as variance problems and computational complexity. Schulman et al. also show that this method on a collection of benchmark tasks outperforms both TRPO and VPG and is easier to implement [19]. Following the various disadvantages linked to the use of a value-based algorithm presented in the previous paragraph, we decided to work with a policy-based algorithm and more specifically a PPO algorithm for the rest of our study.

3.4. Design of the RL framework

This section highlights how the RL agent framework is designed to be integrated into a simulation of a semi-automated production system. An artificial neural network (ANN) is used to approximate the agent's optimal policy. The first layer of the ANN takes as input the state of the environment, then, thanks to the policy determined by the weights of the hidden layers of the ANN, the last layer returns the selected action, i.e. the one with the highest activation value. Following this selection, the state of the environment is modified by the action, and the policy is adapted according to the reward obtained.

The following sections show how the action space and the states of the environment are modeled as well as how the reward function is defined to solve the worker control problem.

3.4.1. Action space representation

The action space represents all the actions that the agent can take. Thus, its definition follows from the principle per agent described in the RL for worker control section. The only constraint to define the action space of the RL agent is that it must contain a fixed number of actions that cannot vary over time. However, in reality, the number of actions to be performed in the production line is never identical, in particular during the production start-up when all the machines are not yet in use and, thus, fewer actions have to be performed by the workers than when the production is in a steady state. To easen this constraint all the actions that can be performed by a worker on the production line are listed and fixed, even if it is possible that at the moment t an action is not actually performed.

The action space is, thus, defined by the sum of all the actions that can be performed manually for each machine of the production line. Let $[M_1, \dots, M_m], m \in \mathbb{N}$ be the set of machines of the production line and $[A_1, \dots, A_n], n \in \mathbb{N}$ the set of all possible actions that can be performed at a machine. The action space is then the set $\Omega = [A_1M_1, \dots, A_nM_m]$ with A_iM_j the action i on machine j.

3.4.2. State representation

The state definition is a crucial element that influences the learning process because it is by observing the state that the agent selects the next action [14]. In the case of a production line, a lot of information can be used to describe the state of the environment. However, the more information there is, the more difficult it is for the agent to construct a link between the chosen action and the state of the input environment. For this reason, the description has to be simplified as much as possible and the environment described only by the actions that can be performed by the workers on the same principle as the description of the action space. The difference, however, is that the state of the environment is used as an input vector to the neural network and can therefore be encoded with binary values, to facilitate data processing in the ANN.

Thus, binary values can be used here to distinguish between actions that are actually available and those that are not available at the time the agent selects an action. The state vector is then represented by $S = [A_1M_1, .., A_1M_m, \dots, A_nM_1, .., A_nM_m]$ with $A_iM_j = 1$ if the action A_i on the machine M_j is to be performed and $A_iM_j = 0$ if the action A_i on the machine M_j is not to be performed in the moment the agent is called to select an action. The RL-agent therefore knows all the actions that are available as well as those which are not and only has to choose the best one among the available actions. Moreover, when an action is selected by the RL-agent, the state value of this action is reset to 0 to indicate that it has been taken over by a worker, which allows to show the change of state in a sufficiently transparent way.

Figure 3 gives an example of possible decisions of the agent over time.

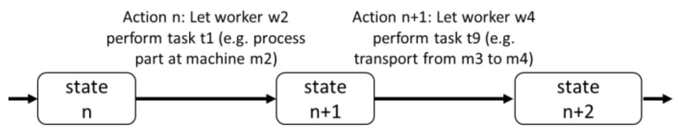

Action n: Let worker w2 perform task t1 (e.g. process part at machine m2)
Action n+1: Let worker w4 perform task t9 (e.g. transport from m3 to m4)
state n
state n+1
state n+2

Figure 3. Visualization of the agents decisions

3.4.3. Reward function

The reward function is used to make the agent understand whether the selected action was good or not and, thus, make the agent adjust its selection policy according to the feedback.

