Procedia CIRP 97 (2020) 104–109

Contents lists available at ScienceDirect

Procedia CIRP

journal homepage: www.elsevier.com/locate/procir

Dynamic scheduling in a job-shop production system with reinforcement learning

Csaba Kardos[a], Catherine Laflamme[a], Viola Gallina[a,*], Wilfried Sihn[a,b]

[a] Fraunhofer Austria Research GmbH, Theresianumgasse 27, Vienna 1040, Austria
[b] Vienna University of Technology, Institute of Management Science, Theresianumgasse 27, Vienna 1040, Austria

ARTICLE INFO

Article history:
Received 27 September 2019
Revised 22 April 2020
Accepted 18 May 2020

Keywords:
Dynamic scheduling
Reinforcement learning
Simulation
Smart factory

ABSTRACT

Fluctuating customer demands, expected short delivery times and the need for quick order confirmation creates a fast-paced scheduling environment for modern production systems. In this turbulent scene, using the data provided by intelligent elements of cyber-physical production systems opens up new possibilities for dynamic scheduling. The paper introduces a reinforcement learning approach, in particular Q-Learning, to reduce the average lead-time of production orders in a job-shop production system. The intelligent product agents are able to choose a machine for every production step based on real-time information. A performance comparison against standard dispatching rules is given, which shows that in the presented dynamic scheduling use-cases the application of RL reduces the average lead-time.

1. Introduction

Job-shop scheduling is one of the most relevant problems currently faced in manufacturing, in particular for systems where meeting customer demand requires a high degree of flexibility. As manufacturers adapt to this demand, changes are made to increase their system flexibility, which necessarily increases the overall system complexity. This results in the need for new methods to replace standard processes which are not able to sufficiently adapt to the more complex environment. Several approaches can be found in the literature to handle this increasing complexity of the production environment (e.g. genetic algorithms (Chryssolouris and Subramaniam, 2001), supervised learning methods (Lingitz et al., 2018)). One of the most promising methods to address this increase in complexity is reinforcement learning (RL), due to its decentralized, autonomous, self-learning and self-optimizing nature. RL, a subset of the broader field of machine learning, has, in the last years, already been successfully applied to different real world control problems, for example, in game control or vehicle routing Sutton and Barto, (Jhung et al., 2018; Mnih et al., 2015). In addition to this broader scope of applications, numerous works have explored the use of RL in production settings, including for maintenance (Kuhnle et al., 2019a; Mourtzis and Vlachou, 2018), process and robot control (Hafner and Riedmiller, 2011; Tsurumine et al., 2019), line configuration (Dornheim and Link, 2018), energy management (Dayani et al., 2019) and order dispatching (Stricker et al., 2018). While in itself RL is not a new method, recent developments in both the required software and hardware, as well as the recent digitalisation of industrial workplaces, have meant that a practical application in an industrial setting has only recently become possible.

In this paper the application of RL, namely Q-learning to a dynamic scheduling problem in a job-shop production system is presented. Currently, this type of scheduling problem is solved via easy-to-use heuristics – the so called dispatching rules – that can be easily implemented in the different manufacturing execution or scheduling systems. Therefore, in this work the performance of the RL algorithm is compared against the most widespread, standard heuristics, which allows for the identification of a range of problems where RL is advantageous over standard methods. This underscores the main contribution of this work: providing a quantitative baseline to understand where, in practice, RL can be expected to out-perform standard methods.

The paper is structured as follows. In the next chapter a short literature review regarding dynamic production scheduling with RL is given. Section 3 describes the developed simulation model and RL environment with the implementation. The results are presented in Chapter 4. The last section summarises the findings and gives a research outlook.

[*] Corresponding author.
E-mail address: viola.gallina@fraunhofer.at (V. Gallina).

https://doi.org/10.1016/j.procir.2020.05.210

2. Literature review

Scheduling of production processes is in the focus of extensive research efforts for a long time. Numerous subproblems exist in the field, depending on the various properties of the problem such as: the capability of the resources (e.g., deterministic vs. stochastic processing time, flexibility in the assignment, etc.) or the properties of the input (e.g., inter-arrival time of jobs, structure of the product mix, etc.), which also results in various applicable solution approaches (Zhang et al., 2019).

In complex and fluctuating production environments, however, the application of dispatching rules are still widely used, especially in dynamic, fast paced or even real-time decision making (Ouelhadj and Petrovic, 2009). A few examples of such rules are: First In First Out (FIFO), Shortest Processing Time (SPT) and Earliest Due Date (EDD). With the prevalence of the intelligent products, which are able to plan their own production routing, this concept is getting even more attention (Mourtzis and Vlachou, 2018). Also they can be effectively combined with heuristics like Genetic Algorithm (GA) or Ant Colony Optimization (Chryssolouris and Subramaniam, 2001).

For load distribution in cloud systems dynamic scheduling is widely used as the demands are typically hard to predict. With advances in AI and the available computational power, in the field of dynamic scheduling recently numerous new approaches appeared which uses RL (Peng et al., 2015; Wang et al., 2019). Also in production scheduling a growing increase in the use of RL algorithms can be observed as these problems can be viewed as sequential embedded-agent tasks and thus modelled as Markovian Decision Processes (MDP) (Singh, 1994).

The reviewed papers use various approaches for (1) the implementation of the RL algorithm and (2) for modelling the problem. Several papers use RL by applying Q-learning (Bouazza et al., 2017; Khader and Yoon, 2018; Wang and Usher, 2005), in (Waschneck et al., 2018) deep Q-learning is used, and in (Schneckenreither and Haeussler, 2019) an adapted version of Q-learning is developed. The algorithms range from being based on Q-tables (Khader and Yoon, 2018), neural networks (Kuhnle et al., 2019b; 2019c), and deep neural networks (Waschneck et al., 2018), respectively. Most cases focus on value-based algorithms (Bouazza et al., 2017; Khader and Yoon, 2018; Qu et al., 2016; Wang and Usher, 2005; Waschneck et al., 2018), however (Kuhnle et al., 2019b; 2019c) use a policy-based algorithm. Moreover, three papers use the ϵ-greedy method (Khader and Yoon, 2018; Wang and Usher, 2005; Waschneck et al., 2018). The production models can vary from flow shop applications

In Bouazza et al. (2017) the authors present an approach to deal with job-shop scheduling problem using intelligent products (IP) that collect data to the current scheduling context, require a set of services from machines. To schedule the production jobs in the simulation environment the IPs decompose the decision into two actions: machine selection and rescheduling of the chosen queue. In Stricker et al. (2018) it was shown that an RL-base adaptive order dispatching algorithm can outperform rule-based heuristic approaches. Increasing the utilisation of the equipment by 8% and reducing the lead time by more than 5% demonstrate the suitability of the RL approach. Kuhnle et al. (2019b,c) and Waschneck et al. (2018) focuse on the practical applications of RL in production control and introduce autonomous order dispatching using RL in real-word use cases. In Wang and Usher (2005) it is shown for a single machine problem that with different reward functions different system objectives can be achieved. All of these papers are dealing with one single RL agent optimising the chosen parameter. One possibility for broadening the scope and bringing the method closer to the practice is the application of multi-agent systems – e.g. (Qu et al., 2016; Vamplew et al., 2010; Wang et al., 2019).

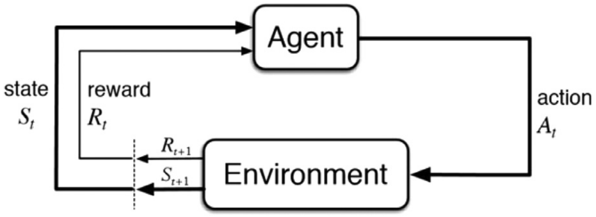

Fig. 1. The RL environment with its elements.

The goal of this paper is to demonstrate the application of RL for dynamic production scheduling problems. A simple job-shop simulation – inspired by Bouazza et al. (2017) – is step-by-step extended in order to identify the degree of complexity of the production environment where RL can be expected to out-perform standard methods.