The reward function must therefore be designed according to the objectives of the RL framework. In this case, the main objective is to define a logic control for the workers that maximizes their productivity on the production line.

Maximizing productivity implies using the available resources in the best possible way. In the present case of semi-automatic assembly, machines and employees play an essential role and can therefore be included in the development of a control logic through the reward function. The idea is to combine resource utilization with a well-known production control concept, the push flow production, defined in [20]. The principle is therefore built on using the reward function to map the application of a central control (push) to obtain a maximum utilization rate of the machines in the production line.

Concretely, this solution is implemented in the simulation by rewarding the RL agent each time a machine completes a process. The underlying idea is therefore to maximize the number of processes that a machine performs and consequently the number of parts that pass through this machine and finally the number of parts produced overall in the production line.

A second possibility to design the reward function is based on the analysis of production line modeling. When the agent assigns an available action to be performed, the action is in fact not always immediately feasible because the machine may still be used by a previous part. Thus, although an action is available, it is not always feasible and, in this case, it is said to be non-feasible. The idea resulting from this analysis is to punish the RL agent when the agent chooses a non-feasible action. In fact, the less time workers waste on working on non-feasible actions, the more efficiently they will work, resulting in higher productivity.

To take into account the two ideas underlying the push and action-feasibility functions a reward function that adds these two functions together and then rewards the agent for how well the selected action meets the objectives defined in these two functions is used.

3.4.4. Implementation of the production system simulation

The different components describing the RL framework presented in the previous sections have been integrated in a simulation of a semi-automated production line built with the python library SimPy. The simulation models the movement of workers on the production line and the production stages of a part, i.e. its progression through the various machines making up the production line. The simulation has been designed in an episodic way so that the agent can be trained by repeating the same simulation as many times as necessary until the optimal policy is found. The RL agent used comes from the tensorforce library written in python and based on tensorflow.

3.5. Evaluation

3.5.1. Performance of the RL algorithm

After defining the different elements of the reinforcement learning framework and the parameters of the PPO algorithm, different performance criteria are evaluated, in particular the progression, stability and convergence of the learning process. Thus, different simulation configurations were tested to observe the influence of the duration of a training episode, the number of training episodes or the number of workers working in the production line on the learning process of the RL agent.

Each time, the analysis of the evolution of the reward received per episode is the first step. The reward should increase as the simulation episodes progress and then stagnate when the optimal policy is reached. In a second step, in order to check that the reward function corresponds to our initial objectives, the evolution of the number of pieces produced per simulation episode is also analyzed and compared with the result obtained without any RL-algorithm.

3.5.2. Analysis of worker's behavior

When the algorithm has converged to an optimal policy, a control logic for the workers can be determined from the results obtained. The criteria for observing this control logic are the number of worker transitions between each machine and the percentagewise deviation between the average time spent working on each machine per worker. These measures make it possible to analyze how the tasks are distributed between the workers by the agent, for example whether each worker is responsible for a fixed number of machines or they have a similar behavior or if they are all responsible for all the machines.

4. Application

4.1.1. Use Case

The methodology was developed in a research partnership between of the wbk Institute for Production Technology at the Karlsruhe Institute of Technology (KIT) and the central department Connected Manufacturing of the Bosch Powertrain Solutions division with the goal to develop an agile production system. Its application and validation are also part of this joint research project.

The approach was implemented and tested in the simulation model of a real world production system for car engine components which is organized in a cell composed of two lines facing each other. The assembly cell is semi-automated, following the Chaku-Chaku principle. This means that the machines perform their processes on their own and workers are mainly required for loading and unloading of machines and transporting parts between them. The line produces various product types with differing material flows, processing times, etc. The default number of workers in the production cell is 5, but can deviate due to external factors such as vacations, sick days, reduced customer demand, trainings, etc. For each number of workers an optimal distribution of task between the workers has to be found for each article type. There are 9 machines in the manufacturing cells and 29 possible tasks for the worker to do.