3. Description of the model

The implementation of RL is carried out by defining an environment, typically a simulation model, in which intelligent elements, the so called agents are able to learn the best decision strategy during a trial and error training process – so called episodes in RL – in order to maximise the overall objective of the optimization (Gerlach, 2011; Sutton and Barto, 0000).The structure of the RL environment is depicted on Fig. 1.

In the presented work, the training environment for the RL algorithm was defined via a dynamic scheduling use-case. The use-case is a job-shop production scenario, where the production system was modelled by simulation, which was combined with a RL model that provided the decision making logic. The next subsections will describe the simulation model and the RL model–providing the decision logic–in detail.

3.1. Production scenario – simulation model

The job-shop production scenario consists of *Service Providers* (SP, i.e. work stations) and *Intelligent Products* (IP). Each SP is a single capacity machine combined with an infinite FIFO buffer in front of it. The machines are assumed to be 100% available, with deterministic process and setup times. The IPs are characterized by their product types, which are composed of a sequence of operations, where each operation is defined by the set of allowed SPs and the SP-dependent setup and processing times. An operation can be performed on multiple SPs, but the number of operations and their sequence is fixed. The product type of the IPs are determined by the *product-mix* model parameter, which sets the distribution of product types in a simulation-run. The simulation model was inspired by Bouazza et al. (2017), however in our case the production process can have several steps, the distribution of the production orders in each episode is random and only one decision, the machine choose is to be made. The simulated production environment can be seen on Fig. 2.

To build up the simulation model Discrete Event Simulation (DES) was used as it is a widely accepted and applied tool to analyse production systems. It offers an efficient way to capture and model the material flow processes in complex, dynamic and even stochastic systems, without the need of the formulation of an analytical model (Hedtstück, 2013). In the simulation the events defining the material flow are associated with the route of IPs and are listed below:

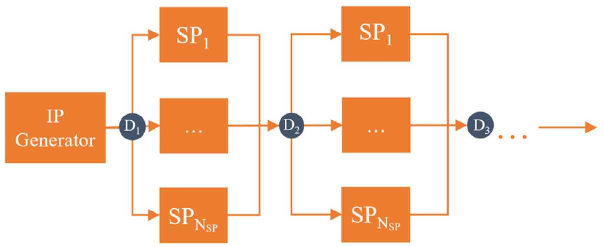

Fig. 2. Overview of the simulation model. At each decision point the RL selects the next SP for the IP.

Table 1
Processing [Setup] Times for each IP type for each process step. An “x” indicates this task cannot be completed on this SP.

Process Step 1	SP0	SP1	SP2	SP3
IP0	4 [0]	4 [1]	1 [2]	1[5]
IP1	1 [0]	x	1 [1]	1 [5]
IP2	x	3 [0]	2 [1]	2[5]
Process Step 2	SP0	SP1	SP2	SP3
IP0	4 [0]	1 [5]	1 [5]	4[0]
IP1	3 [1]	x	1 [2]	1 [1]
IP2	x	3 [1]	2 [2]	2[4]
Process Step 3	SP0	SP1	SP2	SP3
IP0	4 [0]	1 [5]	1 [5]	4[0]
IP1	3 [1]	x	1 [2]	1 [1]
IP2	x	3 [1]	2 [2]	2[4]

1. **Arrival** to the system
2. Begin of **waiting** at an SP
3. End of waiting and start of **processing** at an SP
4. **Leaving** the system

For IPs with multiple operations the events waiting and processing are repeated until all the operations are completed. The waiting event always starts at the queue of the processing SP. Therefore the decision making (i.e. selection of SP) has to be done before the waiting begins: both after the initial arrival to the system and after the completion of each processing event until no more operations left.

3.2. Decision logic – RL

As described above, the goal of this job-shop scenario is the dynamic assignment of each IP to one of the allowed SPs, optimizing with respect to the total lead time. By defining each IP as an *agent*, a RL algorithm – in this case a Q-learning algorithm – will allow the agent to make these decisions, continually improving its strategy over time.

Within this framework the decision making process is described via a Markov Decision Process, where all relevant information for the decision (and subsequent reward) is included in the state description. In the presented setup the states are described in the space $\mathcal{S} = S_{\text{IP}} \otimes S_{\text{WIP}}$, where $S_{\text{IP}} = [s_1, \ldots, s_{N_{\text{IP}}}]$ is a one-hot encoded vector identifying the IP as one of the N_{IP} total IP types $s_j \in 0, 1$ for $j = 1 \ldots N_{\text{IP}}$ and $\mathcal{S}_{\text{WIP}} = [\mathcal{S}_1, \ldots, \mathcal{S}_{N_{\text{SP}}}]$ is the vector of current WIP (Work In Progress, i.e., the number IPs in the system) values for the N_{SP} different SPs. Note that by identifying the IP type, the complete set of process and setup times is defined, (see e.g. Table 1), thus giving complete information about the production process for this IP.

Based on the information given by a given state $s \in \mathcal{S}$, the role of the agent in a Q-learning algorithm is to estimate the expected reward (in this case related to the overall lead time) in choosing each of the allowed SPs. Based on this estimation, the agent can then select the SP with the estimated maximal reward (and minimal overall lead time). Note that, in order to find the optimal policy, a balance has to be found between exploitation (choosing the action to maximize the best estimated expected reward) and exploration (choosing an action at random to try to explore the entire state space). Thus, during the training phase, the agent will follow the so-called $\epsilon-$greedy policy – selecting an SP to maximize the reward with probability $1 - \epsilon$, and choosing a random SP with probability ϵ.

3.3. Implementation

The implementation of the simulation model and the decision logic is done in the Python programming language and the Simpy process based DES framework was used. Simpy offers a lightweight, extensible library to create object oriented DES simulations. Its lightweight property is important in cases such as RL, where the large number of experiments/runs requires minimizing the possible overhead (e.g., extensive data collection, graphic interface, etc.). Another important aspect of efficiency is the tight integration of the decision logic, which was implemented using Tensorflow library. The Simpy model was wrapped into an OpenAI Gym environment, which specifies standard interfaces for using models within an RL loop and therefore makes integration simpler.

Each episode in the RL loop was executed as a simulation experiment. A simulation experiment uses a uniform distributed interarrival time as a parameter for generating the given number of IPs according to the product type distribution. The start of the experiment is the arrival of the first IP and the end is the time of completion for the given number of IPs. In order to calculate the reward in the RL, the average lead-time (time between arrival and leaving) was updated during the simulation each time an IP was completed. To eliminate the distortion caused by the different process times of the different product types, the average lead-time was further transformed. For each product type i and for each product step $s = 1, .., S_i$, given the allowed SPs $m \in M_i$ and the processing time on them ($p_{i,s,m}$), the *base lead-time* λ_i was calculated as follows:

$$\lambda_i = \frac{1}{|S_i|} \sum_{s}^{S_i} \frac{1}{|M_i|} \sum_{m \in M_i} p_{i,s,m} \tag{1}$$

The base lead-time was then used as comparison for calculating the reward, according to the following empirical formula, for product type i, with a lead-time l returned by the simulation:

$$r = \begin{cases} 1, & \text{if } l \le \lambda_i \\ 0.5, & \text{if } \lambda_i < l \le 1.5\lambda_i \\ 0.2, & \text{if } 1.5\lambda_i < l \le 4\lambda_i \\ -1, & \text{if } 4\lambda_i < l \end{cases} \tag{2}$$

The Q-learning algorithm is built on a neural network, which consists of an input layer with a total of $N_{\text{input}} = N_{\text{IP}} + N_{\text{SP}}$ neurons, a single fully connected hidden layer, and an output layer with N_{SP} neurons. The activation function used is $\tanh(z)$.

4. Results

In this section the performance of the policy learned by the RL agent is tested against the standard heuristic. As it is well established that these heuristics perform well for simple scenarios, it is, from a practical point of view, important to understand at what complexity it becomes advantageous to implement RL. To this end a set of problems with increasing complexity are defined, both by increasing the number of total process steps used and by decreasing the set time interval between dynamically generated IPs. In addition the performance of the policy is tested when the agent has the ability to directly choose an SP, compared to when the agent is limited to choosing between the standard dispatching rules.