4.1.2. Hyperparameter selection

Hyperparameters are used to adjust the RL algorithm, in this case the PPO algorithm. Among the most important parameters we find the following. The batch size is the number of data points collected between each update of the ANN weights, this parameter influences the speed and stability of the learning process. The update frequency parameter indicates how often the policy is updated. The learning rate α corresponds to the step size of the ANN-update. The parameter discount γ reduces the anticipation of rewards that lie further in the future. Regularization parameters like L2-regularization or entropy regularization are additional parameters that prevent the algorithm from overfitting the policy [21] or the policy to become deterministic and converge to a poor policy too quickly [22]. After a series of tests, comparisons and analysis of the results obtained by varying the hyperparameters, those that give the best results are presented in Table 1.

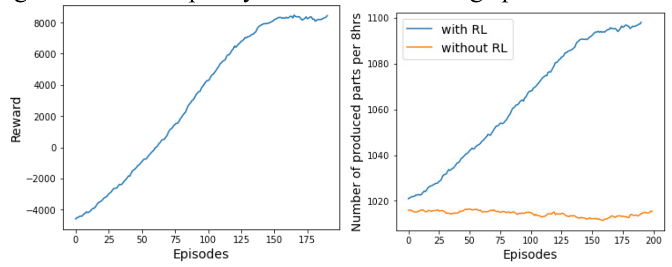
Table 1: Hyperparameters values of the PPO-algorithm.

Hyperparameter	**Value**
Batch size and update Frequency	20 episodes
Learning rate	0.00001
Discount	0.9
Entropy regularization	0.01
L2-regularization	0.01

5. Results

This section outlines the various results obtained by integrating the RL framework presented in the previous sections into the case study. Figure 3 and Figure 4 show the evolution of the reward and the number of pieces produced per episode as the agent's learning process progresses. The duration of each episode is eight hours, i.e. the duration of a production shift, and five workers are working in the production line.

Figure 4 highlights the increasing reward over the episodes which shows that the RL agent learns to maximize it over time and experience. This evolution is stable over time and converges from episode 150 onwards, thus showing that the agent can learn a policy after several training episodes.



(a) (b)

Figure 4: Evolution of the (a) reward per episode and (b) produced parts per episode

The blue curve in Figure 4 shows that the number of parts produced per episode also grows with the progress of the learning process of the RL agent. Thus, the growth of the reward can be associated with a growth of the workers' productivity, which satisfies the initial objective. On the other hand, the orange curve shows the results for the same simulation run but without using the RL framework.

It can be seen that at the beginning the number of pieces produced per episode is quite similar but while without the RL agent this number remains stable, the RL agent significantly improves the output. It would even seem that unlike the evolution of the reward which seems to converge after 150 episodes, the number of produced parts still tends to potentially evolve with a longer simulation. An improvement of 8% in the number produced pieces is observed.

Moreover, Figure 5 illustrates that the number of workers on the production line does not change the learning capacity of the agent and that the number of produced parts per episode still increases throughout the learning process.

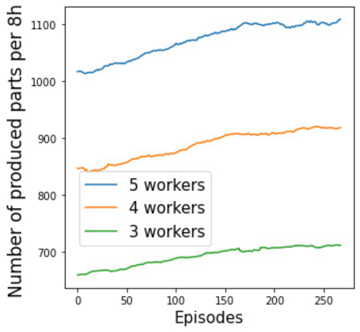


Figure 5: Comparison of the number of produced parts per 8 hours with 3, 4 or 5 workers in the production system

Figure 6 represents the transitions of two workers between machines during the last episode of the simulation, i.e. the one where the largest number of parts is produced. First, the two heatmaps are very similar, i.e. the workers perform globally the same actions on the production line. Secondly, the transitions are on average distributed between all the machines which shows that the optimal configuration is the one in which all workers work on all the machines rather than one worker being responsible for a group of machines in particular. The movement of workers seems to follow the same path as a part through the production line. These movements are more regular between the machines M1 and M5 because the transport of the parts is manual, in contrast to the transport between the machines M5 and M9, which is performed by conveyor.