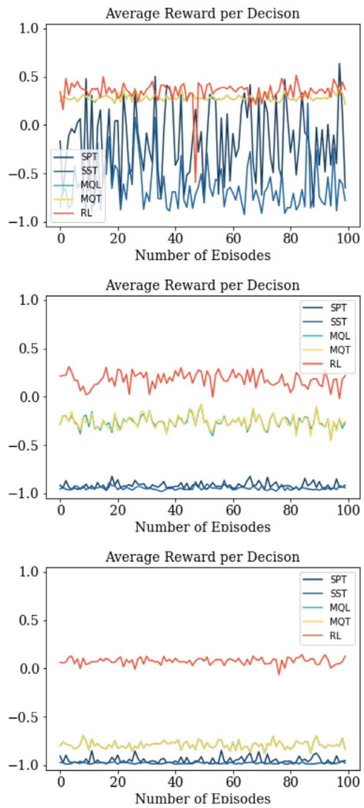

Fig. 3. Comparison of the RL and Dispatching Rules for a process with 1, 2 and 3 decision points (left to right), respectively.

In order to test performance of the Q-learning algorithm for each defined problem, the neural network is trained on a set of 500 random episodes, each episode consisting of 200 products, thus totalling $500 \cdot 200 = 10^4$ training decision points. In the training phase each IP type is chosen randomly, as is the exact time at which it arrives (chosen randomly from within a set interval). At the end of the training phase, the neural network is then tested against the standard dispatching rules: shortest processing time (SPT), shortest setup time (SST), minimal queue length (MQL), and minimal expected queue time (MQT). For this test phase, a fixed sequence of 100 episodes (again each with 200 products) is taken. Note that the "exploration" of a Q-learning algorithm is only relevant during the training, and during the test phase, only decisions based on the expected reward are taken.

4.1. Increasing problem complexity

In this section three scenarios of increasing complexity are considered, each scenario defined with a set of three IP types (IP0-IP2) and a set of four SPs (SP0-SP4)). The three scenarios considered are with one, two and three process steps, respectively. The associated process/setup times for each of the IP types for each step are those given in Table 1. Note that for this example the same SPs are used for each step – thus queues at each SP can include products in line for each of the three steps.

The results with one, two, and three process steps are shown in Fig. 3 where it can be seen that, as the problem becomes more complex (queue lengths grow, etc..), the improvement offered from the Q-learning algorithm when compared to the dispatching rules increases. This is due to the fact that the RL agent has the flexibility to adapt to the changing landscape, and make each decision individually without apply a blanket rule.

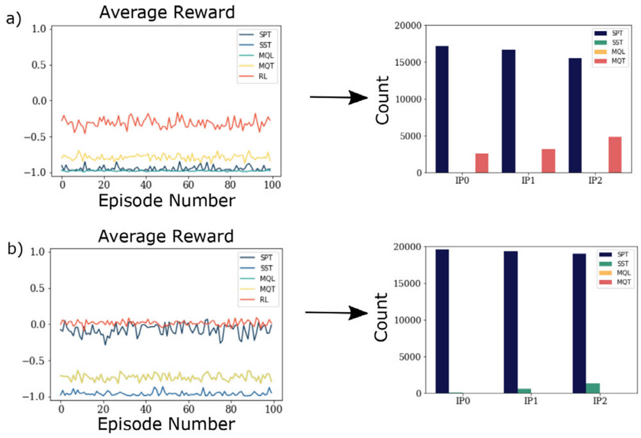

Fig. 4. Effect of increased time interval of product arrival with two levels: (a) normal, (b) increased avg. arrival time. In the left panel, the different colours indicate the different dispatching rules, SPT (Dark Blue), SST (light blue), MQT (Yellow), and the results of the RL Algorithm is indicated in (red). In the right panel, the colors indicate the selected dispatching rule used by the reinforcement learning agent SPT (Dark Blue), MQT (red) and SST (green).

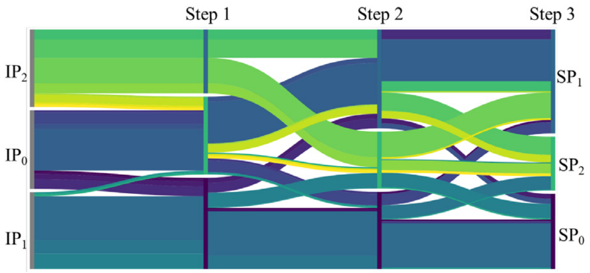

Fig. 5. A Sankey-diagram showing the routes of the IPs controlled by the RL-algorithm in the use-case. Each IP starts from a grey rectangle and then at each step the algorithm chooses an SP.

In a second example, the number of process steps are fixed (at three), but the interval is changed from which the product arrival is determined. (Note that the exact arrival interval is still chosen at random, but the max/min values are altered). The results are shown in Fig. 4, it shows that the Q-learning algorithm performs significantly better than the dispatching rules in the case of a shorter interval, and therefore in the case when longer queues are present at the SPs. Not that for this experiment was carried out with the RL agent choosing a dispatching rule at each decision point, rather than an SP directly, as this allows us to better understand when, and why, the RL agent is advantageous. In this case, when the interval is longer, and thus the queues are almost non-existent, the agent chooses almost exclusively "shortest processing time", and the results between the Q-learning algorithm and this dispatching rule are necessarily very similar. In fact, when the interval between product arrivals becomes long enough that there are no queues, this dispatching rule becomes the optimal solution, and the agent will converge to choosing this dispatching rule. However, in the case of shorter intervals and longer queues (see Fig. 4 (b)) the Q-learning agent can chooses between various dispatching rules, finding the optimal case for each specific product, resulting in a significant performance increase from the RL agent.

This behavior can be observed consistently through several examples – as the system becomes more complex, the more flexibility is required to find an optimal solution. This is additionally shown in Fig. 5, visualizing the assignment of the three IPs to, in this case, a set of three SPs over three process steps. The rising complexity over the three steps here can be clearly seen, underlining the conclusion that a Q-learning agent can be best used in

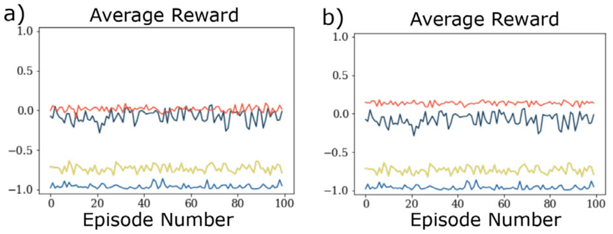

Fig. 6. Average Reward per Decision for each episode for a) Machine Selection and b) Dispatching Rule selection and their respective Machine Selection breakdown. The different colours indicate the different dispatching rules, SPT (Dark Blue), SST (light blue), MQT (Yellow), and the results of the Reinforcement Learning Algorithm is indicated in (red).

real-world scenarios where the problems are more complex - processing times are long compared to the arrival times of the new products, and the queues become necessarily long.

4.2. Machine vs. dispatching rule selection

As it was discussed in the previous section, there is a subtle difference between allowing the RL to choose between a fixed set of dispatching rules, and an SP directly. In this section the algorithm uses two different decisions that can be made: at each decision point in production process, the agent can choose a machine directly or can choose one of the dispatching rules. Here all parameters are kept fixed (processing/setup times, process steps, Product sequence, time intervals between products) and train the network on these two decision possibilities. The results are shown in Fig. 6, again confirming that as the system complexity increases, the more flexibility the agent has, the better the resulting policy is, and the ability to choose a machine directly can allow for better results than if one is restricted to selecting a dispatching rule.