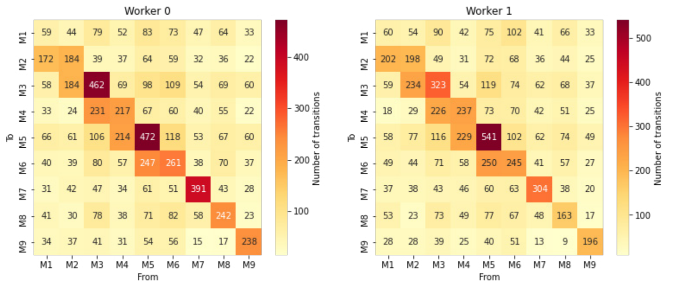


Figure 6: Workers transitions between machines

Figure 6 also shows that many transitions occur between the same machines, i.e. workers regularly perform several actions in a row on the same machine before switching. However, the heatmap does not really show the differences in work time on each machine per worker, which is better illustrated in Figure 7, which shows the percentage deviation in work time of each worker on each machine compared to the overall average work time of workers on each machine. The maximum deviation does not exceed 15%, i.e. the working times of each worker on the machines are pretty similar. This maximum deviation is reached at machine M7, where worker 0 works 15% less than the average work time of the workers on this machine, and worker 4 counterbalances this difference by working 15% more. At machine M2, worker 2 offsets the work deficit on this machine of workers 0, 3 and 4.

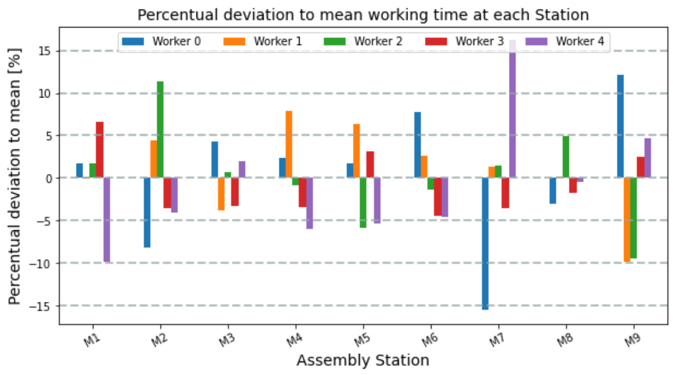


Figure 7: Comparison of the workers working time on each machine

6. Conclusion and Outlook

Based on a chosen PPO agent and a thorough hyperparameter tuning a functioning RL agent is implemented which is able to efficiently allocate tasks between workers in a semi-automated production line. This is evaluated in a real world assembly cell for automotive parts. It can be shown that the RL agent improved its decision quality over time and a stable distribution of tasks was reached for a given simulation environment.

Further research has to investigate how the learned control logic can be implemented in the real world, since the control of workers on a granularity of seconds is in reality impossible. Another line of research can be the improvement of robustness of the agent under changing layouts of the production system. Furthermore the agent has to be trained in different environments to prove its versatility.

Acknowledgements

This research was funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) - DFG 416818066 “Planungsmethode für eine veränderungsfähige Bandabstimmung von getakteten Fließmontagesystemen“.