5. Summary and outlook

In this paper it was demonstrated that the application of RL can successfully reduce the average lead time of production orders in a dynamic environment. In addition it was shown that as the complexity of the production environment increases, the application of RL for dynamic scheduling becomes more beneficial. In this regard, as tomorrow‘s manufacturing systems become more flexible – answering the varying customer demands and continuous disturbances of the production environment –, and necessarily more complex, the application of RL for dynamic decision support in production will certainly attract more attention from the research community. Therefore the authors future goal is the prototypical implementation of decentralised control systems for different production and logistic problems in the era of industry 4.0. In order to get closer to practical applications several topics are planned to be researched in the future – e.g. extension to a multi-agent system or combination with other elements of the production planning system.

Declaration of Competing Interest

Potential conflict of interest exists.

CRediT authorship contribution statement

Csaba Kardos: Conceptualization, Methodology, Software, Visualization, Writing - original draft. **Catherine Laflamme:** Conceptualization, Methodology, Software, Visualization, Validation, Writing - original draft. **Viola Gallina:** Conceptualization, Methodology, Validation, Investigation, Writing - original draft, Funding acquisition, Project administration. **Wilfried Sihn:** Writing - review & editing, Supervision.

References

Bouazza, W., Sallez, Y., Beldjilali, B., 2017. A distributed approach solving partially flexible job-shop scheduling problem with a Q-learning effect. IFAC-PapersOnLine 50 (1), 15890–15895. doi:10.1016/j.ifacol.2017.08.2354.

Chryssolouris, G., Subramaniam, V., 2001. Dynamic scheduling of manufacturing job shops using genetic algorithms. J. Intell. Manuf. 12 (3), 281–293. doi:10.1023/A:1011253011638.

Dayani, A.B., Fazlollahtabar, H., Ahmadiahangar, R., Rosin, A., Naderi, M.S., Bagheri, M., 2019. Applying reinforcement learning method for real-time energy management. In: 2019 IEEE International Conference on Environment and Electrical Engineering and 2019 IEEE Industrial and Commercial Power Systems Europe (EEEIC / I&CPS Europe). IEEE, Genova, Italy, pp. 1–5. doi:10.1109/EEEIC.2019.8783766.

Dornheim, J., Link, N., 2018. Multiobjective reinforcement learning for reconfigurable adaptive optimal control of manufacturing processes. In: 2018 International Symposium on Electronics and Telecommunications (ISETC). IEEE, Timisoara, pp. 1–5. doi:10.1109/ISETC.2018.8583854.

Gerlach, R., 2011. Reinforcement Learning: Roboternavigation in Heimumgebungen. Diplomica-Verl, Hamburg. OCLC: 800430508.

Hafner, R., Riedmiller, M., 2011. Reinforcement learning in feedback control: challenges and benchmarks from technical process control. Mach. Learn. 84 (1–2), 137–169. doi:10.1007/s10994-011-5235-x.

Hedtstück, U., 2013. Simulation diskreter Prozesse. Springer, Berlin, Heidelberg doi:10.1007/978-3-642-34871-6.

Jhung, J., Bae, I., Moon, J., Kim, T., Kim, J., Kim, S., 2018. End-to-end steering controller with CNN-based closed-loop feedback for autonomous vehicles. In: 2018 IEEE Intelligent Vehicles Symposium (IV). IEEE, Changshu, pp. 617–622. doi:10.1109/IVS.2018.8500440.

Khader, N., Yoon, S.W., 2018. Online control of stencil printing parameters using reinforcement learning approach. Procedia Manuf. 17, 94–101. doi:10.1016/j.promfg.2018.10.018.

Kuhnle, A., Jakubik, J., Lanza, G., 2019. Reinforcement learning for opportunistic maintenance optimization. Prod. Eng. 13 (1), 33–41. doi:10.1007/s11740-018-0855-7.

Kuhnle, A., Röhrig, N., Lanza, G., 2019. Autonomous order dispatching in the semiconductor industry using reinforcement learning. Procedia CIRP 79, 391–396. doi:10.1016/j.procir.2019.02.101.

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, 234–239. doi:10.1016/j.procir.2019.03.041.

Lingitz, L., Gallina, V., Ansari, F., Gyulai, D., Pfeiffer, A., Sihn, W., Monostori, L., 2018. Lead time prediction using machine learning algorithms: a case study by a semiconductor manufacturer. Procedia CIRP 72, 1051–1056. doi:10.1016/j.procir.2018.03.148.

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A.A., Veness, J., Bellemare, M.G., Graves, A., Riedmiller, M., Fidjeland, A.K., Ostrovski, G., Petersen, S., Beattie, C., Sadik, A., Antonoglou, I., King, H., Kumaran, D., Wierstra, D., Legg, S., Hassabis, D., 2015. Human-level control through deep reinforcement learning. Nature 518 (7540), 529–533. doi:10.1038/nature14236.

Mourtzis, D., Vlachou, E., 2018. A cloud-based cyber-physical system for adaptive shop-floor scheduling and condition-based maintenance. J. Manuf. Syst. 47, 179–198. doi:10.1016/j.jmsy.2018.05.008.

Ouelhadj, D., Petrovic, S., 2009. A survey of dynamic scheduling in manufacturing systems. J. Sched. 12 (4), 417–431. doi:10.1007/s10951-008-0090-8.

Peng, Z., Cui, D., Zuo, J., Li, Q., Xu, B., Lin, W., 2015. Random task scheduling scheme based on reinforcement learning in cloud computing. Cluster Comput. 18 (4), 1595–1607. doi:10.1007/s10586-015-0484-2.

Qu, S., Wang, J., Govil, S., Leckie, J.O., 2016. Optimized adaptive scheduling of a manufacturing process system with multi-skill workforce and multiple machine types: an ontology-based, multi-agent reinforcement learning approach. Procedia CIRP 57, 55–60. doi:10.1016/j.procir.2016.11.011.

Schneckenreither, M., Haeussler, S., 2019. Reinforcement learning methods for operations research applications: the order release problem. In: Nicosia, G., Pardalos, P., Giuffrida, G., Umeton, R., Sciacca, V. (Eds.), Machine Learning, Optimization, and Data Science, 11331. Springer International Publishing, Cham, pp. 545–559. doi:10.1007/978-3-030-13709-0_46. Series Title: Lecture Notes in Computer Science.

Singh, S.P., 1994. Reinforcement learning algorithms for average-payoff Markovian Decision Processes. In: Proceedings of the Twelfth National Conference on Artificial Intelligence (Vol. 1). American Association for Artificial Intelligence, USA, pp. 700–705. Event-place: Seattle, Washington, USA.

Stricker, N., Kuhnle, A., Sturm, R., Friess, S., 2018. Reinforcement learning for adaptive order dispatching in the semiconductor industry. CIRP Ann. 67 (1), 511–514. doi:10.1016/j.cirp.2018.04.041.

Sutton, R. S., Barto, A. G.,. Reinforcement learning: an introduction, 352.

Tsurumine, Y., Cui, Y., Uchibe, E., Matsubara, T., 2019. Deep reinforcement learning with smooth policy update: application to robotic cloth manipulation. Robot. Auton. Syst. 112, 72–83. doi:10.1016/j.robot.2018.11.004.

Vamplew, P., Dazeley, R., Berry, A., Issabekov, R., Dekker, E., 2010. Empirical evaluation methods for multiobjective reinforcement learning algorithms. Mach. Learn. 84 (1–2), 51–80. doi:10.1007/s10994-010-5232-5.

Wang, Y., Liu, H., Zheng, W., Xia, Y., Li, Y., Chen, P., Guo, K., Xie, H., 2019. Multi-objective workflow scheduling with deep-Q-network-based multi-agent reinforcement learning. IEEE Access 7, 39974–39982. doi:10.1109/ACCESS.2019.2902846.

Wang, Y.-C., Usher, J.M., 2005. Application of reinforcement learning for agent-based production scheduling. Eng. Appl. Artif.Intell. 18 (1), 73–82. doi:10.1016/j.engappai.2004.08.018.

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, 1264–1269. doi:10.1016/j.procir.2018.03.212.

Zhang, J., Ding, G., Zou, Y., Qin, S., Fu, J., 2019. Review of job shop scheduling research and its new perspectives under Industry 4.0. J. Intell. Manuf. 30 (4), 1809–1830. doi:10.1007/s10845-017-1350-2.