References

[1] Kuhnle, A., Schäfer, L., Stricker, N., Lanza, G., 2019. Design, Implementation and Evaluation of Reinforcement Learning for an Adaptive Order Dispatching in Job Shop Manufacturing Systems. Procedia CIRP 81 (7676), 234–239.
[2] May, M. C., Overbeck, L., Wurster, M., Kuhnle, A., & Lanza, G., 2020. Foresighted digital twin for situational agent selection in production control. Procedia CIRP.
[3] Stegherr, F., 2000. Reinforcement-Learning zur dispositiven Auftragssteuerung in der Variantenreihenproduktion. Dissertation. Herbert Utz Verlag, München.
[4] Scholz-Reiter, B., Hamann, T., 2008. The behaviour of learning production control. CIRP Annals 57 (1), 459–462.
[5] Riedmiller, S., Riedmiller, M., 1999. Riedmiller, Simone, and Martin Riedmiller. "A neural reinforcement learning approach to learn local dispatching policies in production scheduling. IJCAI (2), 764–771.
[6] Zhang, W., Dietterich, T.G., 1996. High-performance job-shop scheduling with a timedelay TD network. Advances in neural information processing systems 8, 1024–1030.
[7] Waschneck, B., Reichstaller, A., Belzner, L., Altenmüller, T., Bauernhansl, T., Knapp, A., Kyek, A., 2018. Optimization of global production scheduling with deep reinforcement learning. Procedia CIRP 72 (7540), 1264–1269.
[8] Mahadevan, S., Theocharous, G., 1998. Optimizing Production Manufacturing Using Reinforcement Learning. FLAIRS Conference 372, 377.
[9] Dittrich, M.-A., Fohlmeister, S., 2020. Cooperative multi-agent system for production control using reinforcement learning. CIRP Annals 69 (1), 389–392.
[10] May, M.C., Kiefer, L., Kuhnle, A., Stricker, N., Lanza, G., 2021. Decentralized Multi-Agent Production Control through Economic Model Bidding for Matrix Production Systems. Procedia CIRP 96, 3–8.
[11] Malus, A., Kozjek, D., 2020. Real-time order dispatching for a fleet of autonomous mobile robots using multi-agent reinforcement learning. CIRP Annals 69 (1), 397–400.
[12] Burggräf, P., Wagner, J., Koke, B., Bamberg, M., 2020. Performance assessment methodology for AI-supported decision-making in production management. Procedia CIRP 93 (7587), 891–896.
[13] Lu, C., Long, J., Wu, W., Gu, Y., Lou, J., Huang, Y., 2020. Deep Reinforcement Learning for Solving AGVs Routing Problem. International Conference on Verification and Evaluation of Computer and Communication Systems, 222–236.
[14] Sutton, R.S., Barton, A.G., 1998. Reinforcement learning: An introduction. MIT Press, 548 pp.
[15] Silver, D., Hubert, T., Schrittwieser, J., Antonoglou, I., Lai, M., Guez, A., Lanctot, M., Siefre, L., 2017. Mastering Chess and Shogi by Self-Play with a General Reinforcement Learning Algorithm. arXiv preprint (preprint arXiv:1712.01815.).
[16] Watkins, C.J.C.H., Dayan, P., 1992. Q-Learning. Machine Learning (8), 279–292.
[17] Mnih, V., Kavukcuoglu, K., Silver, D., Antononglou, I., Wierstra, D., Riedmiller, M., 2013. Playing Atari with Deep Reinforcement Learning. arXiv preprint arXiv:1312.5602.
[18] Schulman, J., Levine, S., Moritz, P., Jordan, M., Abbeel, P., 2015. Trust Region Policy Optimization. International conference on machine learning, 1889–1897.
[19] Schulmann, J., Wolski, F., Dhariwal, P., Radford, A., Klimov, O., 2017. Proximal Policy Optimization Algorithms. arXiv preprint arXiv:1707.06347.
[20] Westkämper, E., 2006. Einführung in die Organisation der Produktion. Springer, Berlin Heidelberg New York, 263 pp.
[21] Wang, K., Kang, B., Shao, J., Feng, J., 2020. Improving Generalization in Reinforcement Learning with Mixture Regularization, in: 34th Conference on Neural Information Processing Systems (NeurIPS 2020), Vancouver, Canada.
[22] Zafarali Ahmed, Nicolas Le Roux, Mohammad Norouzi, Dale Schuurmans, 2019. Understanding the Impact of Entropy on Policy Optimization. International conference on machine learning, 151–160.